ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

LÊ ĐÌNH LONG

MỘT SỐ THUẬT TOÁN CHỌN LỌC VÀ ỨNG DỤNG TRONG TIN HỌC PHỔ THÔNG

Chuyên ngành: KHOA HỌC MÁY TÍNH
Mã số: 60 48 0101

LUẬN VĂN THẠC SĨ KHOA HỌC MÁY TÍNH

NGƯỜI HƯỚNG DẪN KHOA HỌC
TS. VŨ VINH QUANG

Thái Nguyên - 2015

MỞ ĐẦU

Thuật toán là một trong những khái niệm quan trọng nhất trong tin học. Thuật toán xuất phát từ nhà khoa học Arập Abu Ja'far Mohammed ibn Musa al Khowarizmi. Chúng ta có thể xem thuật toán là một công cụ dùng để giải bài toán được xác định trước. Việc nghiên cứu về thuật toán có vai trò rất quan trọng trong khoa học máy tính vì máy tính chỉ giải quyết được vấn đề khi đã có hướng dẫn giải rõ ràng và đúng đắn. Nếu hướng dẫn giải sai hoặc không rõ ràng thì máy tính không thể giải đúng được bài toán. Trong khoa học máy tính, thuật toán được định nghĩa là một dãy hữu hạn các thao tác được sắp xếp theo một trình tự nhất định sao cho sau khi thực hiện dãy thao tác ấy, từ input của bài toán, ta nhận được output cần tìm.

Ở Việt Nam môn Tin học được đưa vào giảng dạy chính thức ở trường phổ thông từ năm học 2006 - 2007 tuy nhiên trong thực tế môn Tin học đã được đưa vào tham gia thi học sinh giỏi cấp tỉnh, cấp quốc gia từ rất lâu: Hội thi Tin học trẻ không chuyên toàn quốc được tổ chức lần đầu vào năm 1995, kỳ thi học sinh giỏi Tin học quốc gia được tổ chức vào năm 1995 và đặc biệt kỳ thi Olympic Tin học quốc tế (IOI) tổ chức lần đầu vào năm 1989. Từ đó đến nay các kỳ thi học sinh giỏi, Olympic Tin học ngày một nhiều và đòi hỏi kiến thức rất cao.

Chúng ta biết rằng để có kết quả cao trong kỳ thi chọn học sinh giỏi môn Tin học nói chung thì học sinh phải có vốn kiến thức về thuật toán để giải được các bài toán khó (đặc biệt là các thuật toán nâng cao), sau đó học sinh sẽ sử dụng ngôn ngữ lập trình nào đó để lập trình dựa vào thuật toán đã tìm được và giải bài toán theo yêu cầu. Chương trình giảng dạy ở sách giáo khoa của môn Tin học hiện hành trong trường phổ thông có lượng kiến thức rất hạn chế và đơn giản, không đủ cơ sở để học sinh có thể dựa vào vốn kiến thức đó để tham gia một kỳ thi học sinh giỏi cấp thành phố hay cấp cao hơn. Câu hỏi đặt ra: ***"Làm thế nào để học sinh có thể đạt kết quả cao trong các kỳ thi học sinh giỏi môn Tin học trong trường phổ thông?"*** yêu cầu đặt ra là các giáo viên giảng dạy môn Tin học trong trường phổ thông phải suy nghĩ, tìm tòi tài liệu về một số thuật toán như: Thuật toán đệ quy**,** thuật toán

tham lam, thuật toán xấp xỉ và một số thuật toán trên đồ thị ... là những thuật toán sử dụng hiệu quả để giải nhiều bài toán Tin học.

Xuất phát từ thực tế đó, đề tài luận văn: **"MỘT SỐ THUẬT TOÁN CHỌN LỌC VÀ ỨNG DỤNG TRONG TIN HỌC PHỔ THÔNG"** với mục đích tìm hiểu, nghiên cứu một số thuật toán và cách ứng dụng vào giảng dạy, bồi dưỡng đội tuyển học sinh giỏi môn Tin học ở trường phổ thông.

Nội dung chính của luận văn gồm 3 chương, phần phụ lục với các nội dung chính như sau:

Chương 1: *Luận văn trình bày tổng quan về các khái niệm cơ bản về thuật toán và độ phức tạp của thuật toán, vấn đề phân lớp các bài toán trên cơ sở đánh giá độ phức tạp của thuật toán. Các kiến thức này sẽ là nền tảng về mặt lý thuyết tính toán để nghiên cứu các chương tiếp sau của luận văn.*

Chương 2: *Trong chương này luận văn trình bày tổng quan về thuật toán đệ quy, thuật toán tham lam, thuật toán xấp xỉ và một số thuật toán trên mô hình đồ thị.*

Chương 3: *Dựa vào cơ sở lý thuyết của thuật toán được trình bày ở chương 2, trong chương này luận văn sẽ cài đặt chương trình cho một số bài toán cụ thể.*

Phần phụ lục:

Toàn bộ các kết quả thực nghiệm giải các bài toán được cài đặt bằng ngôn ngữ Pascal version 7.0 trên máy tính PC.

Chương 1
CÁC KHÁI NIỆM VỀ THUẬT TOÁN VÀ ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

1.1 Khái niệm cơ bản về thuật toán

1.1.1 Khái niệm bài toán Tin học

Trong phạm vi tin học, người ta quan niệm ***bài toán là một công việc nào đó muốn máy tính thực hiện*** [2].

Khi dùng máy tính để giải bài toán, ta cần quan tâm tới 2 vấn đề: Dữ liệu cần được đưa vào máy tính (Input) là gì? và cần lấy ra (Output) thông tin gì? nói một cách khác, cho một bài toán là việc mô tả rõ input và output của bài toán.

Vấn đề còn lại là: Làm thế nào để từ input ta có được output?

1.1.2 Khái niệm thuật toán

Khác với toán học (các yêu cầu của bài toán thường là chứng minh sự tồn tại đáp án chứ không yêu cầu tìm một cách chi tiết để tìm ra đáp số đó), giải một bài toán Tin học là việc đi tìm một lời giải cụ thể, tường minh để đưa ra output của bài toán dựa trên input đã cho. Việc chỉ ra một cách tìm output của bài toán gọi là một thuật toán. Có nhiều cách phát biểu khái niệm về thuật toán. Dưới đây là cách phát biểu được chọn để đưa vào sách giáo khoa Tin học phổ thông.

Khái niệm về thuật toán: ***Thuật toán là một dãy hữu hạn các thao tác được sắp xếp theo một trình tự nhất định để sau khi thực hiện dãy các thao tác đó, từ input ta có output cần tìm*** [2].

Trong lĩnh vực khoa học máy tính, cụm từ "thuật toán" đôi khi còn được gọi là: "giải thuật".

Ví dụ 1: Thuật toán tô màu đồ thị

- Input: đồ thị G = (V, E).
- Output: đồ thị G = (V, E) có các đỉnh đã được gán màu.

Thuật toán: Có nhiều cách để mô tả thuật toán khác nhau. Dưới đây là cách mô tả thuật toán dạng liệt kê các bước:

Bước 1: Lập danh sách các đỉnh của đồ thị $E' := [v_1, v_2, ..., v_n]$ được sắp xếp theo thứ tự bậc giảm dần: $d(v_1) \geq d(v2) \geq ... d(v_n)$

Đặt i := 1;

Bước 2: Tô màu i cho đỉnh đầu tiên trong danh sách. Duyệt lần lượt các đỉnh tiếp theo và tô màu i cho đỉnh không kề đỉnh đã được tô màu i.

Bước 3: Nếu tất cả các đỉnh đã được tô màu thì kết thúc, đồ thị được tô bằng i màu. Ngược lại, chuyển sang bước 4;

Bước 4: Loại khỏi E' các đỉnh đã tô màu. Sắp xếp lại các đỉnh trong E' theo thứ tự bậc giảm dần.

Đặt i := i +1 và quay lại bước 2.

1.2 Yêu cầu của thuật toán

Thuật toán phải đảm bảo được các yêu cầu sau đây [2], [4].

1. Tính xác định: Các bước của thuật toán phải được trình bày rõ ràng, mạch lạc, đảm bảo cho người đọc chỉ hiểu theo một nghĩa duy nhất.

2. Tính khả thi: Thuật toán phải thực hiện được, nghĩa là ta có thể sử dụng máy tính kết hợp giữa các ngôn ngữ lập trình để thể hiện thuật toán hay có thể kiểm tra thuật toán chỉ bằng giấy và bút (còn gọi là Test).

3. Tính dừng: Nếu dữ liệu vào thỏa mãn điều kiện đầu vào thì thuật toán phải kết thúc và cho ra kết quả sau một số hữu hạn bước.

4. Tính chính xác (tính đúng đắn): Thuật toán phải cho kết quả chính xác và thể hiện đúng đắn trên cơ sở toán học.

5. Tính tối ưu: Thuật toán phải có chi phí về không gian bộ nhớ ít nhất và chạy trong thời gian nhanh nhất.

1.3. Thể hiện thuật toán

Thuật toán được thể hiện bằng một trong các cách sau

- Sử dụng liệt kê các bước.
- Sử dụng lưu đồ (sơ đồ khối).
- Sử dụng ngôn ngữ lập trình.

1.4 Độ phức tạp của thuật toán

1.4.1 Chi phí phải trả cho một quá trình tính toán

Chi phí phải trả cho một quá trình tính toán bao gồm chi phí về không gian (bộ nhớ - số ô nhớ cần sử dụng trong quá trình tính toán) và chi phí về thời gian (thời gian cần sử dụng cho một quá trình tính toán).

Nếu cho một thuật toán A. Thuật toán này thực hiện trên bộ dữ liệu e.

$\Rightarrow$ Thuật toán này phải trả 2 giá: giá về không gian là $L_A(e)$, giá về thời gian là $T_A(e)$, e là bộ dữ liệu vào.

Ví dụ 2: Xét thuật toán A, "Tìm số lớn nhất trong một dãy số".

```
Begin
    Max := x1;
    For i := 2 to n do
    If max < xi then max := xi ;
End.
```

Thực hiện A trên hai bộ dữ liệu khác nhau:

+ Bộ dữ liệu $e_1 = \{0, 4, 9, 5, 7, 6\}$:

Khi đó $L_A(e_1) = 7$ (dữ liệu vào) + 2 (biến trung gian) = 9.

$T_A(e_1) = 8$ (thời gian để thực hiện tất cả các phép tính cơ bản).

+ Bộ dữ liệu $e_2 = \{3, 4, 6, 7, 9, 10, 12, 15\}$:

$L_A(e_2) = 11$.

$T_A(e_2) = 15$.

Khi đó ta có các khái niệm về chi phí phải trả trong các trường hợp như sau:

- Chi phí phải trả trong trường hợp xấu nhất:

  - Chi phí xấu nhất về bộ nhớ: $L_A(n) = \text{Max}\ \{L_A(e) \mid e \leq n\}$

  - Chi phí xấu nhất về thời gian: $T_A(n) = \text{Max}\ \{T_A(e) \mid e \leq n\}$

- Chi phí phải trả trung bình:

Là tổng số các chi phí khác nhau ứng với các bộ số liệu chia cho tổng số số bộ số liệu.

- Chi phí phải trả tiệm cận:

Đó là biểu thức biểu diễn tốc độ tăng của chi phí thực tế phải trả. Nó có gía trị tiệm cận với chi phí thực tế.

- Nhận xét: Ngày nay do sự phát triển không ngừng của khoa học công nghệ kỹ thuật điện tử nên chi phí về bộ nhớ không còn là vấn đề cần thiết phải bàn tới mà ta chỉ quan tâm tới chi phí phải trả về thời gian thực hiện giải thuật. Từ đây ta chỉ xét đến thời gian thực hiện giải thuật T(n), hay đó chính là độ phức tạp của thuật toán.

Sau đây là việc phân tích thời gian thực hiện giải thuật, một trong các tiêu chuẩn quan trọng để đánh giá hiệu lực của giải thuật vốn hay được đề cập tới.

1.4.2. Phân tích thời gian thực hiện giải thuật

Với một bài toán, không chỉ có một giải thuật. Chọn một giải thuật đưa tới kết quả nhanh là một đòi hỏi thực tế. Nhưng căn cứ vào đâu để có thể nói được giải thuật này nhanh hơn giải thuật kia?

Có thể thấy ngay: thời gian thực hiện một giải thuật, (hay chương trình thể hiện một giải thuật đó) phụ thuộc vào rất nhiều yếu tố. Một yếu tố cần chú ý trước tiên đó là *kích thước của dữ liệu đưa vào.* Chẳng hạn thời gian sắp xếp một dãy số phải chịu ảnh hưởng của số lượng các số thuộc dãy số đó. Nếu gọi n là số lượng này (kích thước của dữ liệu vào) thì thời gian thực hiện T của một giải thuật phải được biểu diễn như một hàm của n: T(n).

Các kiểu lệnh và tốc độ xử lý của máy tính, ngôn ngữ viết chương trình và chương trình dịch ngôn ngữ ấy đều ảnh hưởng tới thời gian thực hiện; nhưng những yếu tố này không đồng đều với mọi loại máy trên đó cài đặt giải thuật, vì vậy không thể đưa chúng vào khi xác lập T(n). Điều đó có nghĩa là T(n) không thể được biểu diễn thành đơn vị thời gian bằng giây, bằng phút được. Tuy nhiên, không phải vì thế mà không thể so sánh được các giải thuật về mặt tốc độ. Nếu như thời gian thực hiện của một giải thuật là $T_1(n) = c.n^2$ và thời gian thực hiện một giải thuật khác là

$T_2(n) = k.n$ (với c và k là một hằng số nào đó), thì khi n khá lớn, thời gian thực hiện giải thuật T_2 rõ ràng ít hơn so với thời gian thực hiện giải thuật T_1. Như vậy, nếu nói thời gian thực hiện giải thuật bằng T(n) tỉ lệ với n^2 hay tỉ lệ với n cũng cho ta ý niệm về tốc độ thực hiện giải thuật đó khi n khá lớn (với n nhỏ thì việc xét T(n) không có ý nghĩa). Cách đánh giá thời gian thực hiện giải thuật độc lập với máy tính và các yếu tố liên quan với máy như vậy sẽ dẫn tới khái niệm về cấp độ lớn của thời gian thực hiện giải thuật hay còn gọi là độ phức tạp tính toán của giải thuật.

1.4.3 Độ phức tạp của thuật toán

Nếu thời gian thực hiện một giải thuật là $T(n) = c.n^2$ (với c là hằng số) thì ta nói: Độ phức tạp tính toán của giải thuật này có cấp là n^2 (hay cấp độ lớn – tốc độ tăng – của thời gian thực hiện giải thuật là n^2) và ký hiệu là:

$T(n) = O(n^2)$ (kí hiệu chữ O lớn). Một cách tổng quát có thể định nghĩa:

Một hàm f(n) được xác định là O(g(n))

f(n) = O(g(n)) và được gọi là có cấp g(n) nếu tồn tại các hằng số c và n_0 sao cho $f(n) \leq c.g(n)$ khi $n \geq n_0$ nghĩa là f(n) bị chặn trên bởi một hằng số nhân với g(n), với mọi giá trị của n từ một điểm nào đó. Thông thường các hàm thể hiện độ phức tạp tính toán của giải thuật có dạng: $O(\log_2 n)$, O(n), $O(n\log_2 n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, O(n!), $O(n^n)$.

O(g(n)) còn gọi là độ phức tạp tiệm cận của hàm f(n).

Dưới đây là một số hàm số hay dùng để ký hiệu độ phức tạp tính toán và bảng giá trị của chúng để tiện theo dõi sự tăng của hàm theo đối số n.

$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	2.147.483.648

Các hàm như 2^n, n^n được gọi là *hàm loại mũ*, ngoài ra còn có hàm n! và một số hàm khác có độ phức tạp lớn hơn các hàm mũ. Một giải thuật mà thời gian thực hiện của nó có cấp là các hàm loại mũ thì tốc độ rất chậm. Các như n^3, n^2, $n\log_2 n$, $\log_2 n$ được gọi là các *hàm loại đa thức*. Giải thuật với thời gian thực hiện có cấp hàm đa thức thì thường hiệu quả và chấp nhận được.

Ví dụ 3: Tính giá trị đa thức $P(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ *với* $a_0, a_1, ..., a_n$, x nhập từ bàn phím.

Thuật toán 1:

```
1. Input n, a_o, a_1, a_2, ..., a_n, x;
2. S := a_o;
3. for i := 1 to n do
   begin
      p := 1;
      for j := 1 to i do  p := p*x;
      S:= S + a_i*p;
   end;
4. Output s;
```

Với mỗi giá trị i của vòng lặp 3, vòng lặp 3.2 thực hiện i vòng lặp nên khi n = i nó thực hiện đủ n vòng lặp. Vậy vòng lặp 3 thực hiện $\frac{n(n-1)}{2}$ lần câu lệnh sau do nên thời gian tính toán tỉ lệ thuận với n^2.

Vậy độ phức tạp tính toán của thuật toán trên là $O(n^2)$.

Thuật toán 2: Vì $x^n = x*x^{n-1}$ nên có thể tận dụng kết quả của lần tính trước cho lần tính sau:

```
1. Input n, a_o, a_1, a_2, ..., a_n, x;
2. S := a_o;  p:=1;
3. for i := 1 to n do
   begin
        p := p* x;
```

```
        S := s + p;
    end;
4.  Output S;
```

Hai lệnh 2 và 4 đều có độ phức tạp tính toán là O(1). Vòng lặp 3 cần thực hiện n lần hai thao tác tính s và p. Vậy số lần thực hiện lệnh 3 là 2n. Do vậy, độ phức tạp tính toán của thuật toán trên là O(n).

1.4.4. Các qui tắc xác định độ phức tạp tính toán của giải thuật

Xác định độ phức tạp tính toán của một giải thuật bất kì có thể dẫn tới những bài toán phức tạp. Tuy nhiên, trong thực tế, đối với một số giải thuật ta cũng có thể phân tích được bằng một số quy tắc đơn giản như [2], [4]:

- *Quy tắc tổng*

Giả sử $T_1(n)$ và $T_2(n)$ là thời gian thực hiện của 2 đoạn chương trình P_1 và P_2 mà $T_1(n) = O(f(n))$; $T_2(n) = O(g(n))$ thì thời gian thực hiện P_1 rồi P_2 tiếp theo sẽ là:

$T_1(n) + T_2(n) = O(\max(f(n), g(n)))$.

Ví dụ, trong một chương trình có 3 bước thực hiện mà thời gian thực hiện từng bước lần lượt là $O(n^2)$, $O(n^3)$ và $O(\log_2 n)$ thì thời gian thực hiện 2 bước đầu là $O(\max(n^2, n^3)) = O(n^3)$.

Một ứng dụng khác của quy tắc này là nếu $g(n) \leq f(n)$ với mọi $n \geq n_o$ thì $O(f(n) + g(n))$ cũng là $O(f(n))$. Chẳng hạn: $O(n^4+n^2) = O(n^4)$ và $O(n+\log_2 n) = O(n)$.

- *Quy tắc nhân*

Nếu tương ứng với P_1 và P_2 là $T_1(n) = O(f(n))$; $T_2(n) = O(g(n))$ thì thời gian thực hiện P_1 và P_2 lồng nhau sẽ là: $T_1(n).T_2(n) = O(f(n).g(n))$.

Ví dụ, câu lệnh gán: x := x+1 có thời gian thực hiện bằng c (hằng số) nên được đánh giá là O(1).

Câu lệnh: for i := 1 to n do x := x+1; có thời gian thực hiện O(n.1)=O(n).

Câu lệnh:

```
for i := 1 to n do
        for j := 1 to n do x := x+1;
```

có thời gian thực hiện được đánh giá là: $O(n.n) = O(n^2)$. Có thể suy ra $O(c.f(n)) = O(f(n))$ chẳng hạn $O(n^2/2) = O(n^2)$.

* Chú ý: *Phép toán tích cực* (active operation) đó là phép toán thuộc giải thuật mà thời gian thực hiện nó không ít hơn thời gian thực hiện các phép khác (tất nhiên các phép toán tích cực không phải là duy nhất) hay nói một cách khác: số lần thực hiện nó không kém gì các phép khác. Thông thường đó là các phép toán cộng, trừ, nhân, chia và các phép so sánh.

- Quy tắc tổng quát

1. Thời gian thực hiện mỗi câu lệnh Gán, Read, Write là O(1).

2. Thời gian thực hiện mỗi chuỗi tuần tự các câu lệnh được tính theo quy tắc cộng.

3. Thời gian thực hiện cấu trúc *if* (điều kiện) *then* ... được tính bằng thời gian thực hiện câu lệnh sau *then* hoặc sau *else*. Còn câu lệnh điều kiện thường là O(1).

4. Thời gian thực hiện vòng lặp được tính là tổng trên tất cả số lần lặp thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp là hằng số thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

Ví dụ 4: Giải thuật toán tính giá trị của e^x:

$e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$ với x và n cho trước.

```
Program EXP1;        {tính từng số hạng rồi cộng lại}
1. Read(x); S :=1;
2. for i:=1 to n do
   Begin
      P:=1;
      for j:=1 to i do p:= p*x / j;
      S:=s+p;
   End;
3. End.
```

Phép toán tích cực ở đây là phép p := $\frac{p*x}{j}$.

Ta thấy nó được thực hiện $\frac{n*(n-1)}{2}$ lần.

Vậy thời gian thực hiện giải thuật này được đánh giá là T(n) = O(n^2).

1.5. Phân lớp các bài toán dựa trên độ phức tạp của thuật toán

Khi cho một bài toán, có hai khả năng xảy ra là: bài toán không giải được hoặc bài toán giải được. Trong thực tế có rất nhiều các bài toán không thể giải trong thời gian đa thức. Ví dụ bài toán treo (Halting Problem) nổi tiếng của Turing không thể giải bất kỳ máy tính nào, bất kể cung cấp bao nhiêu thời gian. Cũng có các bài toán có thể giải được, nhưng không phải trong thời gian đa thức O(n^k) với một hằng k. Nói chung, ta xem các bài toán có thể giải được bằng các thuật toán thời gian đa thức là "dễ trị", và các bài toán yêu cầu thời gian siêu đa thức là "khó trị".

Vì độ phức tạp giải thuật đối với mỗi bài toán là khác nhau thông qua thời gian đa thức và siêu đa thức, trên cơ sở đó các bài toán cũng được phân chia thành các lớp thông qua độ phức tạp thuật toán (đa thức hay hàm mũ). Đó là các lớp P, NP, NPC được định nghĩa như sau [1], [11].

1.5.1. Lớp P

Lớp P (Polynomial time – thời gian đa thức) là lớp các bài toán dễ, có thể giải được bằng thuật toán đơn định đa thức.

1.5.2. Lớp NP

Lớp NP (Nondeterministic Polynomial – thời gian đa thức không tất định) là lớp các bài toán có thể giải được bằng các thuật toán không đơn định đa thức.

Nhiều giả thiết đặt ra rằng liệu lớp P và lớp NP có đồng nhất với nhau hay không? Điều đó đang còn là vấn đề mở chưa được làm sáng tỏ. Bởi trong NP vẫn tồn tại lớp các bài toán không giải được bằng các thuật toán đa thức, đó chính là sự có mặt của lớp NPC. Như vậy, chúng ta đang chấp nhận P $\subseteq$ NP.

1.5.3. Lớp NPC

Để định nghĩa lớp NPC, dựa vào các khái niệm sau:

- Khái niệm dẫn về được: Bài toán B được gọi là dẫn về được bài toán A một cách đa thức nếu có một thuật toán đơn định đa thức để giải bài toán A thì cũng có một thuật toán đơn định đa thức để giải bài toán B. ký hiệu $B \propto A$.

Khi đó bài toán A khó hơn bài toán B hay còn gọi B dễ hơn A hay B là trường hợp riêng của A.

Quan hệ $\propto$ có tính bắc cầu: $B \propto C, C \propto A \Rightarrow B \propto A$.

- Khái niệm *khó tương đương*: Bài toán A được gọi là khó tương đương bài toán B nếu như $A \propto B$ và $B \propto A$. Ký hiệu $A \sim B$.

- Bài toán NP – khó (NP hard):

  Bài toán A được gọi là NP – khó nếu có bài toán $L \propto A$ với $\forall L \in NP$.

- Bài toán NP đầy đủ

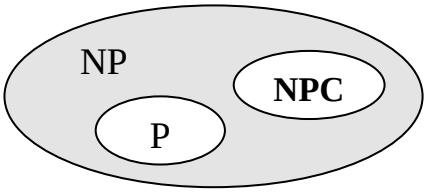
Bài toán A được goi là NP đầy đủ (NP-Complate) nếu:

$$\begin{cases} A & \text{là NP - khó} \\ A & \in NP \end{cases}$$

- Lớp NPC

Lớp NPC là lớp các bài toán NP đầy đủ, có độ phức tạp hàm mũ. Qua đó cho thấy: $P \cap NPC = Ø$.

Hình 1.1 minh họa các lớp P, NP, NPC dưới dạng tập hợp.



Hình 1.1: Các lớp P, NP và NPC

KẾT LUẬN CHƯƠNG 1

Chương này luận văn trình bày những khái niệm cơ bản về bài toán - thuật toán trong tin học, khái niệm về độ phức tạp của thuật toán và các nguyên tắc đánh giá độ phức tạp. Trên cơ sở đó đưa ra nguyên tắc phân lớp các bài toán để tiến hành lựa chọn giải thuật tốt nhất giải các lớp bài toán trong thực tế. Các kiến thức này đã được tham khảo trong các tài liệu [2], [3], [4], [7].

Chương 2
MỘT SỐ THUẬT TOÁN CHỌN LỌC VÀ ỨNG DỤNG

2.1 Thuật toán đệ quy

2.1.1 Khái niệm đệ quy

Đệ quy (trong Tiếng Anh là recursion) là phương pháp dùng trong các chương trình máy tính trong đó có một hàm tự gọi chính nó [2], [7]

Một khái niệm X được định nghĩa theo đệ quy nếu trong định nghĩa X có sử dụng ngay chính khái niệm X.

Ví dụ 5: Để định nghĩa về số nguyên, người ta định nghĩa như sau

- Số 1 là số nguyên
- Nếu n >1 là số nguyên thì (n+1) cũng là số nguyên

Ví dụ 6: Để định nghĩa về số n!, người ta định nghĩa

- Nếu n = 0 thì n! = 1
- Nếu n > 0 thì n! = n*(n-1)!

Ví dụ 7: Để định nghĩa về cây, người ta định nghĩa

- R là gốc cây
- Cây là hợp của các tập hợp $T_1,T_2,..,T_n$ trong đó các T_i cũng là cây

Trong các định nghĩa trên đều yêu cầu 2 vấn đề quan trọng

1. Luôn luôn tồn tại 1 trường hợp đặc biệt không định nghĩa được phải công nhận (1 là số nguyên, 0!=1 hay R là gốc)
2. Luôn dùng khái niệm cấp thấp hơn để định nghĩa ra khái niệm cấp cao hơn

+ Thuật toán đệ quy là một thuật toán mà khi thiết kế nó, ta dùng chính nó để thiết kế ra nó, khi thực hiện nó thì sẽ tồn tại lời gọi đến chính nó.

Ví dụ 8: Xuất phát từ định nghĩa n! Ta có thể thiết kế 1 thuật toán đệ quy như sau:

```
Function giaithua(n:integer);
  Begin
      If  n = 0  then giaithua = 1
         else
```

```
        giaithua = n*giaithua(n-1);
    End.
```

Như vậy đặc điểm của thuật toán đệ quy đòi hỏi

1. Phải có định nghĩa đệ quy
2. Điểm dừng của thuật toán chính là trường hợp đặc biệt không định nghĩa được.
3. Tồn tại lời gọi tới chính bản thân nó theo đúng định nghĩa.

Ví dụ 9: Định nghĩa dãy Fibonaci

B(n) = 1 nếu n = 1 hoặc n= 2 - Đây là trường hợp đặc biệt phải công nhận

B(n) = B(n-1) + B(n-2) - Đây là định nghĩa theo đệ quy

Khi đó ta sẽ có thuật toán

```
Function B(n:integer);
  Begin
     if (n=1) or (n=2) then B=1
       else B=B(n-1)+B(n-2);
  End.
```

2.1.2 Giải thuật đệ quy và thủ tục đệ quy:

Một thủ tục gọi là đệ quy nếu trong quá trình thực hiện nó phải gọi đến chính nó nhưng với kích thước nhỏ hơn của tham số.

- Cách xây dựng hàm, thủ tục đệ quy thường được viết theo thuật toán sau:

```
IF trường hợp suy biến THEN
  Begin
      trình bày cách giải bài toán
  End
       else
    Begin
      gọi đệ quy tới hàm, thủ tục đang xây dựng với bộ tham số mới
    End;
```

Ví dụ 10: Để lập hàm tính giai thừa của một số nguyên không âm ta có thể làm theo 2 cách như sau:

Cách 1: Hàm tính n giai thừa dùng theo đệ quy

```
Function Giaithua ( n:longint ) : longint;
  begin
     if n = 0 then giaithua :=1
   else giaithua := n * giaithua ( n-1);
end;
```

Cách 2: Hàm tính n giai thừa bằng cách dùng vòng lặp for:

```
Function Giaithua( n: longint) : longint;
  Var       i, GT : longint;
  Begin
     GT := 1;
     If  n > 0 then
        For i:= 1 to n do GT := GT * i;
     Giaithua := GT;
   End;
```

Ví dụ 11: Xét bài toán tính tổng:

f(a,n)= a[1] + a[2] + …+ a[n]

+ Trường hợp suy biến là trường hợp n = 1, khi đó: f(a,n) = a[1]

+ Trường hợp tổng quát n>1, bài toán quy về việc tính tổng của (n-1) phần tử như sau:

f(a,n) = (a[1] + … +a[n-1]) + a[n] = f(a, n-1) + a[n]

Hàm đệ quy tính tổng được viết như sau:

```
Type   DS = Array [1..100] of Real;
Function TongDS (a: DS; n: Integer) : Real;
  Begin
     If  n = 1 then
          TongDS := a[1]
```

```
        Else
            TongDS := TongDS (a, n-1) + a[n];
    End;
```

2.1.3 Cấu trúc và đặc điểm của đệ quy.

- Cấu trúc: Gồm 2 phần

+ *Phần cơ sở:* chứa các tác động của hàm hoặc thủ tục với một số giá trị cụ thể ban đầu của tham số.

+ *Phần đệ quy:* định nghĩa tác động cần được thực hiện cho giá trị hiện thời của các tham số bằng các tác động đã được định nghĩa trước đây với kích thước tham số nhỏ hơn.

- Đặc điểm:

+ Trong thủ tục đệ quy có lời gọi đến chính nó.

+ Mỗi lần có lời gọi lại thủ tục thì kích thước của bài toán thu nhỏ hơn trước.

+ Sử dụng đệ quy là một phương pháp làm cho chương trình ngắn gọn, dễ hiểu nhưng nó sẽ làm tốn bộ nhớ và thời gian nếu như cấu trúc hàm đệ quy "phức tạp".

2.1.4 Một số bài toán thường gặp trong đệ quy:

Thực tế có rất nhiều bài toán có sử dụng giải thuật đệ quy như bài toán tính giai thừa của n! hay bài toán tính giá trị của dãy Fibonacci … Trong luận văn chỉ đi sâu vào một vài bài toán điển hình nhất trong đó đã đưa ra được bản chất nổi bật nhất của đệ quy.

Ví dụ 12: Bài toán Tháp Hà Nội.

♦ Bài toán: Có 3 cái cọc , đánh dấu A, B, C và N cái đĩa. Mỗi đĩa đều có một lỗ chính giữa để đặt xuyên qua cọc, các đĩa đều có kích thước khác nhau. Ban đầu tất cả các đĩa đều được đặt ở cọc thứ nhất theo thứ tự đĩa nhỏ hơn ở trên.

Yêu cầu của bài toán là chuyển tất cả các đĩa từ cọc A qua cọc C với 3 ràng buộc như sau:

1. Mỗi lần chỉ chuyển được một đĩa.
2. Trong quá trình chuyển đĩa có thể dùng cọc còn lại (B) để làm cọc trung

gian.

3. Chỉ cho phép đặt đĩa có bán kính nhỏ hơn lên đĩa có bán kính lớn hơn.

♦ Phân tích bài toán:

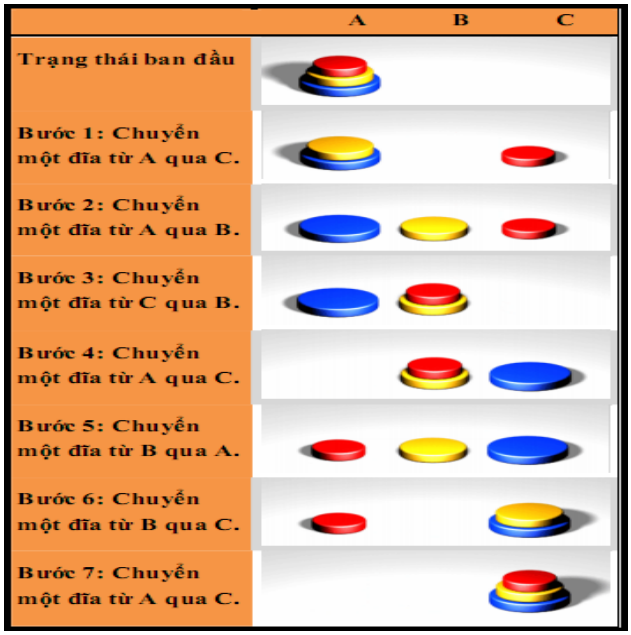
Trong bài toán trên hình dung một lời giải tổng quát cho trường hợp tổng quát N là không dễ dàng.

Hãy bắt đầu với các trường hợp đơn giản sau:

- Với N = 1: Chỉ cần chuyển đĩa này từ cọc A qua cọc C là xong.

- Với N = 2: Để đảm bảo ràng buộc thứ hai ta bắt buộc chuyển đĩa trên cùng từ cọc A qua cọc B. Chuyển tiếp đĩa còn lại từ cọc A qua cọc C. Chuyển tiếp đĩa đang ở cọc B sang cọc C.

- Với N = 3: ta phải thực hiện 7 bước như sau:



Hình 2.1 Mô tả bước chuyển của đĩa

♦ Nhận xét:

Ở kết quả của bước thứ ba. Đây là một kết quả quan trọng vì nó cho ta thấy từ trường hợp N = 3 bài toán đã được phân chia thành hai bài toán với kích thước nhỏ hơn. Đó là bài toán chuyển 1 đĩa từ cọc A qua cọc C lấy cọc B làm trung gian và bài toán chuyển 2 đĩa (dời) từ cọc B sang cọc C lấy cọc A làm trung gian. Hai bài toán con này đã biết cách giải (trường hợp N = 1 và trường hợp N = 2).

+ Nhận xét đó cho ta gợi ý trong trường hợp tổng quát:

Bước 1: Dời (N-1) đĩa trên cùng từ cọc A sang cọc B lấy cọc C làm trung gian.

Bước 2: Chuyển 1 đĩa dưới cùng từ cọc C.

Bước 3: Dời (N-1) đĩa đang ở cọc B sang cọc C lấy cọc A làm trung gian.

Như vậy, bài toán đối với N đĩa ở trên được "đệ quy" về hai bài toán (N-1) đĩa và bài toán 1 đĩa. Quá trình đệ qui sẽ dừng lại khi N = 0 (không còn đĩa để dời hoặc chuyển).

- Giải thuật đệ quy cho bài toán tháp Hà Nội:

Procedure dich_chuyen(n, A, B, C);

1. **if** n = 1 **then** chuyển đĩa từ A sang C
2. **else**
   begin
   dich_chuyen(n-1, A, C, B);
   dich_chuyen(1, A, B, C);
   dich_chuyen(n-1, B, A, C)
   end
3. **return**

2.2 Thuật toán tham lam

2.2.1 Tổng quan về thuật toán tham lam

2.2.1.1 Thuật toán tham lam là gì?

Thuật toán tham lam (Tiếng Anh: Greedy algorithms) là một thuật toán giải quyết một số bài toán theo kiểu metaheuristic để tìm kiếm lựa chọn tối ưu địa phương ở mỗi bước đi với hy vọng tìm được tối ưu toàn cục [1], [3].

2.2.1.2 Đặc điểm của thuật toán tham lam

Mục đích của thuật toán tham lam là xây dựng bài toán giải nhiều lớp bài toán khác nhau, đưa ra quyết định dựa ngay vào thuật toán đang có, và trong tương lai sẽ không xem xét lại quyết định trong quá khứ.

- Ưu điểm của thuật toán tham lam

  - Dễ đề xuất.

  - Thời gian tính nhanh.

- Đặc điểm

Lời giải bài toán là một tập hữu hạn S các phần tử thỏa mãn điều kiện nào đó, ta phải giải quyết bài toán một cách tối ưu. Nói cách khác nghiệm S phải được xây dựng sao cho hàm mục tiêu f(S) có giá trị tốt nhất (lớn nhất, nhỏ nhất) có thể.

Các bước giải bài toán như sau

- Có một tập các ứng cử viên C để chọn cho các thành phần của nghiệm tại mỗi bước.

- Xuất phát từ lời giải rỗng S, tại mỗi bước của thuật toán, ta sẽ lựa chọn một ứng cử viên trong C để bổ sung vào lời giải S hiện có.

- Xây dựng được hàm Seclect(C) tại mỗi bước chọn để lựa chọn một ứng cử viên có triển vọng nhất để đưa vào lời giải S.

- Xây dựng được hàm Feasible(S$\cup$x) để kiểm tra tính chấp nhận được ứng cử viên x khi đưa vào tập nghiệm S.

- Cuối cùng khi có được tập S, xây dựng hàm Soluition(S) để kiểm tra tính chấp nhận được của lời giải S.

2.2.1.3 Điều kiện để một bài toán áp dụng được giải thuật tham lam

Các dạng bài toán tìm phương án tối ưu như bài toán người du lịch, bài toán cái túi … Chúng thuộc lớp các bài toán tối ưu tổ hợp là một trường hợp riêng của bài toán tối ưu.

Các bài toán tối ưu tổ hợp có rất nhiều ứng dụng trong thực tiễn và việc ứng dụng trở nên tốt hơn rất nhiều khi người ta nghiên cứu các thuật toán tối ưu và cài đặt trên máy tính.

Một trong những thuật toán để giải quyết các bài toán trên là thuật toán tham lam.

Thuật toán tham lam (Greedy Algorithms) được dùng để giải quyết các bài toán mà chúng ta có thể quyết định đâu là lựa chọn tốt nhất.

Nếu có thể chứng minh rằng một thuật toán tham lam cho ra kết quả tối ưu toàn cục cho một lớp bài toán nào đó, thì thuật toán thường sẽ trở thành phương pháp được chọn lựa, vì nó chạy nhanh hơn các phương pháp tối ưu hóa khác như quy hoạch động. Tuy nhiên trong một số trường hợp thuật toán tham lam chỉ cho nghiệm gần đúng với nghiệm tối ưu.

2.2.1.4 Những dạng bài toán thường dùng thuật toán tham lam để giải

- Các thuật toán tham lam chủ yếu để giải quyết các bài toán tối ưu.

- Các bài toán tối ưu là các bài toán có dạng tổng quát như sau

1. Hàm f(x) được gọi là hàm mục tiêu, xác định trên một tập hữu hạn các phần tử D.

2. Mỗi phần tử X thuộc D có dạng $X=(x_1, x_2,\ldots,x_n)$ được gọi là một phương án.

3. Tìm một phương án X_0 thuộc D sao cho f(x) đạt giá trị lớn nhất hoặc đạt giá trị nhỏ nhất trên D. Thì X_0 được gọi là phương án tối ưu.

4. Tập D được gọi là tập các phương án của bài toán.

Ví dụ như các dạng bài toán sau

- Một tập các đối tượng

- Một dãy các đối tượng đã lựa chọn

- Một hàm để xem một tập các đối tượng có lập thành một giải pháp hay không (không nhất thiết tối ưu)

- Một hàm để xem một tập đối tượng có là tiềm năng hay không

- Một hàm để lựa chọn ứng viên có triển vọng nhất

- Một hàm đích cho một giá trị của một giải pháp (để tối ưu hóa)

2.2.2 Vấn đề thiết kế thuật toán

2.2.2.1 Các thành phần của thuật toán tham lam

Xét bài toán chọn hoạt động, ta định nghĩa bài toán con S_{ij} với $i \neq j$, nếu luôn thực hiện lựa chọn tham lam ta có thể giới hạn các bài toán con được thành lập bởi $S_{i,n+1}$

Như một sự lựa chọn, ta có thể tạo nên cấu trúc con tối ưu với một lựa chọn tham lam có nghĩa. Điều đó có nghĩa là, có thể bỏ qua chỉ số dưới thứ hai và định nghĩa các bài toán con của công thức $s_i = \{a_k \in S | f_i \le s_k\}$. Sau đó, chứng minh rằng một lựa chọn tham lam (hoạt động đầu tiên a_m để kết thúc s_i) kết hợp với một giải pháp tối ưu để đi đến tập còn lại S_m của các hoạt động tương thích, mang lại một giải pháp tối ưu đối với S_i. Một cách tổng quát, thuật toán tham lam được thiết kế theo các bước:

1. Tìm lựa chọn sao cho bước tiếp theo chỉ giải quyết một bài toán con.

2. Chứng minh với sự lựa chọn tham lam tại mỗi bước ta luôn tìm được một giải pháp tối ưu của bài toán ban đầu.

3. Chỉ ra rằng với sự lựa chọn tham lam tại mỗi bước, giải pháp tối ưu của bài toán con còn lại kết hợp với sự lựa chọn tham lam này sẽ đi đến một giải pháp tối ưu cho bài toán ban đầu.

Không có cách tổng quát cho một thuật toán tham lam giải quyết một bài toán tối ưu, nhưng chiến lược lựa chọn tham lam và cấu trúc con tối ưu là hai thành phần then chốt, thực tế đã chứng minh rằng các bài toán có 2 thuộc tính này là rất thuận lợi cho việc xây dựng một thuật toán tham lam giải quyết nó.

Nói chung, giải thuật tham lam thường có 5 thành phần

1. Một tập hợp các ứng cử viên (tập giá trị đề cử) để từ đó tạo ra lời giải.
2. Một hàm lựa chọn (select) để theo đó lựa chọn ứng cử viên tốt nhất để bổ sung vào lời giải.
3. Một hàm khả thi, dùng để quyết định nếu một ứng cử viên có thể được dùng để xây dựng lời giải.
4. Một hàm mục tiêu, ấn định giá trị của lời giải hoặc một lời giải chưa hoàn chỉnh.
5. Một hàm đánh giá, chỉ ra khi nào tìm được một lời giải hoàn chỉnh

Trong 5 thành phần trên có 2 yếu tố quyết định tới tính tham lam của thuật toán:

+ ***Tính lựa chọn tham lam***: Đây là thành phần then chốt đầu tiên, một giải pháp tối ưu toàn cục có thể đạt được bằng cách lựa chọn tối ưu cục bộ (tham lam).

Như vậy, khi có nhiều sự lựa chọn thì ta lựa chọn phương án nào tốt nhất ở hiện tại trong bài toán đang xét mà không cần quan tâm đến kết quả của các bài toán con của nó.

+ ***Cấu trúc con tối ưu***: Một bài toán có cấu trúc con tối ưu nếu giải pháp tối ưu cho bài toán này chứa trong nó các giải pháp tối ưu cho các bài toán con. Thuộc tính này là điểm quyết định để có thể giải bài toán bằng phương pháp quy hoạch động cũng như tham lam được hay không?

Thuật toán tham lam có được một giải pháp tối ưu cho một bài toán bằng cách thực hiện một chuỗi các lựa chọn. Đối với mỗi quyết định chỉ ra trong thuật toán sự lựa chọn này thường là tốt nhất tại thời điểm được chọn.

Chứng minh:

- Theo tính chất lựa chọn tham lam, tồn tại giải pháp tối ưu S chứa một lựa chọn tham lam a_1. Theo tính chất cấu trúc con tối ưu, X-{a_1} là giải pháp tối ưu của bài toán con không chứa a_1.

- Áp dụng cho bài toán con không chứa a_1, theo tính chất lựa chọn tham lam, X-{a_1} là giải pháp tối ưu chứa lựa chọn tham lam a_2. Theo tính chất cấu trúc con tối ưu, X-{a_1,a_2} là giải pháp tối ưu cho bài toán con không chứa a_1 và a_2.

- Tiếp tục như thế, cuối cùng ta có:

$$X-\{a_1,a_2,\ldots,a_n\} = \varnothing.$$

Vậy giải pháp tối ưu X của bài toán ban đầu là một dãy các sự lựu chọn tham lam thực hiện bởi thuật toán tham lam.

2.2.2.2 Sơ đồ chung để giải các bài toán bằng giải thuật tham lam

Tư tưởng của phương pháp tham lam

Ta xây dựng tập *S* dần từng bước, bắt đầu từ tập rỗng. Tại mỗi bước ta sẽ chọn một phần tử "tốt nhất" trong các phần tử còn lại của *A* để đưa vào *S*. Việc lựa chọn một phần tử như thế ở mỗi bước được hướng dẫn bởi hàm chọn. Phần tử được chọn sẽ bị loại khỏi tập *A*. Nếu khi thêm phần tử được chọn vào tập *S* mà *S* vẫn còn thỏa mãn các điều kiện của bài toán thì ta mở rộng *S* bằng cách thêm vào phần thử được chọn.

Mô hình:

Chọn *S* từ tập *A.*

Tính chất tham lam của thuật toán định hướng bởi hàm Chọn

- Khởi động S = ∅;

- Trong khi A ≠ ∅;

  - Chọn phần tử tốt nhất của A gán vào x : x = Chọn (A);
  - Cập nhật các đối tượng để chọn A : A = A - {x};
  - Nếu S ∪ {x} thỏa mãn yêu cầu bài toán thì:

  Cập nhật lời giải: S = S ∪ {x}

- Thủ tục thuật toán tham lam

  + Input A// *Tập các đối tượng cho trước*

  + Output X //*Lời giải, xây dựng phương án X từ A*

Greedy_Method(A,X);

Begin

X:= [];

While (A <> ∅) Do

Begin

X = Select(A);// Hàm chọn x tốt nhất trong A

A = A-{x}// Loại x ra khỏi A

If (X U {x} chấp nhận được) then X = X U {x};

Return X;

End;

End;

Trong thủ tục tổng quát trên, Select là hàm chọn, cho phép chọn từ tập A một phàn tử được xem là tốt nhất, nhiều hứa hẹn nhất là thành viên của tập nghiệm.

2.2.3 Một số bài toán áp dụng thuật toán tham lam

Bài toán xếp lịch cho các hoạt động

1. Mô tả bài toán

- Xét S = $\{a_1,a_2,..,a_n\}$ là tập các hoạt động muốn sử dụng tài nguyên (vd: hội trường)

- Mỗi hoạt động a_i sẽ có thời điểm bắt đầu là S_i và thời điểm kết thúc là f_i, với điều kiện $0 \le S_i < f_i < \infty$. Nếu hoạt động a_i được chọn, thì nó sẽ độc chiếm tài nguyên trong khoảng thời gian $[S_i,f_i)$. Hoạt động a_i và a_j được gọi là tương thích lẫn nhau nếu như khoảng thời gian $[S_i,f_i)$ và $[S_j,f_j)$ là không giao nhau

2. Yêu cầu: Mỗi thời điểm chỉ có 1 hoạt động sử dụng tài nguyên chung.

3. Mục tiêu: Chọn được một tập lớn nhất các hoạt động tương thích với nhau (khoảng thời gian thực hiện không giao nhau) => Tận dụng tối đa tài nguyên.

Ví dụ a_i và a_j *là tương thích nếu* $S_i \ge f_j$ *hoặc* $S_j \ge f_i$

4. Ý tưởng giải quyết bài toán

Xét một bài toán con khác rỗng S_{ij}, và nếu a_m là một hoạt động trong S_{ij} có thời điểm kết thúc sớm nhất: $f_m = min\{f\,k: ak \in S_{ij}\}$. Thì:

• Hoạt động a_m được sử dụng trong một tập con lớn nhất nào đó của các hoạt động tương thích lẫn nhau của S_{ij}.

• Bài toán con S_{im} là rỗng, do đó nếu chọn a_m thì chỉ còn duy nhất bài toán con khác rỗng S_{mj}.

Mục tiêu :

Giảm số các bài toán con và số cách chọn:

Chỉ duy nhất một bài toán con được sử dụng trong giải pháp tối ưu (một bài toán con khác rỗng)

Chỉ cần 1 chọn lựa cho bài toán con: chọn hoạt động nào có thời gian kết thúc sớm nhất trong S_{ij} (dễ dàng).

Có thể giải mỗi bài toán con theo phương pháp top down (thay vì bottom up trong lập trình động).

Vì khi chọn a_m chắc chắn lời giải S_{mj} sẽ được dùng trong lời giải tối ưu của $S_{ij} \rightarrow$ không cần giải S_{mj} trước khi giải S_{ij}

Tóm lại: Để giải bài toán con S_{ij}, đầu tiên chọn hoạt động a_m trong S_{ij} có thời gian kết thúc sớm nhất rồi mới tìm lời giải cho bài toán con S_{mj}

5. Chương trình minh họa cho thuật toán

- Input:

Mảng S và f biểu diễn thời điểm bắt đầu và kết thúc của các hoạt động

Giả sử N hoạt động đã xếp theo thứ tự tăng dần của thời điểm kết thúc.

$f_0 \leq f_1 \leq f_2 \leq \ldots \leq f_n \leq f_{n+1}$

- Output:

Trả về một tập lớn nhất của các hoạt động tương thích với nhau trong $S_{i, n+1}$

Chương trình:

RECURSIVE-ACTIVITY-SELECTOR (s ,f ,i, n)

M ←i+1

While m < n and $s_m < f_i$ {Tìm hoạt động đầu tiên trong $S_{i, n+1}$}

Do m←i+1

If m < n Then

Return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s ,f ,m ,n)

Else Return Ø

2.3 Thuật toán xấp xỉ (Heuristic)

2.3.1 Các khái niệm

Thuật toán xấp xỉ là các thuật toán tìm lời giải xấp xỉ cho các bài toán tối ưu hóa. Thuật toán xấp xỉ thường được sử dụng cho các bài toán NP - khó, hoặc các bài toán có thuật toán đa thức nhưng quá chậm cho dữ liệu lớn [1].

Người ta cho rằng ngày nay máy tính với tốc độ rất lớn, không cần quan tâm nhiều tới thuật toán nhanh nhưng với sự kiểm chứng sau đây: Bài toán xử lý với n đối tượng, có 3 thuật toán với 3 mức phức tạp khác nhau, sau 1 giờ xử lý sẽ chịu 3 hậu quả khác nhau.

Thuật toán	Độ phức tạp	Xử lý/1 giờ
A	O(n)	3,6 triệu đối tượng
B	$O(n\log_2 n)$	0,2 triệu đối tượng
C	$O(2^n)$	21 đối tượng

Trong khi đó nhiều bài toán có ý nghĩa thực tế lại thuộc lớp các bài toán NPC và rất quan trọng. Nếu một bài toán là NPC ta ắt không tìm một thuật toán thời gian đa thức. Vì vậy, có hai cách tiếp cận để có thể khắc phục tính NPC.

1. Nếu dữ liệu đầu vào thực tế là nhỏ thì một thuật toán có thời gian thực hiện hàm mũ có thể hoàn toàn thoả mãn.

2. Tìm các giải pháp gần tối ưu trong thời gian đa thức.

** Một thuật toán trả về các kết qủa gần tối ưu được gọi là một thuật toán xấp xỉ.*

Ta có các khái niệm sau đây:

- *Thuật toán tối ưu nhanh*: Là thuật toán tìm nghiệm tối ưu, nhưng nhanh (độ phức tạp thời gian là đa thức).
- *Thuật toán tối ưu chậm*: Là thuật toán tìm nghiệm tối ưu nhưng chậm (độ phức tạp thời gian là hàm mũ).
- *Thuật toán xấp xỉ nhanh:* Là các thuật toán tìm ra nghiệm gần đúng của bài toán với độ chính xác nào đó nhưng đủ nhanh. Thuật toán như vậy còn được gọi là "Thuật toán xấp xỉ đa thức".

Ví dụ 13: Bài toán phủ đỉnh tối ưu

- Input: Cho đồ thị vô hướng G = (V, E).
- Output: Tìm phủ đỉnh tối ưu (phủ đỉnh có kích thước cực tiểu).

Bài toán VC tìm ra phủ đỉnh có kích cỡ cực tiểu là NPC. Do đó khó có thể tìm ra 1 phủ đỉnh tối ưu nhưng không quá khó để tìm ra một phủ đỉnh gần tối ưu.

Sau đây là một thuật toán xấp xỉ cho kết qủa là một phủ đỉnh có kích cỡ không lớn hơn 2 lần kích cỡ một phủ đỉnh tối ưu trong thời gian đa thức:

```
Procedure   Approx _VertexCover;
  Begin
     C := ϕ;  {C - tập phủ gần tối ưu}
     E := Tập cạnh của đồ thị G;
   While  E ≠ ϕ  do
     Begin
          Chọn (u, v) là một cạnh tuỳ ý của E;
```

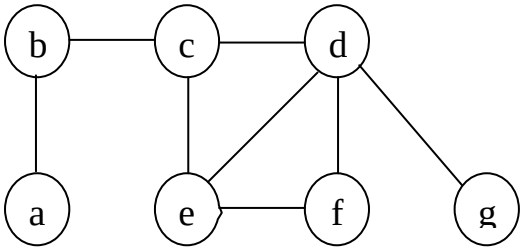
C := C ∪{u, v}; {Kết nạp hai đỉnh u, v vào phủ đỉnh C};

Gỡ bỏ khỏi E mọi cạnh liên thuộc với u hoặc v;

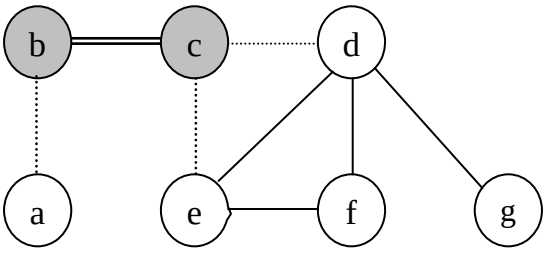
End;

Return(C);

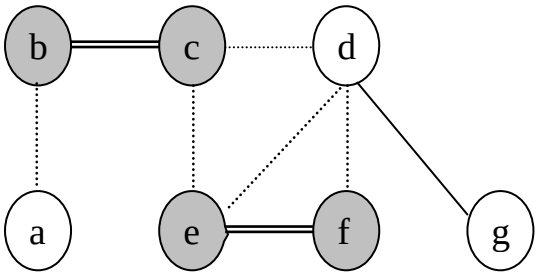
End;



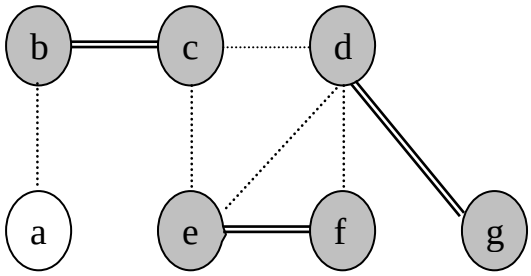
a) Đồ thị G



b) Chọn cạnh (b, c), gỡ bỏ (b,a),(c,e),(c,d)



c) Chọn tiếp (e, f), gỡ bỏ (e, d),(d, f)



d) Chọn cạnh (d,g), E = ϕ

Hình 2.2 Tìm phủ đỉnh tối ưu

Kết quả: Phủ đỉnh C ={b, c, d, e, f, g} gần tối ưu có kích thước 6. (phủ đỉnh tối ưu {b, e, d} có kích thước 3)

2.3.2 Thuật toán ε - xấp xỉ tuyệt đối

Cho P là bài toán cực đại hóa.

Gọi H là thủ tục Heuristic, thuật toán tìm một nghiệm nào đó cho P.

Kí hiệu OPT(I) là nghiệm tối ưu của bài toán P đối với thể hiện I.

Kí hiệu H(I) là nghiệm gần đúng của P do thuật toán H tìm ra.

Cho ε>0, thủ tục Heuristic H được gọi là thuật toán ε - xấp xỉ tuyệt đối khi và chỉ khi $|OPT(I) - H(I)| \le \varepsilon$ cho mọi thể hiện I của bài toán P (I: instance) với $\forall$ bộ dữ liệu.

Ví dụ 14: Bài toán lưu trữ tối đa số lượng chương trình

- Input: + N chương trình với dung lượng nhớ (độ dài) $d_1,d_2,\ldots, d_n$

+ Hai băng nhớ với dung lượng (độ dài) mỗi băng là L.

- Output: Hãy ghi các chương trình lên 2 băng nhớ với số lượng tối đa, mỗi chương trình chỉ được ghi trên một băng nhớ.

Bài toán này đã được chứng minh là NP-đầy đủ. Vì vậy việc tìm thuật toán đa thức cho nó là ít hi vọng.

Người ta đã dùng giải pháp tìm thuật toán xấp xỉ nhanh cho phép tìm được nghiệm gần đúng của nó nhưng chỉ mất thời gian đa thức.

Sau đây là thuật toán ε-xấp xỉ tuyệt đối: Cho kết quả nghiệm tối ưu và nghiệm gần đúng chỉ chênh nhau có 1.

Thuật toán ε - xấp xỉ tuyệt đối:

Procedure XXTD;

Begin

1. Sắp xếp các chương trình theo thứ tự tăng dần của $d_1,d_2,\ldots, d_n$;

2. i := 1;

For j:=1 to 2 do

Begin

dodai:=0;

While (dodai + $d_i \le L$) and ($i \le n$) do

Begin

<ghi chương trình i vào băng nhớ j>;

Dodai := dodai + d_i;

i := i+1;

End;

End;

End;

Chú ý: Mục đích muốn chỉ trên 2 băng nhớ có độ dài L mà lưu trữ được tối đa các chương trình. Theo suy nghĩ thông thường thì hãy ưu tiên các chương trình

có độ dài ngắn hơn, vì vậy đầu tiên là sắp xếp các chương trình theo thứ tự tăng dần các độ dài của chúng. Tiếp theo lần lượt xếp theo thứ tự này lên từng băng nhớ một. Do 2 băng nhớ có độ dài như nhau nên dùng băng nào trước cũng được. Theo thuật toán trên thì dùng băng 1 trước. Biến "dodai" ghi lại tổng độ dài băng nhớ dùng để lưu các chương trình.

Bài toán này còn được gọi là bài toán cắt n đoạn sắt từ 2 thanh sắt có cùng độ dài L sao cho số lượng đoạn sắt cắt ra là nhiều nhất (Chặt sắt - Cutting problem).

Chứng minh $|OPT(I) - H(I)| \le 1$

- Đặt k = H(I) là nghiệm của thuật toán Heurtstic $\Rightarrow$ k là số lượng chương trình được lưu trữ trên 2 băng nhớ theo cách sắp đặt của thuật toán xấp xỉ trên.

- Gọi p là số lượng chương trình được ghi trên 1 băng nhớ có độ dài bằng 2 băng nói trên

Như vậy $k \le OPT(I) \le p$ và $\sum_{i=1}^{p} d_i \le 2L$ (1)

Ta chứng minh $|OPT(I) - H(I)| \le 1 \Leftrightarrow OPT(I) - k \le 1 \Leftrightarrow OPT(I) \le k+1$ (2)

Theo (1) thì chỉ cần chứng minh $p \le k+1$ là đủ vì khi đó $OPT(I) \le p \le k+1$.

Chứng minh bằng phản chứng:

Giả sử $p > k+1 \Leftrightarrow p \ge k+2 \Leftrightarrow \sum_{i=1}^{k+2} d_i \le \sum_{i=1}^{p} d_i \le 2L$ (3)

Gọi m là số lượng chương trình được ghi trên một băng theo thuật toán xấp xỉ trên.

Khi đó : $\sum_{i=1}^{m} d_i + d_{m+1} > L \Rightarrow \sum_{i=1}^{m} d_i + d_{k+1} > L$ (4)

Tương tự trên băng nhớ 2 : $\sum_{i=m+1}^{k} d_i + d_{k+1} > L \Rightarrow \sum_{i=m+1}^{k} d_i + d_{k+2} > L$ (5)

Từ (4),(5) ta có : $\sum_{i=1}^{k+2} d_i > 2L \Rightarrow$ mâu thuẫn với (3) $\Rightarrow p \le k+1$ (đpcm).

Độ phức tạp thời gian của thuật toán:

Thời gian xử lý của thuật toán xấp xỉ trên là $O(n\log_2 n)$ (chủ yếu là phần sắp xếp các chương trình theo thứ tự của độ dài). Trong khi thuật toán chính xác phải

cần có thời gian hàm mũ, mà hiệu quả là 2 nghiệm chỉ chênh nhau có 1. Nếu những bài toán được giải tốt như ví dụ trên thì dùng giải pháp ε - xấp xỉ tuyệt đối. Nhưng không phải khi nào cũng suôn sẻ như vậy vì các thuật toán ε - xấp xỉ tuyệt đối tìm được không nhiều.

Hiện nay phần lớn các bài toán NP - đầy đủ thì việc tìm thuật toán ε - xấp xỉ tuyệt đối cho chúng cũng lại là NP - đầy đủ. Chẳng hạn như bài toán xếp balô (KNAPSACK), bài toán người bán hàng (Traverling Salesman Problem), bài toán MaxClique … Chính vì lẽ đó người ta dẫn ra khái niệm yếu hơn gọi là Thuật toán ε - xấp xỉ.

2.3.3 Thuật toán ε - xấp xỉ

Cho P là bài toán cực đại hóa.

Gọi H là thủ tục Heuristic, thuật toán xấp xỉ tìm một nghiệm nào đó cho P.

Kí hiệu OPT(I) là nghiệm tối ưu của bài toán P đối với thể hiện I (Instance).

H(I) là nghiệm gần đúng của P do H tìm ra.

Thủ tục Heuristic H được gọi là thuật toán ε - xấp xỉ khi và chỉ khi:

$$\frac{OPT(I)-H(I)}{OPT(I)} \le \varepsilon \text{ cho } \forall \text{ I.}$$

Ví dụ 15: Bài toán xếp balô giá trị nguyên (Integer - Valued Knapsack)

- Input: Một ba lô có thể tích B, n đồ vật có thể tích: $a_1, a_2,\ldots a_n$, giá trị tương ứng của các đồ vật là: $p_1, p_2,\ldots p_n$. Số lượng mỗi loại đồ vật là không hạn chế, x_i nguyên là số lượng loại đồ vật i.

- Ouput: Tìm nhóm đồ vật thoả mãn $\sum_{i=1}^{n} a_i x_i \le B$ và $\sum_{i=1}^{n} p_i x_i$ đạt max ?.

Tóm tắt: $\sum_{i=1}^{n} x_i p_i \rightarrow$ Max với $\sum_{i=1}^{n} x_i a_i \le B, x_i \in Z, 0 \le x_i \le b_i$. ở đây $b_i \le \left[\frac{B}{w_i}\right]$, điều này là hiển nhiên vì $\left[\frac{B}{w_i}\right]$ chính là số nguyên đồ vật có cùng thể tích a_i có thể nhét được vào ba lô.

Trường hợp $b_i = 1 \ \forall$ i thì vấn đề trên gọi là bài toán xếp balô 0-1, tức là chỉ được xếp nhiều nhất là 1 đồ vật vào balô (0-1 Knapsack)

Bài toán này đã được chứng minh là NP- đầy đủ. Vì vậy việc tìm thuật toán đa thức cho nó là không hi vọng. Người ta đã thử tìm thuật toán xấp xỉ tuyệt đối (nhanh) cho nó nhưng cũng không thành công vì việc tìm một thuật toán như vậy cũng lại là NP- khó.

Sau đây là thuật toán 1/2 - xấp xỉ cho bài toán xếp balô trị nguyên

Procedure XAPXI12;

Begin

1) sắp xếp tỉ số p_i/a_i , i = 1,2..,n theo thứ tự giảm dần;

2) T := 0;

for i := 1 to n do

begin

x_i := [(B-T)/a_i];

T := T+$x_i a_i$;

end;

End.

Chứng minh:

- Với mỗi thể hiện I ta có $OPT(I) \le p_1.\frac{B}{a_1}$ và $p_1.\left[\frac{B}{a_1}\right] \le H(I)$

Mặt khác $B - \left[\frac{B}{a_1}\right].a_1 < \frac{B}{2}$ (vì nếu ngược lại thì dẫn đến vô lý)

Khi đó $\frac{OPT(I)-H(I)}{OPT(I)} = 1 - \frac{H(I)}{OPT(I)} \le 1 - \frac{\left[\frac{B}{a_1}\right]}{\frac{B}{a_1}} = \frac{B - a_1\left[\frac{B}{a_1}\right]}{B} = \frac{B/2}{B} = \frac{1}{2}.$

Độ phức tạp thời gian của thuật toán

Thời gian xử lý thuật toán xấp xỉ trên chỉ là O(nlogn) (chủ yếu là phần sắp xếp tỉ số p_i/a_i), trong khi nếu dùng thuật toán chính xác phải cần thời gian hàm mũ.

Ngoài bài toán xếp balô (Knapsack) trên, hiện nay người ta đã tìm được thuật toán ε - xấp xỉ cho nhiều bài toán khác nhau, đặc biệt trong các vấn đề lập lịch. Tuy vậy với nhiều bài toán NP - đầy đủ thì việc tìm thuật toán ε - xấp xỉ cho chúng cũng lại là NP - đầy đủ. Chẳng hạn như bài toán Traveling Salesman Problem (TSP), bài toán quy hoạch nguyên (Integer Programming)..

2.3.4 Chứng minh tính đúng đắn của thuật toán

a. Ví dụ 16: - Input: Cho dãy đã sắp $a_1 \le a_2 \le ... \le a_n$. và một số TIM

- Output: Tìm vị trí của phần tử trong dãy a_k = TIM.

Giải thuật tìm kiếm nhị phân:

Left := 1; Right := N; Found := False;

While (not found) and (Left ≤ Right) do

Begin

Mid := (Left +Right) Div 2;

If Tim < a_{mid} Then Right := Mid

else If Tim >a_{mid} Then Left :=Mid

else found := True;

End.

b. Phương pháp thử

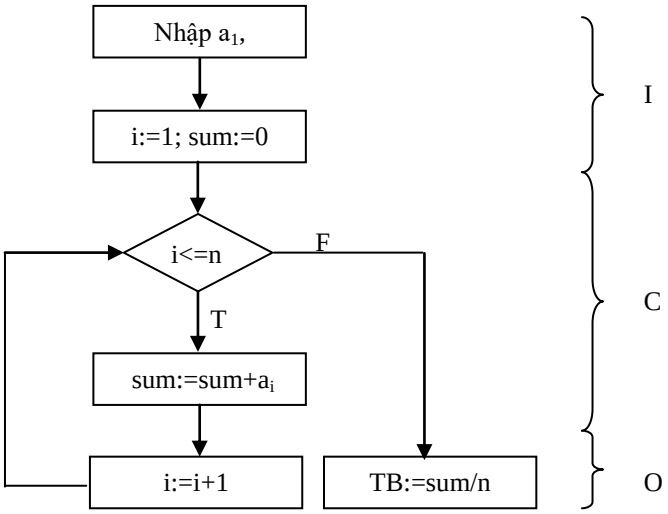
- Thử một số bộ dữ liệu: 9, 11, 15, 20, 23, 25, 30.
- Nếu Tim $\in \{a_2,..,a_{n-1}\}$ thì cho kết quả đúng.
- Nếu Tim = a_n dẫn đến lặp vô hạn

$\rightarrow$ Phải thay Right := Mid - 1;

Left := Mid + 1;

c. Kiểm chứng tính đúng đắn

Ví dụ 17: Tính trung bình n số.



Hình 2.3 Sơ đồ thuật toán tính trung bình n số

Chứng minh: Từ (I) qua (C) ra (O) bằng phương pháp quy nạp theo n

(1) Với n = 1 $\rightarrow$ sum := sum + a_1 = a_1 $\rightarrow$ đúng.

(2) Giả sử quá trình (C) đúng với i = n-1, phải chứng minh (C) đúng với i = n

$\Rightarrow \sum_{i=1}^{n-1} a_i + a_n = \sum_{i=1}^{n} a_i$.

Trở lại ví dụ 16 (tìm phần tử nhỏ nhất thứ k):

Procedure CHON(S,k);

1. If $\|S\| = 1$ *then return (phần tử duy nhất của S)*

else

Begin

2. Chọn phần tử tách thuộc S

Tách S thành 3 tập S_1, S_2, S_3

3. $S_1 := \{x \in S \mid x < tach\}$;

$S_2 := \{x \in S \mid x = tach\}$;

$S_3 := \{x \in S \mid x > tach\}$;

4. If $\|S_1\| > k$ *then CHON(*S_1*,k)*

else

if $\|S_1\| + \|S_2\| \geq k$ *then Return (phần tử là tach)*

else CHON($S_3, k - \|S_1\| - \|S_2\|$ *);*

End.

Chứng minh tính đúng đắn: Dùng phương pháp quy nạp theo n:

1. Với n=1 $\Rightarrow$ Hiển nhiên đúng.

2. Giả sử đúng với $\|S\| = n-1$, cần chứng minh đúng với $\|S\| = n$

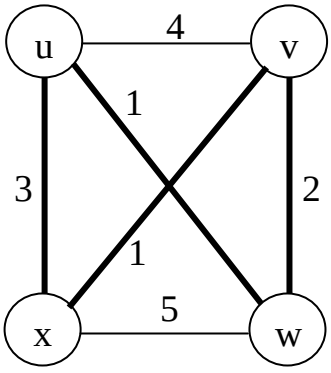
Theo thuật toán $\|S_1\|, \|S_3\| \leq n-1$ (bỏ ra phần tử tách) và ta phải sử dụng tiếp thuật toán này trên S_1 hoặc S_3 $\Rightarrow$ Vì đúng với n-1 nên thuật toán CHON trên S_1, S_3 là đúng $\Rightarrow$ điều phải chứng minh.

2.3.5 Bài toán TSP – Người bán hàng

- ***Phát biểu bài toán***

Có một người bán hàng cần đi giao hàng tại n thành phố. Người giao hàng xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu. Mỗi thành phố chỉ đến một lần, và khoảng cách từ một thành phố đến các thành phố khác đã được biết trước. Hãy tìm một chu trình (một đường đi khép kín thỏa mãn điều kiện trên) sao cho tổng mức hao phí là nhỏ nhất.

Ví dụ trong hình (2.4), hành trình người giao hàng có mức hao phí nhỏ nhất là <u, w, v, x, u>, với mức hao phí là 7.



Hình 2.4 Hành trình có mức hao phí nhỏ nhất

Hình (2.4): Một bộ dữ liệu vào của bài toán người bán hàng. Các cạnh tô đậm biểu diễn một hành trình có mức hao phí nhỏ nhất, với mức hao phí là 7.

Ngôn ngữ hình thức cho bài toán người bán hàng là:

TSP = {<G, c, k>: G= (V, E) là một đồ thị đầy đủ, C là một hàm từ V x V $\rightarrow$ Z, k$\in$Z và G có một hành trình người bán hàng với mức hao phí tối đa k}.

Định lý dưới đây chứng tỏ một thuật toán nhanh cho bài toán người bán hàng ắt không tồn tại.

- Đinh lý 1 "*Bài toán người bán hàng là NP đầy đủ*" [1].
- Chứng minh: Trước tiên ta chứng tỏ TSP thuộc về NP. Cho một bộ dữ liệu vào của bài toán, ta dùng dãy n đỉnh trong hành trình làm một giải pháp. Thuật toán sẽ xác minh, kiểm tra dãy này chứa mỗi đỉnh chính xác một lần, tổng cộng các mức hao phí cạnh, và kiểm tra xem tổng có phải tối đa là k hay không. Tiến trình này chắc chắn có thể thực hiện trong thời gian đa thức.

Để chứng minh TSP là NP - khó, ta chứng tỏ HAM-CYCLE $\leq_p$ TSP.

Cho G = <V, E> là một bộ dữ liệu vào của hàm HAM-CYCLE. Ta thiết kế một bộ dữ liệu vào của TSP như sau. Ta hình thành đồ thị đầy đủ G' = (V, E'), trong đó E' = {(i, j): i, j $\in$V}, và ta định nghĩa mức hao phí hàm c bằng

$$c(i,j) = \begin{cases} 0 & nÕu (i,j) \in E \\ 1 & nÕu (i,j) \notin E \end{cases}$$

Như vậy, bộ dữ liệu vào của TSP là (G', c, 0), được hình thành dễ dàng trong thời gian đa thức.

Ta chứng tỏ đồ thị G có một chu trình hamilton nếu và chỉ nếu đồ thị G' có một hành trình có mức hao phí tối đa là 0. Giả sử đồ thị G có một chu trình hamilton h. Mỗi cạnh trong h thuộc về E và như vậy có mức hao phí 0 trong G'. Như vậy, h là một hành trình trong G' có mức hao phí tối đa là 0. Bởi các mức hao phí của các cạnh trong E' là 0 và 1, mức hao phí của hành trình h' chính xác là 0. Do đó, h' chỉ chứa các cạnh trong E. Như vậy h là một chu trình hamilton trong đồ thị G.

Trong bài toán người bán hàng đã giới thiệu trên, ta có một đồ thị vô hướng đầy đủ G = (V, E) có một mức hao phí số nguyên không âm c(u, v) kết hợp với mỗi cạnh (u, v)$\in$E, và ta phải tìm một chu trình hamilton (một hành trình) của G với

mức hao phí cực tiểu. Để mở rộng hệ ký hiệu, ta cho c(A) thể hiện tổng mức hao phí của các cạnh trong tập hợp con $A \subseteq E$:

$$c(A)= \sum_{(u,v)\in A} c(u,v) .$$

Trong nhiều tình huống thực tiễn, đi trực tiếp từ một nơi u đến một nới w luôn là cách rẻ nhất; việc đi qua một điểm dừng trung gian v bất kỳ không thể ít tốn kém hơn. Đặt nó theo một cách khác, việc cắt giảm một điểm dừng trung gian không bao giờ gia tăng mức hao phí c thỏa bất đẳng thức tam giác nếu với tất cả các đỉnh u, v, w $\in$ V, $c(u, w) \leq c(u, v) + c(v, w)$.

Bất đẳng thức tam giác là một dạng tự nhiên, và trong nhiều ứng dụng nó tự động được thỏa. Ví dụ, nếu các đỉnh của đồ thị là các điểm trong mặt phẳng và mức hao phí du hành giữa hai đỉnh là khoảng cách euclid bình thường giữa chúng, thì bất đẳng thức tam giác được thỏa.

Việc hạn chế hao phí để thỏa bất đẳng thức tam giác sẽ là không làm thay đổi tính đầy đủ NP của bài toán người bán hàng. Như vậy, không chắc ta có thể tìm ra một thuật toán thời gian đa thức để giải bài toán này một cách chính xác. Do đó thay vì, ta tìm các thuật toán xấp xỉ tốt.

Sau đây, ta xét hai thuật toán xấp xỉ cho bài toán người bán hàng với bất đẳng thức tam giác có một cận tỷ số là 2:

Bài toán người bán hàng với APPROX-TSP-TOUR thỏa bất đẳng thức tam giác:

Thuật toán sau đây tính toán một hành trình gần tối ưu của một đồ thị vô hướng G, dùng thuật toán cây tỏa nhánh cực tiểu MST-PRIM (áp dụng tìm cây khung nhỏ nhất, hoạt động tương tự Dijkstra) để tìm các lộ trình ngắn nhất trong đồ thị. Nhưng ở đây, các cạnh trong tập hợp đã tìm được luôn hình thành một cây đơn lẻ.

Thuật toán xấp xỉ APPROX-TSP-TOUR:

APPROX-TSP-TOUR(G, c);

1 Lựa chọn đỉnh r ∈ V(G) là một đỉnh "gốc".

2 Tăng tưởng một cây tỏa nhánh cực tiểu T cho G từ gốc r (dùng MST-PRIM (G, c, r)) – Thuật toán tìm cây tỏa nhánh cực tiểu.

3 Cho L là danh sách các đỉnh được ghé thăm trong một tầng cây tiền cấp của T.

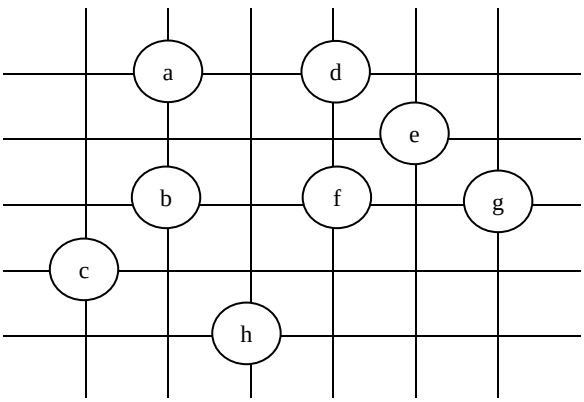
4 Return chu trình hamilton H ghé thăm các đỉnh theo thứ tự L.

- Cây khung nhỏ nhất: chứa tất cả các đỉnh của đồ thị có tổng trọng số các cạnh nhỏ nhất.
- Duyệt thứ tự trước: một nút được thăm trước khi thăm các nút con của nó
- Độ phức tạp thuật toán: $O(|V|^2)$

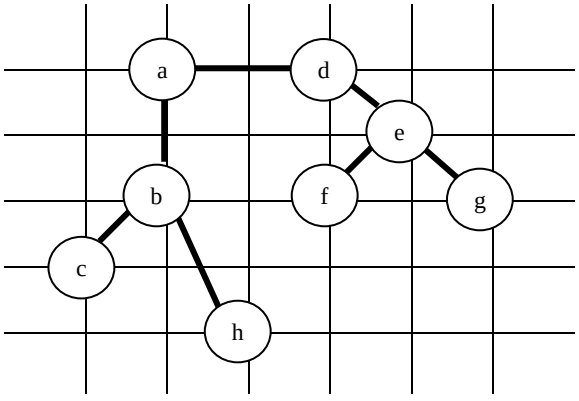
❖ Độ phức tạp thuật toán Prim

Lưu ý rằng một tầng cây tiền cấp ghé thăm đệ quy mọi đỉnh trong cây, liệt kê một đỉnh khi gặp nó lần đầu tiên, trước khi bất kỳ trong số các con của nó được ghé thăm.

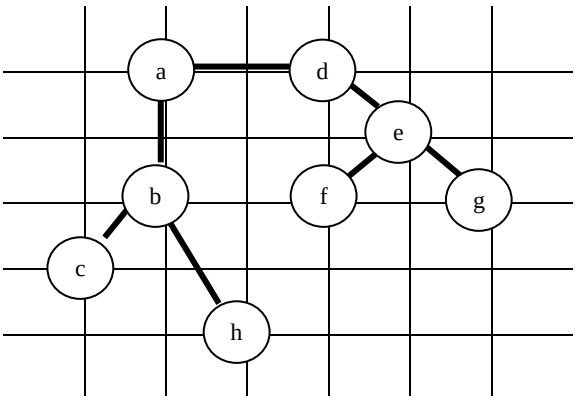
Hình (2.5) minh họa phép toán của APPROX-TSP-TOUR. Phần (a) của hình nêu tập hợp các đỉnh đã cho, và phần (b) nêu cây tỏa nhánh cực tiểu T được MST-PRIM tăng trưởng từ đỉnh gốc a. Phần (c) nêu cách các đỉnh được ghé thăm bởi một tầng tiền cấp của T, và phần (d) hiển thị hành trình tương ứng, là hành trình mà APPROX-TSP-TOUR trả về. Phần (e) hiển thị một hành trình tối ưu, ngắn hơn khoảng 23%.



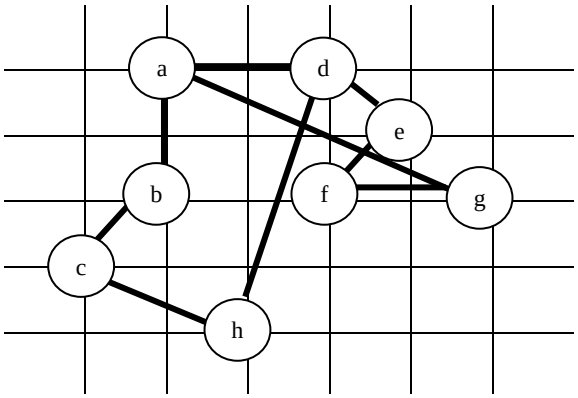
(a) Đồ thị G với khoảng cách giữa các đỉnh là trọng số của cạnh tương ứng.



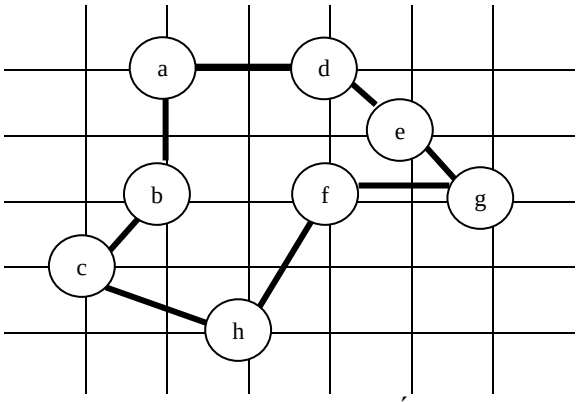
(b) Cây khung nhỏ nhất, gốc là a



(c) Duyệt cây T theo thứ tự trước: a, b, c, h, b, a, d, e, f, g, e, d, a



(d) Chu trình Hamilton, các đỉnh của L: a, b, c, h, d, e, f, g, a. Giải pháp xấp xỉ



(e) Giải pháp tối ưu

Hình 2.5 Minh họa thuật toán APPROX-TSP-TOUR

Hình 2.5: Phép toán của APPROX-TSP-TOUR. (a) tập hợp các điểm đã cho nằm trên các đỉnh của một khung kẻ ô số nguyên.

Ví dụ, f là một đơn vị về bên phải và hai đơn vị tiến lên từ h. Khoảng cách euclicd bình thường được dùng làm hàm hao phí giữa hai điểm. (b) Một cây tỏa nhánh cực tiểu T của các điểm này, như được tính toán bởi MST-PRIM. Đỉnh a là đỉnh gốc. Các đỉnh tình cờ được gán nhãn theo cách chúng được MST- PRIM bổ sung vào cây chính theo thứ tự abc. (c) Một tầng của T, bắt đầu tại a. Một tầng đầy đủ ghé thăm các đỉnh theo thứ tự a, b, c, b, h, b, a, d, e, f, e, g, e, d, a. Một tầng tiền cấp của T liệt kê một đỉnh ngay khi gặp nó lần đầu tiên, cho ra cách sắp xếp thứ tự a, b, c, h, d, e, f, g. (d) Một hành trình các đỉnh có được bằng cách ghé thăm các đỉnh theo thứ tự căn cứ vào tầng tiền cấp. Đây là hành trình H được APPROX-TSP-

TOUR trả về. Tổng mức hao phí của nó xấp xỉ là 19.074. (e) Một hành trình tối ưu H^* cho một tập hợp các đỉnh đã cho. Tổng mức hao phí của nó xấp xỉ là 14.715.

Thời gian thực hiện của APPROX-TSP-TOUR là $O(E)=O(V^2)$, bởi đầu vào là một đồ thị đầy đủ. Giờ đây ta chứng tỏ nếu hàm hao phí dành cho một bộ dữ liệu vào của bài toán người bán hàng thỏa bất đẳng thức tam giác, thì APPROX-TSP-TOUR trả về một hành trình có mức hao phí không nhiều hơn gấp hai lần mức hao phí của một hành trình tối ưu.

- Định lý 2

"APPROX-TSP-TOUR là một thuật toán xấp xỉ có một cận tỷ số là 2 cho bài toán người bán hàng với bất đẳng thức tam giác" [1].

***** Chứng minh: Cho H^* thể hiện một hành trình tối ưu cho một tập hợp các đỉnh đã cho. Một phát biểu tương đương của một định lý đó là $c(H) \leq 2c(H^*)$, trong đó H là hành trình do APPROX-TSP-TOUR trả về. Bởi ta được một cây tỏa nhánh bằng cách xóa một cạnh bất kỳ ra khỏi một hành trình, nếu T là một cây tỏa nhánh cực tiểu cho tập hợp các đỉnh đã cho, thì

$$c(T) \leq c(H^*) \qquad (2.3.5a)$$

Một *tầng đầy đủ* (full walk) của T liệt kê các đỉnh khi lần đầu tiên chúng được ghé thăm và mỗi khi chúng được trả về sau một lần ghé thăm một cây con. Ta hãy gọi tầng này là W. Tầng đầy đủ của ví dụ cho thứ tự

a, b, c, b, h, b, a, d, e, f, e, g, e, d, a .

Bởi tầng đầy đủ băng ngang mọi cạnh của T chính xác hai lần, nên ta có:

$$c(W) = 2c(T). \qquad (2.3.5b)$$

Các phương trình (2.3.5a) và (2.3.5b) hàm ý rằng $c(W) \leq 2c(H^*)$, và do đó mức hao phí của W nằm trong một thừa số của 2c so với mức hao phí của một hành trình tối ưu.

Nhưng thông thường W không phải là một hành trình, bởi nó ghé thăm vài đỉnh nhiều lần. Tuy nhiên, theo bất đẳng thức tam giác, ta có thể xóa một lần ghé thăm đến một đỉnh bất kỳ từ W và mức hao phí không tăng. (Nếu một đỉnh v được xóa ra khỏi W giữa các lần ghé thăm u và w, cách sắp xếp kết quả chỉ định đi trực

tiếp từ u to w). Bằng cách liên tục áp dụng phép toán này, ta có thể gỡ bỏ ra khỏi W tất cả, ngoại trừ lần ghé thăm đầu tiên đến mỗi đỉnh. Trong ví dụ này, điều này để lại cách sắp xếp thứ tự: a, b, c, h, d, e, f, g.

Cách sắp xếp này giống như cách sắp xếp có được bằng một tầng tiền cấp của cây T. Cho H là một chu trình tương ứng với tầng tiền cấp này. Nó là một chu trình hamilton, bởi mọi đỉnh được ghé thăm chính xác một lần, và thực tế nó là chu trình đã tính toán bởi APPROX-TSP-TOUR. Bởi H có được bằng cách xóa các đỉnh ra khỏi tầng đầy đủ W, ta có

$$c(H) \leq c(W). \qquad (2.3.5c)$$

Tổ hợp các bất đẳng thức (2.3.5b) và (2.3.5c) sẽ hoàn tất phần chứng minh. Bất kể cận tỷ số tế nhị mà định lý trên đã cung cấp, APPROX-TSP-TOUR thường không phải là lựa chọn thực tế tốt nhất cho bài toán này. Có các thuật toán xấp xỉ khác thường thực hiện ít hơn nhiều trong thực tế.

* ***Chú ý***: Nếu bỏ giả thiết hàm hao phí c thỏa bất đẳng thức tam giác, ta không thể tìm thấy các hành trình xấp xỉ tốt trong thời gian đa thức, trừ khi P = NP.

- Xét: "Nếu P ≠ NP và P ≥ 1, Không có thuật toán xấp xỉ thời gian đa thức có cận tỷ số P cho bài toán Người bán hàng nếu bỏ qua giả thiết hàm hao phí c thỏa bất đẳng thức tam giác".

***** Chứng minh: Vì sự mâu thuẫn, giả sử rằng với một số p ≥ 1, ta có một thuật toán xấp xỉ thời gian đa thức A với cận tỷ số p. Không mất tính tổng quát, ta mặc nhận p là một số nguyên, bằng cách làm tròn lên nếu cần. Như vậy, ta sẽ nêu cách sử dụng A để giải các bộ dữ liệu vào của bài toán Chu trình hamilton (theo định lý "*Bài toán người bán hàng là NP đầy đủ*") trong thời gian đa thức. Bởi bài toán Chu trình hamilton là NP đầy đủ (theo định lý "*Bài toán Chu trình hamilton là NP đầy đủ*"), nên việc giải nó trong thời gian đa thức hàm ý rằng P = NP, theo định lý "*Nếu một bài toán NP đầy đủ bất kỳ là giải được theo thời gian đa thức, thì P = NP. Nếu một bài toán trong NP bất kỳ không giải được theo lời giải đa thức, thì tất cả các bài toán NP đầy đủ đều không giải được theo lời giải đa thức*".

Bài toán Người bán hàng với HEURISTIC-TSP-TOUR (heuristic điểm sát nhất) thỏa bất đẳng thức tam giác:

Giả sử bài toán Người bán hàng có hao phí c thỏa mãn bất đẳng thức tam giác.

Xét thuật toán "tham xấp xỉ" *heuristic điểm sát nhất* có tên là HEURISTIC-TSP-TOUR để xây dựng một hành trình người bán hàng xấp xỉ. Bắt đầu bằng một chu trình tầm thường bao gồm một đỉnh đơn lẻ được chọn tùy ý. Tại mỗi bước, định danh đỉnh u không nằm trên chu trình nhưng có khoảng cách đến một đỉnh bất kỳ trên chu trình là cực tiểu. Giả sử rằng đỉnh trên chu trình là u gần nhất chính là đỉnh v. Đến khi tất cả các đỉnh đều nằm trên chu trình. Khi đó hàm heuristic này trả về một hành trình có tổng mức hao phí không nhiều hơn gấp 2 lần so với mức hao phí của một hành trình tối ưu (cận tỷ số 2). Bản chất của thuật toán là phương pháp "tham lam suy nghiệm".

HEURISTIC-TSP-TOUR heuristic điểm sát nhất thực hiện các bước như sau:

HEURISTIC-TSP-TOUR(G)

1. Lựa một đỉnh r ∈ V(G) là một đỉnh "gốc"

2. Tăng trưởng trên G từ gốc r, theo nguyên tắc điểm sát nhất (cạnh có hao phí thấp nhất được bổ sung)

3. Cho L là danh sách các đỉnh được ghé thăm lần lượt trên G.

4. Return chu trình Hamilton H ghé thăm các đỉnh theo thứ tự trong L.

Thời gian thực hiện của HEURISTIC-TSP-TOUR là $O(V^2)$, bởi đầu vào là một đồ thị đầy đủ. Cũng như thuật toán APPROX-TSP-TOUR, nếu hàm hao phí dành cho một bộ dữ liệu vào của bài toán người bán hàng thỏa bất đẳng thức tam giác, thì HEURISTIC-TSP-TOUR trả về một hành trình có mức hao phí không nhiều hơn gấp hai lần mức hao phí của một hành trình tối ưu.

- Nhận xét: "HEURISTIC-TSP-TOUR là một thuật toán xấp xỉ có một cận tỷ số là 2 cho bài toán người bán hàng với bất đẳng thức tam giác";

- Thủ tục thể hiện thuật toán HEURISTIC-TSP-TOUR:

Giả sử bài toán TSP có n đỉnh.

```
Program HEURISTIC-TSP-TOUR(G);
Var  i, j, n, vt, min: Integer;
     A : array [1..n,1..n] of integer; {lưu độ dài đường đi của đồ thị G}
     C : array [1..n,1..n] of boolean; {đánh dấu các đoạn đường đã đi qua}
     U : set of integer; {Tập chứa các đỉnh của đồ thị}
     L : danh sách chứa các cạnh theo thứ tự trong hành trình;
     Dem: Integer; {theo dõi hành trình, nếu đi được n đỉnh rồi thì trở về đỉnh xuất phát}.
Begin
        u := [];        L ← Ø;
        for i := 1 to n do u := u + [i];  {u chứa tập các đỉnh của đồ thị G}
        i := 1; dem:=0;
        While dem < n do
      begin
          u := u – [i]; {đang xét đỉnh i}   min :=  maxlongint;
          for  j := 1 to n do    if  j in  u  then
          if (a[i,j] < min) and (i<>j) then  {lấy cạnh ngắn nhất nối với đỉnh chưa xét}
        begin
           vt := j; {xác định vị trí của đỉnh vt được chọn}
           min := a[i,j];
       end;
          L ← 1 + a(i, vt); {bổ sung cạnh (i, vt) vào danh sách các đỉnh đã chọn}
          c[i, vt] := true {đánh dấu những cạnh đã đi qua}
          c[vt, i] := true;
          i := vt;  dem:=dem+1;
    end;
End;
```

Thời gian thực hiện của HEURISTIC-TSP-TOUR là $O(E) = O(V^2)$, trong thời gian đa thức. Thuật toán HEURISTIC-TSP-TOUR trả về một hành trình có mức hao phí không nhiều hơn gấp hai lần mức hao phí của một hành trình tối ưu.

2.4 Một số thuật toán trên đồ thị

2.4.1 Khái niệm cơ bản

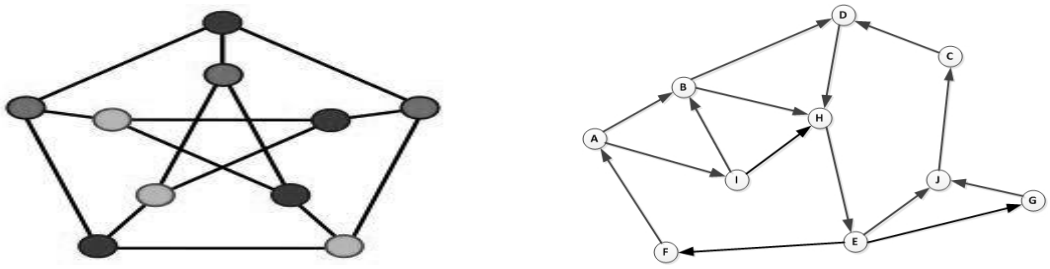
Đồ thị là mô hình biểu diễn một tập đối tượng và mối quan hệ hai ngôi giữa cặp đối tượng đó [2], [8], [9]11.

G = (V, E)

Có thể định nghĩa đồ thị là một cặp (V, E): G = (V, E). Trong đó V là tập các đỉnh biểu diễn các đối tượng và E là tập các cạnh biểu diễn mối quan hệ giữa các đối tượng.

Ví dụ: sơ đồ giao thông gữa các thành phố là một đồ thị, với tập các đỉnh V là tập các thành phố, tập cạnh E là tập các con đường trực tiếp nối hai thành phố thuộc V.

Một số hình ảnh của đồ thị:



Đồ thị vô hướng Đồ thị có hướng

Hình 2.6 Một số hình ảnh của đồ thị có hướng – vô hướng

2.4.2 Phân loại đồ thị

Có thể phân loại đồ thị theo đặc tính và số lượng của tập các cạnh E:

Cho đồ thị G = (V, E). Định nghĩa một cách hình thức

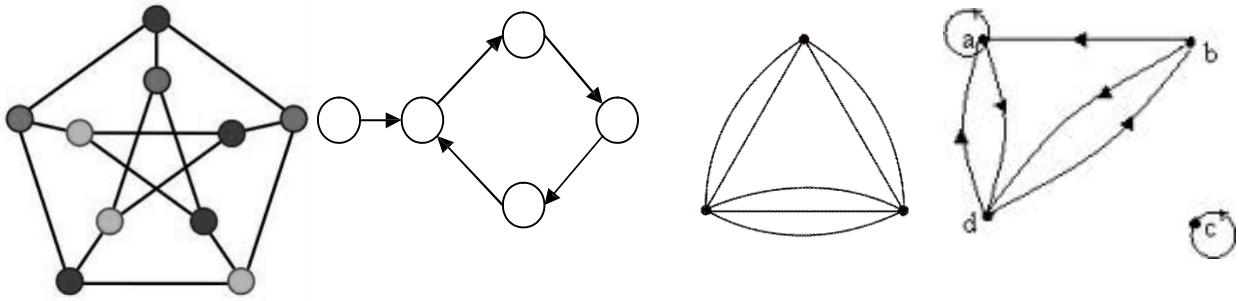
- G được gọi là **đơn đồ thị** nếu giữa hai đỉnh u, v của V có nhiều nhất là 1 cạnh trong E nối từ u tới v.

- G được gọi là **đa đồ thị** nếu giữa hai đỉnh u, v của V có thể có nhiều hơn 1 cạnh trong E nối từ u tới v.

- G được gọi là đồ thị **vô hướng** (undirected graph) nếu các cạnh trong E là không định hướng, tức là cạnh nối hai đỉnh u, v bất kỳ cũng là cạnh nối hai đỉnh v, u. Hay nói cách khác, tập E gồm các cặp (u, v) không tính thứ tự (u, v) ≡(v , u).

- G được gọi là đồ thị **có hướng** (directed graph) nếu các cạnh trong E là có định hướng, có thể có cạnh nối từ đỉnh u tới đỉnh v nhưng chưa chắc đã có cạnh nối từ đỉnh v tới đỉnh u. Hay nói cách khác, tập E gồm các cặp (u, v) có tính thứ tự: (u, v) ≠ (v, u). Trong đồ thị có hướng, các cạnh được gọi là các **cung**. Đồ thị vô hướng cũng có thể coi là đồ thị có hướng nếu như ta coi cạnh nối hai đỉnh u, v bất kỳ tương đương với hai cung (u, v) và (v, u).

Ví dụ:



Hình 2.7 Đơn đồ thị, đa đồ thị

2.4.3 Một số khái niệm

Các phần tử thuộc tập V gọi là **đỉnh** của đồ thị.

Cho hai **đỉnh** u, v thuộc V, nếu cạnh e = (u, v) ∈ E là cặp thứ tự thì e được gọi là một **cung** của đồ thị, hoặc nếu e là cặp không sắp xếp thứ tự thì e được gọi là một cạnh của đồ thị.

Khi e = (u ,v) là một cung thì u là đỉnh đầu của cung, v là đỉnh cuối của e

Khi e = (u ,v) là cạnh thị u và v gọi là **hai đỉnh kề** của cạnh e hoặc **hai đỉnh liên** thuộc e

Hai đỉnh u và v (u ≠ v) của đồ thị được gọi là hai đỉnh kề nhau nếu chúng là hai đầu của một cạnh hay một cung.

Hai cạnh a và b gọi là **hai cạnh kề nhau** (hoặc hai cung kề nhau) nếu chúng có chung một đỉnh.

Với một đỉnh v trong đồ thị, ta định nghĩa **bậc** (degree) của v, ký hiệu deg(v) là số cạnh liên thuộc với v. Dễ thấy rằng trên đơn đồ thị thì số cạnh liên thuộc với v cũng là số đỉnh kề với v.

Định lý 3: Giả sử G = (V, E) là đồ thị vô hướng với m cạnh, khi đó tổng tất cả các bậc đỉnh trong V sẽ bằng 2m.

Chứng minh: Khi lấy tổng tất cả các bậc đỉnh tức là mỗi cạnh e = (u, v) bất kỳ sẽ được tính một lần trong deg(u) và một lần trong deg(v). Từ đó suy ra kết quả.

Hệ quả: Trong đồ thị vô hướng, số đỉnh bậc lẻ là số chẵn

Đối với đồ thị có hướng G = (V, E). Xét một cung $e \in E$, nếu e = (u, v) thì ta nói **u nối tới v** và **v nối tới u**, cung e là đi **ra khỏi đỉnh u và đi vào đỉnh v**. Đỉnh u khi đó được gọi là đỉnh đầu, đỉnh v được gọi là đỉnh cuối của cung e.

Với mỗi đỉnh v trong đồ thị có hướng, ta định nghĩa: **Bán bậc ra** của v ký hiệu $deg^+(v)$ là số cung đi ra khỏi nó; **bán bậc vào** ký hiệu $deg^-(v)$ là số cung đi vào đỉnh đó.

Khuyên là cạnh (hoặc cung) có 2 đỉnh đầu trùng nhau.

Đỉnh treo là đỉnh thuộc duy nhất một cạnh hoặc cung.

Đỉnh cô lập là đỉnh không thuộc cạnh hoặc cung nào.

2.4.4 Biểu diễn đồ thị trên máy tính

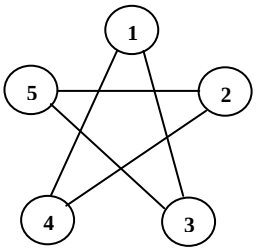
Ma trận kề (Adjacency Matrix)

Giả sử G = (V, E) là một đơn đồ thị có số đỉnh (ký hiệu $|V|$) là n, không mất tính tổng quát có thể coi các đỉnh được đánh số 1, 2, 3, …, n. Khi đó ta có thể biểu diễn đồ thị bằng một ma trận vuông A=[a[i, j]] cấp n. trong đó

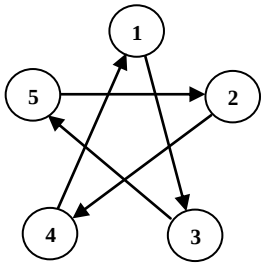
- a[i, j] = 1 nếu $(i, j) \in E$
- a[i, j] = 0 nếu $(i, j) \notin E$

Với $\forall i$, giá trị của a[i, j] có thể đặt tùy ý theo mục đích, thông thường nên đặt bằng 0; Đối với đa đồ thị thì việc biểu diễn cung tương tự trên, chỉ có điều nếu như (i, j) là cạnh thì không phải ghi số 1 vào vị trí a[i, j] mà ghi số cạnh nối giữa đỉnh i và đỉnh j.

Ví dụ biểu diễn đồ thị trên máy tính:



$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$



$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Hình 2.8 Biểu diễn đồ thị trên máy tính

2.4.5 Một số thuật toán chọn lọc trên mô hình đồ thị [8], [9]

2.4.5.1 Thuật toán tìm cây khung nhỏ nhất

- Phát biểu bài toán:

Định nghĩa: Cho đồ thị G vô hướng, liên thông và có trọng số không âm. Cây khung nhỏ nhất của đồ thị G là cây khung có tổng trọng số trên các cạnh của nó nhỏ nhất (gọi là trọng số của cây khung nhỏ nhất).

a- Thuật toán Kruskal

Ý tưởng: Nạp dần các cạnh nhỏ nhất vào cây khung nếu cạnh ấy không tạo thành chu trình với các cạnh đã nạp.

Thuật toán:

Bước 1: Sắp xếp các cạnh tăng dần

Bước 2: Lần lượt kết nạp các cạnh có trọng số nhỏ nhất trong các cạnh còn lại vào cây nếu sau khi kết nạp cạnh này không tạo thành chu trình trong cây. Quá trình này dừng khi kết nạp được n-1 cạnh vào cây.

```
Procedure  Kruskal(G: đồ thị n đỉnh, liên thông có trọng số);
 Begin
      T := ∅;
      for i:= 1 to n – 1 do
    begin
          e := một cạnh bất kỳ của G với trọng số nhỏ nhất và khi ghép
               vào T không tạo ra chu trình trong T;
          T := T ∪{e};
   end;
 End;  {T là cây khung nhỏ nhất}
```

b- Thuật toán Prim

Ý tưởng: Nạp dần tập các đỉnh vào cây khung. Mỗi lần chọn một đỉnh chưa nạp là đỉnh kề và gần các đỉnh đã nạp nhất.

Thuật toán

Bước 1: Nạp một đỉnh đầu tiên vào cây khung (thường là đỉnh 1)

Bước 2: Lần lượt nạp n-1 đỉnh còn lại (tương ứng với n-1 cạnh) vào cây khung bằng cách: mỗi lần chọn một cạnh có trọng số nhỏ nhất mà một đầu của cạnh đã thuộc cây, đầu kia chưa thuộc cây (nghĩa là chọn một đỉnh gần các đỉnh đã nạp nhất).

Procedure Prim;

Begin

T := min (d[u, v]); {cạnh có trọng số nhỏ nhất}

for i:=1 to n – 2 do

begin

e := cạnh có trọng số tối thiểu liên thuộc với một đỉnh trong T và khi ghép nó vào T không tạo ra chu trình trong T.

T := T ∪{e};

end;

End; {T là cây khung nhỏ nhất trong G}

2.4.5.2 Chu trình Euler

a. Định nghĩa

Đường đi qua tất cả các cạnh/cung, mỗi cạnh/cung qua đúng một lần gọi là đường đi Euler. Đường đi Euler có điểm đầu và điểm cuối trùng nhau gọi là chu trình Euler.

Đồ thị có đường đi Euler gọi là đồ thị nửa Euler.

Đồ thị có chu trình Euler gọi là đồ thị Euler.

b. Định lý 4

Đồ thị vô hướng, liên thông là Euler khi và chỉ khi mọi đỉnh đều có bậc chẵn.

Đồ thị vô hướng, liên thông là nửa Euler khi và chỉ khi có đúng hai đỉnh bậc lẻ.

Đồ thị có hướng, liên thông yếu và mọi đỉnh có bán bậc ra bằng bán bậc vào thì có chu trình Euler.

Đồ thị có hướng, liên thông yếu nếu tồn tại hai đỉnh U và V thỏa mãn: bán bậc vào của U ít hơn bán bậc ra của U một đơn vị, bán bậc vào của V nhiều hơn bán bậc ra của V một đơn vị, mọi đỉnh khác nhau có bán bậc ra bằng bán bậc vào, thì có đường đi Euler từ U tới V.

c. Thuật toán Fleury

Ý tưởng: Lần lượt chọn các cạnh liên tiếp nhau, chỉ chọn cạnh là cầu khi không còn cách chọn cạnh không là cầu. Nếu cạnh nào được chọn thì ghi nhận vào kết quả rồi xóa cạnh đó trên đồ thị.

❖ Thuật toán tìm chu trình Euler (đồ thị vô hướng, liên thông)

Bước 1: Tính số đỉnh bậc lẻ. Nếu số đỉnh bậc lẻ >0 thì kết luận không có chu trình, sau đó thoát. Nếu số đỉnh bậc lẻ bằng 0 thì kết luận có chu trình.

Bước 2: Nạp đỉnh 1 vào Stack, thực hiện vòng lặp trong khi Stack chưa rỗng:

- Lấy đỉnh X ở Stack.
- Nếu còn đỉnh J kề với X thì cho J vào Stack, xóa cạnh (X, J), nếu không còn đỉnh J kề với X thì ghi nhận X vào mảng kết quả.

Bước 3: Hiện mảng kết quả.

❖ Thuật toán tìm đường đi Euler (đồ thị vô hướng, liên thông)

Bước 1: Tính số đỉnh bậc lẻ. Nếu số đỉnh bậc lẻ lớn hơn 2 thì kết luận không có đường đi, sau đó thoát. Nếu số đỉnh bậc lẻ bằng 2 thì kết luận có đường đi Euler và sang bước 2.

Bước 2: Tạo thêm đỉnh mới là N+1, nối thêm cạnh từ hai đỉnh bậc lẻ tới đỉnh mới này, rồi tìm chu trình Euler xuất phát từ đỉnh mới này.

Bước 3: Hiện đường đi Euler, là đường đi sinh ra từ chu trình vừa tìm được bằng cách loại đi cạnh đầu và cạnh cuối cùng của chu trình này.

2.4.5.3 Chu trình Hamilton

a- Phát biểu bài toán

Khái niệm chu trình Hamilton: Cho đồ thị G = (V, E) có n đỉnh

Chu trình (x_1, x_2, ..., x_n) được gọi là chu trình Hamilton nếu $x_i \neq x_j$ với $1 \leq i < j \leq n$

Đường đi (x_1, x_2, ..., x_n) được gọi là đường đi Hamilton nếu $x_i \neq x_j$ với $1 \leq i < j \leq n$

Có thể phát biểu một cách hình thức: *Chu trình Hamilton là chu trình xuất phát từ 1 đỉnh, đi thăm tất cả những đỉnh còn lại mỗi đỉnh đúng 1 lần, cuối cùng quay trở lại đỉnh xuất phát. Đường đi Hamilton là đường đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng 1 lần.*

Bài toán tìm chu trình Hamilton: Cho đồ thị G = (V, E) có n đỉnh, m cạnh.

Yêu cầu: Hãy chỉ ra một chu trình Hamilton của đồ thị đã cho.

Để tìm chu trình Hamilton, ta quan tâm đến các định lý sau:

Định lý 5: Đồ thị vô hướng G, trong đó tồn tại k đỉnh sao cho nếu xóa đi k đỉnh này cùng với những cạnh liên thuộc của chúng thì đồ thị nhận được sẽ có nhiều hơn k thành phần liên thông. Thì khẳng định là G không có chu trình Hamilton. Mệnh đề phản đảo của định lý này cho ta điều kiện cần để một đồ thị có chu trình Hamilton

Định lý 6: (Định lý Dirac (1952)): Đồ thị vô hướng G có N đỉnh (N≥3). Khi đó nếu mọi đỉnh v của G đều có deg(v) ≥ N/2 thì G có chu trình Hamilton. Đây là một điều kiện đủ để một đồ thị có chu trình Hamilton.

Định lý 7: Đồ thị có hướng G liên thông mạnh và có n đỉnh. Nếu $deg^+(v) \geq \frac{n}{2}$ và $deg^-(v) \geq \frac{n}{2}$ với mọi đỉnh v thì G có chu trình Hamilton.

b- Thuật toán tìm chu trình Hamilton

Dưới đây ta sẽ cài đặt một chương trình liệt kê tất cả các chu trình Hamilton xuất phát từ đỉnh 1, các chu trình Hamilton khác có thể có được bằng cách hoán vị vòng quanh. Lưu ý rằng cho tới nay, người ta vẫn **chưa tìm ra** một

phương pháp nào thực sự hiệu quả hơn phương pháp quay lui để tìm dù chỉ một chu trình Hamilton cũng như đường đi Hamilton trong trường hợp đồ thị tổng quát.

Bước 1: Khởi tạo: free[u] = true với u thuộc G

Bước 2: Visit(i): Thăm đỉnh ở bước thứ i

2.1 u = đỉnh vừa thăm xong

Xét các đỉnh v kề u và free[v] = true

2.2 Ghi nhận đỉnh ở bước thứ i

2.3 Nếu (i = n) và (u kề với 1) đến Bước 3

2.4 Đánh dấu thăm v và Vitsit(i+1);

Bước 3: Truy vết tìm đường đi (nếu có).

2.4.5.4 Thuật toán tìm đường đi ngắn nhất

- Phát biểu bài toán

Cho đồ thị G = <X, E> có trọng số không âm. S là đỉnh xuất phát, T là đỉnh kết thúc. Hãy xác định đường đi từ S đến T sao cho tổng độ dài đường đi là nhỏ nhất.

Đây là một bài toán rất quan trọng trong công nghệ thông tin.

Dưới đây, luận văn giới thiệu hai thuật toán giải bài toán này là thuật toán Ford – Bellman và thuật toán Dijkstra.

a- Thuật toán Ford - Bellman

Thuật toán Ford-Bellman có thể phát biểu:

Với đỉnh xuất phát S. Gọi d(v) là khoảng cách từ S tới v.

Ban đầu d(S) được khởi gán bằng 0 còn các d(v) với v ≠ S được khởi gán bằng +∞.

Sau đó ta tối ưu hoá dần các d(v) như sau: Xét mọi cặp đỉnh u, v của đồ thị, nếu có một cặp đỉnh u, v mà d(v) > d(u) + c(u, v) thì ta đặt lại d(v) := d(u) + c(u, v). Tức là nếu độ dài đường đi từ S tới v lại **lớn hơn** tổng độ dài đường đi từ S tới u cộng với chi phí đi từ u tới v thì ta sẽ huỷ bỏ đường đi từ S tới v đang có và coi đường đi từ S tới v chính là đường đi từ S tới u sau đó đi tiếp từ u tới v. Chú ý rằng ta đặt c[u, v] = +∞ nếu (u, v) không là cung. Thuật toán sẽ kết thúc khi không

thể tối ưu thêm bất kỳ một nhãn d[v] nào nữa.

Tính dừng của thuật toán:

Tại bước lặp 0: Bước khởi tạo d(S) = 0; d(v) := +∞ với v ≠ S: thì dãy d(v) chính là độ dài đường đi ngắn nhất từ S tới v đi qua không quá 0 cạnh

Giả sử tại bước lặp thứ i, d(v) bằng độ dài đường đi ngắn nhất từ S tới v qua không quá i cạnh, thì do tính chất: đường đi từ S tới v qua không quá i + 1 cạnh sẽ phải thành lập bằng cách: lấy một đường đi từ S tới một đỉnh u nào đó qua không quá i cạnh, rồi đi tiếp tới v bằng cung (u, v). Nên độ dài đường đi ngắn nhất từ S tới v qua không quá i + 1 cạnh sẽ được tính bằng giá trị nhỏ nhất trong các giá trị: (Nguyên lý tối ưu Bellman)

Độ dài đường đi ngắn nhất từ S tới v qua không quá i cạnh

Độ dài đường đi ngắn nhất từ S tới u qua không quá i cạnh cộng với trọng số cạnh (u, v) (∀u)

Nên sau bước lặp tối ưu các d(v) bằng công thức $d(v)_{\text{bước } i+1} = \min(d(v)_{\text{bước } i}, d(u)_{\text{bước } i} + c(u, v))$ thì các d(v) sẽ bằng độ dài đường đi ngắn nhất từ S tới v qua không quá i + 1 cạnh.

Sau bước lặp tối ưu thứ n - 1, ta có d(v) = độ dài đường đi ngắn nhất từ S tới v qua không quá n - 1 cạnh. Vì đồ thị không có chu trình âm nên sẽ có một đường đi ngắn nhất từ S tới v là đường đi cơ bản (qua không quá n-1 cạnh). Tức là d(v) sẽ là độ dài đường đi ngắn nhất từ S tới v.

Vậy số bước lặp tối ưu hóa sẽ không quá n-1 bước.

Ta có thuật toán Ford – Bellman

Procedure Ford_Bellman;

```
Begin
    for (∀v ∈V) do d[v] := +∞;
     d[s] :=0;
     ke_truoc[v] := null;
    While not (stop)  do
  begin
```

```
        for (∀u ∈V) do
          for (∀v ∈V: (u, v) ∈ E) do
            if d[v] > d[u] + c [u, v] then
            begin
              d[v] := d[u] + c[u, v];
              ke_truoc[v] := u;
            end;
        end;
    End;
```

b.Thuật toán Dijkstra

Thuật toán Dijkstra (E.Dijkstra - 1959) có thể mô tả như sau:

Bước 1: Khởi tạo

Với đỉnh v ∈ V, gọi nhãn d[v] là độ dài đường đi ngắn nhất từ s tới v. Ta sẽ tính các d[v]. Ban đầu d[v] được khởi gán bằng w[s, v]. Nhãn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhãn tự do có nghĩa là có thể còn tối ưu hơn được nữa và nhãn cố định tức là d[v] đã bằng độ dài đường đi ngắn nhất từ s tới v nên không thể tối ưu thêm. Để làm điều này ta có thể sử dụng kỹ thuật đánh dấu: Free[v] = TRUE hay FALSE tuỳ theo d[v] tự do hay cố định. Ban đầu các nhãn đều tự do.

Bước 2: Lặp

Cố định nhãn: Chọn trong các đỉnh có nhãn tự do, lấy ra đỉnh u là đỉnh có d[u] nhỏ nhất, và cố định nhãn đỉnh u.

Sửa nhãn: Dùng đỉnh u, xét tất cả những đỉnh v và sửa lại các d[v] theo công thức: d[v] := min(d[v], d[u] + c[u, v])

Bước lặp sẽ kết thúc khi mà đỉnh đích t được cố định nhãn (tìm được đường đi ngắn nhất từ s đến t); hoặc tại thao tác cố định nhãn, tất cả các đỉnh tự do đều có nhãn là +∞ (không tồn tại đường đi).

Có thể đặt câu hỏi, ở thao tác 1, tại sao đỉnh u như vậy được cố định nhãn, giả sử d[u] còn có thể tối ưu thêm được nữa thì tất phải có một đỉnh t mang nhãn tự

do sao cho d[u] > d[t] + c[t, u]. Do trọng số c[t, u] không âm nên d[u] > d[t], trái với cách chọn d[u] là nhỏ nhất. Tất nhiên trong lần lặp đầu tiên thì S là đỉnh được cố định nhãn do d[s] = 0.

Bước 3: Kết hợp với việc lưu vết đường đi trên từng bước sửa nhãn, thông báo đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi (d[t] = +∞). Thuật toán Dijkstra giả mã như sau:

```
Procedure Dijkstra;
 Begin
        for (∀v ∈ V) do d[v] := +∞;
         d[s] := 0;
         S := ∅;
       While t ∉ S do
     begin
          u := (e(u, v) ∉ S) and mind(d[u]);
            S := S ∪{u};
              for  (∀v ∉ S) do
            if d[v] > d[u] + c[u, v] then
          d[v] := d[u] + c[u, v];
    end;
 End;  {d[t] = độ dài đường đi ngắn nhất từ s đến t}
```

2.4.5.5 Thuật toán tô màu đồ thị

a. Định nghĩa

Tô màu một đơn đồ thị là việc gán màu cho các đỉnh của nó sao cho hai đỉnh liền kề có số màu khác nhau.

b. Tư tưởng của thuật toán

Đầu tiên ta xét các đỉnh theo thứ tự và gán cho mỗi đỉnh một màu riêng theo nguyên tắc: Các đỉnh không kề với đỉnh đang xét (không có cạnh nối

trực tiếp) thì được tô cùng một màu, cấm tô màu đó cho các đỉnh có cạnh kề với đỉnh đang xét. Thuật toán lặp lại cho đến khi tất cả các đỉnh được tô màu.

Thuật toán greedy - Tìm gần đúng số màu ít nhất

- Các đỉnh chưa đánh dấu
- Tính bậc các đỉnh
- Thực hiện vòng lặp

- Tô màu có đỉnh cao nhất bằng màu nhỏ nhất chưa dùng và không mâu thuẫn với quy tắc nêu trên.

- Giảm bậc của các đỉnh kề đỉnh này.

Quá trình lặp cho đến khi các đỉnh đều đã được tô màu.

c. Thuật toán tô màu

- Input: Cho đơn đồ thị G = (V, E). với tập đỉnh V={v_1,..,v_n} và tập các đỉnh kề A_{v_j} .

- Output: đồ thị G = (V, E) có các đỉnh đã được gán màu.

Mô tả thuật toán tô màu:

Bước 1: Lập danh sách các đỉnh của đồ thị $E^{'}$=[v_1, v_2, ...,v_n] được sắp xếp theo thứ tự bậc giảm dần: $d(v_1) \geq d(v2) \geq ... d(v_n)$

Đătl i := 1;

Bước 2: Tô màu i cho đỉnh đầu tiên trong danh sách. Duyệt lần lượt các đỉnh tiếp theo và tô màu i cho đỉnh không kề đỉnh đã được tô màu i.

Bước 3: Nếu tất cả các đỉnh đã được tô màu thì kết thúc, đồ thị được tô bằng i màu. Ngược lại, chuyển sang bước 4;

Bước 4: Loại khỏi E' các đỉnh đã tô màu. Sắp xếp lại các đỉnh trong E' theo thứ tự bậc giảm dần.

Đặt i := i +1 và quay lại bước 2.

Thủ tục mô tả thuật toán tô màu

Begin

1. *Sort(v) with deg(v) descending*
2. *Set* $c(v_j)=0$ *với* $\forall j = 1..n$
3. *Set* $c(v_1)=1$
4. *For i=2 to n do*
5. *Set* $c(v_j)=$ *Min*$\{ k \in Z | k>0$ *và* $c(u) \neq k$ *với* $\forall u \in A_{v_j} \}$// *Tô màu* $k>0$ *cho đỉnh* v_j *khác với đỉnh kề của nó*
6. *Lặp lại đến khi tô màu xong*

End;

KẾT LUẬN CHƯƠNG 2

Trong chương 2, luận văn đã trình bày một số thuật toán chọn lọc trong tin học tìm nghiệm các bài toán tối ưu trong thực tế, phân tích độ phức tạp cũng như mô phỏng sơ đồ khối thực hiện các thuật toán này. Đây chính là những thuật toán thường được sử dụng để giảng dạy cho các đội tuyển chuyên tin học tại các trường phổ thông hiện nay. Những thuật toán này sẽ làm cơ sở để thiết kế các chương trình tìm nghiệm tối ưu của các bài toán trong chương 3 của luận văn.

Chương 3
MỘT SỐ ỨNG DỤNG THỰC TẾ

3.1 Bài toán gia công (trình tự gia công ngắn nhất)

(Đề thi chọn học sinh giỏi môn tin học Thành phố Hải Phòng năm 2009)

a. Mô tả bài toán

Trong một phân xưởng sản xuất phụ tùng xe gắn máy có trang bị 2 máy A và B dùng để sản xuất biết rằng để sản xuất 1 phụ tùng, nó cần được gia công lần lượt trên máy A rồi đến máy B.

Yêu cầu: Biết thời gian gia công của mỗi phụ tùng trên máy A và máy B, cho biết trình tự gia công các phụ tùng sao cho thời gian gia công là ngắn nhất.

Dữ liệu: Vào từ file GIACONG.INP

- Dòng đầu ghi số N - số lượng phụ tùng cần gia công. ($1 \le N \le 20$)
- Dòng thứ 2 - ghi các số A[i] (i = 1, 2, .., N) là thời gian gia công phụ tùng thứ i trên máy A.
- Dòng thứ 3 - ghi các số B[i] (i = 1, 2, .., N) là thời gian gia công phụ tùng thứ i trên máy B.

Kết quả: Ghi ra file GIACONG.OUT dãy chỉ số các phụ tùng theo trình tự gia công có thời gian ngắn nhất.

Ví dụ:

GIACONG.INP	GIACONG.OUT
4 1 5 3 4 2 3 2 5	1 4 2 3
5 6 7 8 5 4 9 8 3 2 1	1 2 3 4 5

b. Giải thuật

- Ta lưu trữ thời gian thực hiện N (0<=N<=20) sản phẩm trên máy A và B vào mảng A (Máy A: A[1]; Máy B: A[2]).
- Với sản phẩm thứ i ta xét độ chênh lệch giữa thời gian gia công sản phẩm này trên máy A và thời gian gia công sản phẩm trước đó trên máy B.

- Lưu tổng chêch lệch vào biến CL và xác định giá trị nhỏ nhất bằng biến Min.

- Kết quả được đưa vào mảng Result và in ra khi đệ qui kết thúc.

Thuật toán được mô tả chi tiết trong phụ lục 1 (*Tên chương trình Gia_Cong.Pas*)

3.2 Bài toán xếp việc [5]

(*Đề thi Olympic 30/4/2003 Tỉnh Thừa Thiên Huế***)**

a/. Mô tả bài toán

Có N công việc cần thực hiện trên một máy tính, mỗi việc đòi hỏi đúng 1 giờ máy. Với mỗi việc ta biết thời hạn phải nộp kết quả thực hiện sau khi hoàn thành việc đó và tiền thưởng thu được nếu nộp kết quả trước hoặc đúng thời điểm quy định. Chỉ có một máy tính trong tay, hãy lập lịch thực hiện đủ N công việc trên máy tính sao cho tổng số tiền thưởng thu được là lớn nhất và thời gian hoạt động của máy là nhỏ nhất.

Giả thiết rằng máy được khởi động vào đầu ca, thời điểm t=0 và chỉ tắt máy sau khi đã hoàn thành đủ N công việc.

- Dữ liệu vào: tệp văn bản viec.inp

+ N dòng tiếp theo: Mỗi việc được mô tả bằng hai số tự nhiên, số thứ nhất là thời hạn giao nộp, số thứ hai là tiền thưởng. Các số cách nhau bởi dấu cách.

Ví dụ trên tệp viec.inp sẽ có:

viec.inp
4
1 15
3 10
5 100
1 27

Ý nghĩa: Cho biết có 4 việc với các thông tin sau:

Việc thứ nhất phải nộp không muộn hơn thời điểm 1 (giờ) với tiền thưởng 15 (ngàn đồng);

Việc thứ hai phải nộp không muộn hơn thời điểm 3 (giờ) với tiền thưởng 10 (ngàn đồng);

Việc thứ ba phải nộp không muộn hơn thời điểm 5 (giờ) với tiền thưởng 100 (ngàn đồng);

Việc thứ tư phải nộp không muộn hơn thời điểm 1 (giờ) với tiền thưởng 27 (ngàn đồng).

- Dữ liệu ra: tệp văn bản viec.out:

+ N dòng đầu tiên, dòng thứ t ghi một số tự nhiên i cho biết việc thứ i được làm trong giờ t.

+ Dòng cuối cùng ghi tổng số tiền thu được.

Ví dụ trên tệp viec.out sẽ có:

viec.out	Ý nghĩa:
4	Giờ thứ 1 thực hiện việc 4 và nộp đúng hạn nên được thưởng 27;
2	Giờ thứ 2 thực hiện việc 2 và nộp trước hạn nên được thưởng 10;
3	Giờ thứ 3 thực hiện việc 3 và nộp trước hạn nên được thưởng 100;
1	Giờ thứ 4 thực hiện việc 1;
137	Tổng tiền thưởng thu được do đã hoàn thành đúng hạn ba việc 4, 2 và 3 là 27 + 10 + 100 = 137.

b/. Giải thuật

Ta ưu tiên cho những việc có tiền thưởng cao, do đó ta sắp các việc giảm dần theo tiền thưởng. Với mỗi việc k ta đã biết thời hạn giao nộp việc đó là h = t[k]. Ta xét trục thời gian b. Nếu giờ h trên trục đó đã bận do việc khác thì ta tìm từ thời điểm h trở về trước một thời điểm có thể thực hiện được việc k đó. Nếu tìm được một thời điểm m như vậy, ta đánh dấu bằng mã số của việc đó trên trục thời gian b, b[m]:= k. Sau khi xếp việc xong, có thể trên trục thời gian còn những thời điểm rỗi, ta dồn các việc đã xếp về phía trước nhằm thu được một lịch làm việc, tức là không có giờ trống. Cuối cùng ta xếp tiếp những việc trước đó đã xét nhưng không xếp được. Đây là những việc phải làm nhưng không thể nộp đúng hạn nên sẽ không có tiền thưởng.

Thuật toán được mô tả chi tiết trong phụ lục 1 (*Tên chương trình Xep_Viec.Pas*)

3.3 Bài toán đường đi ngắn nhất

(Đề thi Olympic tin học trẻ Thành phố Hải Phòng năm 2012)

a. Mô tả bài toán

Có N thành phố. Biết rằng đường đi giữa 2 thành phố bất kỳ (nếu có) đều là đường đi hai chiều. Sơ đồ mạng lưới giao thông của N thành phố này cho bởi ma trận trọng số đối xứng A[i,j], trong đó:

+ A[i,j] là độ dài đường đi từ thành phố i đến thành phố j.

+ A[i,j] = 0 nếu không có đường đi từ thành phố i đến thành phố j.

+ A[i,j] = A[j,i] và A[i,i]=0

+ A[i,j] nguyên, không âm.

- Dữ liệu vào: Cho file DDTOIUU.INP gồm N + 2 dòng
  - Dòng đầu là số N (N Nguyên dương, N ≤ 50)
  - Dòng i+1 (1≤ i ≤ N) ghi N số nguyên A[i, 1], A[i, 2],…, A[i, N].
  - Dòng N + 2 ghi 2 số P và Q
- Dữ liệu ra: Ghi ra tệp DDTOIUU.OUT
  - Nếu có đường đi thì thông báo:

  + Dòng đầu cho biết độ dài đường đi ngắn nhất từ P đến Q

  + Dòng thứ i+1 là sơ đồ đường đi.
  - Nếu không có đường đi thông báo:

+ Không có đường đi từ P đến Q

Ví dụ

DDTOIUU.INP	DDTOIUU. OUT
6 0 5 0 0 0 9 5 0 6 0 0 0 0 6 0 7 0 0 0 0 7 0 8 0 0 0 0 8 0 9 9 0 0 0 9 0 1 5	Đường đi ngắn nhất từ 1 đến 5 dài 18 1 ⇒ 6 ⇒ 5
4	Không có đường đi từ 2 đến 4

0 2 3 0 2 0 5 0 3 5 0 0 0 0 0 0 2 4	

b. Giải thuật

Gọi A[i,j] là độ dài đường đi ngắn nhất từ thành phố i đến thành phố j, thì độ dài đường đi ngắn nhất từ thành phố P đến thành phố Q là A[P, Q].

- Nếu có đường đi từ i đến k và từ k đến j (tức là A[i,k]>0 và A[k,j]>0)

+ Nếu A[i,j]=0 tức là không có đường đi trực tiếp từ i đến j.

+ Nếu có đường đi từ i đến j và A[i,j]>A[i,k]+A[k,j] thì gán lại A[i,j]:=A[i,k]+A[k,j];

- Độ dài đường đi ngắn nhất từ thành phố i đến chính nó thì = 0 nghĩa là A[i,j]=0

- Ta xét phương án A[i,j] gồm N dòng, N cột. Lúc đầu A[i,j] là độ dài đường đi từ i đến j (nếu A[i,j] ≠0), nếu không có đường đi từ i đến j thì A[i,j] = 0 nghĩa là mọi phần tử trên đường chéo chính đều = 0.

Gọi C[i,j] là điểm cần đi qua trên đường đi ngắn nhất từ i đến j

Phép duyệt 3 vòng lặp lồng nhau

```
for k:=1 to N do
   for i:=1 to N do
      for j:=1 to N do
```

+ Nếu không có đường đi từ i đến j (A[i,j] = 0)

+ Nếu có đường đi từ i đến k và từ k đến j và độ dài A[i,j]>A[i,k]+A[k,j] thì gán lại A[i,j]:=A[i,k]+A[k,j]; và lưu C[i,j]:=k;

- Dựa vào kết quả của C[i,j] ta thấy C[i,j] = 0 thì A[i,j] không thay đổi còn C[i,j] = k là giá trị A[i,j] đã thay đổi tại bước thứ k. Khi đó giá trị A[P,Q] chính là độ dài ngắn nhất của đường đi từ P đến Q. Nếu giá trị của A[P,Q] = 0 tức là không

có đường đi từ P đến Q. Nếu có đường đi thì việc xác định đường đi được gọi đệ qui trong chương trình xác định đường đi: Procedure Xdduong(P,Q);

Thuật toán được mô tả chi tiết trong phụ lục 1 (*Tên chương trình Duong_Di.Pas*)

3.4 Bài toán mắc dây điện

(Đề thi chọn học sinh giỏi các trường chuyên khu vực duyên hải và đồng bằng bắc bộ năm 2013)

- *Mô tả bài toán*

Một khu dân cư có n hộ dân *(3<n<=500)*, mỗi một nhà dân có một tọa độ nhất định giống như tọa độ trong hệ trục tọa độ đề các. Hiện nay do khu dân cư này mới được xây dựng nên một số nhà thì đã có điện và một số nhà thì chưa có điện.

Yêu cầu đặt ra là mắc dây điện để tất cả các nhà đều có điện và sao cho tổng độ dài đường dây của tất cả các hộ mới nối thêm là nhỏ nhất, các hộ chưa có điện thì có thể nối dây qua hộ đã có điện để cùng có điện.

Dữ liệu vào từ file Daydien.inp:

- Dòng đầu là số n;

- N dòng tiếp theo mỗi dòng là tọa độ và ký hiệu có điện (1) hoặc chưa có điện (0) của từng hộ trong khu dân cư.

Kết quả ghi ra file Daydien.out:

Tổng độ dài đường dây điện cần nối thêm để tất cả các hộ trong khu dân cư đều có điện. Kết quả lấy chính xác 3 chữ số sau dấu chấm thập phân.

Ví dụ:

Daydien.inp	Daydien.out
5 1 1 1 1 2 0 1 3 0 2 3 0 3 4 1	3.000

Thuật toán được mô tả chi tiết trong phụ lục 1 (*Tên chương trình MAC_DAY_DIEN.Pas*)

3.5 Bài toán quân cờ Domino

(Đề thi chọn học sinh giỏi các trường chuyên khu vực duyên hải và đồng bằng bắc bộ năm 2014)

a. Mô tả bài toán

Trong bài toán này, một quân cờ Domino là hình có kích thước 2x1, mỗi ô ghi một số nguyên dương không quá 10. Có N quân cờ Domino xếp thành hàng ngang như hình vẽ dưới đây.

1	6	6	8	6	4	4	3	8	9
6	10	2	8	5	2	1	6	9	5

Yêu cầu: Cho hiện của các quân cờ Domino hãy tìm cách lật các quân cờ Domino (đổi chỗ ô trên với ô dưới) sao cho chênh lệch của tổng các số hàng trên với hàng dưới là nhỏ nhất với một số ít nhất các phép lật.

Dữ liệu: đọc từ tệp văn bản Domino.inp

+ Dòng đầu tiên ghi số nguyên dương N (N<=50)

+ Dòng thứ hai ghi các số ở hàng trên.

+ Dòng thứ ba ghi các số của hàng dưới.

Kết quả: Ghi ra tệp văn bản Domino.out

+ Dòng đầu ghi chêch lệch nhỏ nhất tìm được và số các Domino cần chuyển.

+ Dòng thứ 2 ghi danh sách thứ tự các quân Domino cần chuyển

b. Giải thuật

Ta xây dựng mảng W_i là độ chênh lệch giữa phần trên và phần dưới của thanh Domino thứ i. Gọi C(i,j) là số các phép lật nhỏ nhất với các quân cờ từ 1 đến i, để hàng trên có tổng là j. C(i,j) = min {C(i-1, j-Tren(i)), C(i-1, Duoi(i))+1}.

Từ các C(i,j) ta dễ dàng tìm ra chênh lệch tối thiểu và số lần lật nhỏ nhất.

Thuật toán được mô tả chi tiết trong phụ lục 1 (*Tên chương trình Domino.Pas*)

3.6 Bài toán mạng giao thông:

(Đề thi chọn học sinh giỏi các trường chuyên khu vực duyên hải và đồng bằng bắc bộ năm 2014)

a. Mô tả bài toán

Trong một mạng giao thông liên thông gồm N nút và M đoạn đường giữa một số cặp nút, mỗi đoạn đường đều có giới hạn tải trọng xe. Một xe cần đi từ nút 1 đến nút N. Cần chọn cho xe một hành trình sao cho theo hành trình đó, xe có trọng tải được phép lớn nhất.

Input: được cho bởi file TRANSPO.INP trong đó dòng thứ nhất ghi hai số N, M. Trong M dòng tiếp theo, mỗi dòng ghi ba số nguyên dương, hai số đầu là tên hai nút có đoạn đường nối trực tiếp, số thứ ba là tải trọng cho phép.

Output: Ghi ra dòng thứ nhất của file TRANSPO.OUT tải trọng lớn nhất, dòng thứ hai ghi số S là số nút trên hành trình từ 1 đến N, dòng thứ ba ghi S nút theo thứ tự trên hành trình.

Các hạn chế

- $2 \le n \le 100\ 000$
- $1 \le m \le 200\ 000$
- $1 \le w \le 10\ 000$

TRANSPO.INP	TRANSPO.OUT
6 9 1 2 5 1 4 3 2 4 2 2 3 6 4 5 4 3 4 5 3 5 1 3 6 3 5 6 5	4 6 1 2 3 4 5 6

b. Giải thuật

Áp dụng tư tưởng thuật toán Dijkstra tìm cách đi tốt (tải trọng cho phép (TTCP) lớn nhất) từ 1 đến N. Với mỗi cặp đỉnh I, J, ký hiệu C[I,J] = tải trọng cho phép nếu có cạnh (I,J) và = 0 nếu không có cạnh (I,J). Với mỗi đỉnh I, ký hiệu D[I] là TTCP khi đi từ 1. Các D[I] khởi tạo = 0 trừ D[1] cho lớn tùy ý.

Thuật toán này có hai pha.

- Pha 1. Giả sử Last là đỉnh vừa có đường đi tốt từ 1 đến, với mỗi đỉnh I chưa có đường đi tốt từ 1 đến, so sánh TTCP của đường đi cũ với TTCP của đường đi từ 1 đến I vòng qua Last, nếu D[I]<Min(D[last], C[last,I]) thì thay D[I] bằng Min(D[last], C[last,I]).

- Pha 2. Trong số các đỉnh I chưa có đường đi tốt từ 1 đến, chọn làm Last mới đỉnh có D lớn nhất.

Thuật toán được mô tả chi tiết trong phụ lục 1 (*Tên chương trình Mang_giao_thong.Pas*)

KẾT LUẬN

Thuật toán là một mảng rất rộng, nếu đi hầu hết tất cả vấn đề của thuật toán đó là một khối lượng kiến thức khổng lồ, các vấn đề ứng dụng của thuật toán cũng rất nhiều, phong phú và đa dạng. Trong luận văn đã tập trung nghiên cứu, trình bày những kiến thức cơ bản về thuật toán và ứng dụng của thuật toán vào việc giải một số bài toán tin học, đề thi học sinh giỏi, đề thi olympic. Qua đó luận văn đã đạt được một số kết quả như sau:

Về lý thuyết

Luận văn tập trung nghiên cứu các kiến thức chung nhất về thuật toán và độ phức tạp của thuật toán, một số thuật toán trên mô hình đồ thị. Luận văn đã phân tích kỹ về các thuật toán của các bài toán ứng dụng.

Về ứng dụng

Luận văn đã phân tích và cài đặt các thuật toán của các bài toán ứng dụng trong thi học sinh giỏi, Olympic Tin học.

Phạm vi và khả năng áp dụng

Luận văn là một tài liệu tham khảo tốt cho giáo viên và học sinh trong các trường phổ thông.

Hướng nghiên cứu tiếp theo

Tối ưu hóa các thuật toán và cài đặt nhiều thuật toán cho các bài toán ứng dụng, đồng thời bổ sung một số bài toán mới trong các kỳ thi chọn học sinh giỏi cấp tỉnh, cấp quốc gia và Olympic, …

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Nguyễn Thanh Bình (2007), *Giáo trình - bài giảng thuật toán nâng cao*, trường Đại học Bách khoa - Đại học Đà Nẵng.

[2] Hồ Sĩ Đàm (2009), *Tài liệu giáo khoa chuyên tin*, Nxb Giáo dục.

[3] Nguyễn Hữu Điển (2002), *Một số vấn đề về thuật toán,* Nxb Giáo dục.

[4] Đỗ Đức Giáo (1998), *Toán rời rạc,* Nxb Đại học Quốc gia Hà Nội.

[5] Nguyễn Xuân Huy (2011), *Sáng tạo trong thuật toán và lập trình,* Tập 1, 2, 3, Nxb Thông tin và truyền thông.

[6] Lê Minh Hoàng (2002), *Giải thuật & lập trình*, Đại học sư Phạm Hà Nội.

[7] Đỗ Xuân Lôi (1998), *Cấu trúc dữ liệu và giải thuật,* Nxb Giáo dục.

[8] Nguyễn Xuân My (2002), *Một số vấn đề chọn lọc trong Tin học*, Tập 1, 2 – Nxb Giáo dục.

[9] Nguyễn Đức Nghĩa – Nguyễn Tô Thành (1997), *Toán rời rạc,* Nxb Đại học Quốc gia Hà Nội.

Tiếng Anh

[10] Garey Michael R-David S Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 2007

[11] Silvano Martello, Silvano; Paolo Toth , *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, 2009

Thái Nguyên, ngày ... tháng 6 năm 2015

Xác nhận của giáo viên hướng dẫn **Học viên**

TS. Vũ Vinh Quang **Lê Đình Long**

PHẦN PHỤ LỤC

Một số chương trình nguồn

3.1 Bài toán gia công

```
Program     GIA_CONG;
Uses Crt;
Const   Fi              = 'GIACONG.INP';
        Fo              = 'GIACONG.OUT';
Var     A               : Array [1..2,0..20] of Integer;
        Tg,Result       : Array [0..20] of Byte;
        B               : Array [1..20] of Boolean;
       CL, Min          : Integer;
        N               : Byte;
{-------------------------------------------------------------------------------------}
Procedure       Input;
Var     F: Text;  I,J :Byte;
Begin
   Assign (F,Fi);  Reset (F);  Readln (F,N);
    For I:=1 to 2 do
 Begin
      For J:=1 to N do Read(F,A[I,J]);
       Readln(F);
  End;
    Close(F);
End;
{-------------------------------------------------------------------------------------}
Procedure       Output;
      Var       F:Text;  I:Byte;
  Begin
       Assign(F, Fo);  Rewrite(F);
          For I:=1 to N do  Write(F,Result[I],' ');
       Close(F);
End;
{-------------------------------------------------------------------------------------}
Procedure       Check;
 Begin
      If CL < Min then
       Begin  Result := Tg;  Min :=CL;  End;
End;
{-------------------------------------------------------------------------------------}
Procedure       Solve (M:Byte);
      Var       I :Byte;
Begin
```

```
        If M = N+1 then Begin Check;    Exit;     End;
            For I :=1 to N do
              If B[I] = False then
    Begin
             CL := CL + abs (A[2,Tg[M-1]]-A[1,I]);
               If CL< Min then
        Begin
          Tg[M] := I;   B[I] := True;
            Solve(M+1);
          B[I] := False;
        End;
      CL := CL - abs(A[2,Tg[M-1]]-A[1,I]);
  End;
End;
{-------------------------------------------------------------------------------------------}
BEGIN
   Input;
   FillChar(Tg,SizeOf(Tg),0);
   Fillchar(B,SizeOf(B),False);
   Min:=MaxInt;
   Solve(1);
   Output;
END.
```

3.2 Bài toán xếp việc

```
Program XEP_VIEC;
Uses crt;
Const  MN = 200; bl = #32; nl = #13#10;
          fn = 'viec.inp';   fo = 'viec.out';
Var  a, id, t: array[1..MN] of integer;          {a:t thuong, t:thoi han giao nop id:chi dan}
          h: array[0..MN] of integer;            {truc thoi gian}
          f: text;            N: integer;        {so luong viec}
          M: integer;                            {so viec da xep}
          tt: longint;                           {tong so tien thuong}
(*-------------------------------  Doc du lieu tu input file ------------------------------*)
Procedure Doc;
Var i,k: integer;
  begin
       assign(f,'viec.inp'); reset(f); readln(f,N);
        for i := 1 to N do  readln(f,t[i],a[i]);
       close(f);
  end;
(*---------------------------- Khoi tri cho mang chi dan id ------------------------------*)
Procedure InitID;
   var i: integer;
  begin
```

```
      for i :=  1 to N do id[i] :=  i;
  end;
(*---------------------------- Sap giam a[1..N] theo chi dan-----------------------------*)
Procedure  IDQuickSort(d,c: integer);
  var i, j, m, k: integer;
  begin
       i := d;   j:= c;    m := a[id[(i+j) div 2]];     {phan tu giua}
           while i <= j do
     begin
            while a[id[i]] > m do inc(i);     while a[id[j]] < m do dec(j);
               if i <= j then
             begin
              k:=id[i];          id[i]:=id[j];       id[j]:=k;
              inc(i); dec(j);
            end;
     end;
            if d < j then IDQuickSort(d,j);
            if i < c then IDQuickSort(i,c);
    end;
(*------------------------- Xep viec theo giai thuat tham lam--------------------------*)
Procedure  XepViec;
 var i, k, v: integer;
 begin
    fillchar(h,sizeof(h),0);
    for i :=  1 to N do
   begin
      v :=  id[i];                              {viec nao}
      for k :=  t[v] downto 1 do
      if h[k]= 0 then
    begin                                       {xep duoc viec v tai thoi diem k}
      h[k]:=v;  id[i]:=-v;  break;
   end;      end;
 end;
(*------Don cac viec da xep trong h len phia truoc va tinh tong tien thuong------*)
Procedure DonViec;
  var i: integer;
  begin
     tt := 0;                                   {tim gio trong dau tien trong h}
     for i := 1 to MN do
     if h[i]=0 then
   begin
         M := i;    break;
   end
else tt := tt+a[h[i]];
   if M > N then exit;
     for i := M+1 to MN do
       if h[i] > 0 then
    begin
          h[M] := h[i];       tt := tt+a[h[i]];   inc(M);
       if M > N then exit;
    end;
```

```
  end;
(*-------------------------------------Xep cac viec con lai----------------------------------*)
Procedure XepTiep;
var i: integer;
  begin
    for i := 1 to N do    if id[i] > 0 then
     begin
          h[M] := id[i];     inc(M);
     end;
  end;
(*----------------------------------------  Ghi ket qua  ----------------------------------*)
   Procedure GhiTep;
   var i: integer;
   begin
      assign(f, fo); rewrite(f);
          for i :=  1 to N do writeln(f,h[i]);
        writeln(f, 'Tong so tien thuong thu duoc:', tt);         close(f);
    end;
BEGIN
    Doc;          InitID;        IDQuickSort(1,n);
    XepViec;     DonViec;    XepTiep;     GhiTep;
END.
```

3.3 Bài toán đường đi ngắn nhất

```
Program DUONG_DI;
  Uses Crt;
  Const Max=50;
          Fi                  = 'DDTOIUU.INP';
          Fo                  = 'DDTOIUU.OUT';
  Var   A,C                   : Array[1..Max,1..Max] of Integer;
       N, P, Q,i, j, k        : Integer;
          F                   : Text;
{-------------------------------------------------------------------------------------------}
  Procedure INPUT;
  Begin
      Assign(F,Fi);    Reset(F);    Readln(F,N);
       Fillchar(A,sizeof(A),0);
        For i:=1 to N do
     begin
       For j:=1 to N do Read(F,A[i,j]);
        Readln(F);
     end;
     Readln(F,P,Q);  Close(F);
  End;
{-------------------------------------------------------------------------------------------}
  Procedure XDduong(i,j:Integer);
  Begin
    If C[i,j]=0 Then Write('--> ',j)
    Else Begin
          XDduong(i,C[i,j]);
```

```
        XDduong(C[i,j],j);
       End;
 End;
{-------------------------------------------------------------------------------------}
 Procedure Output;
 Begin
    Assign(f,fo);  Rewrite(f);
    If A[P,Q]=0 Then
     begin
       Writeln('Khong co duong di tu ',P,' den ',Q);
       Writeln(f,'Khong co duong di tu ',P,' den ',Q);
       close(f);
     end
       else
    begin
       Writeln('Duong di ngan nhat tu ',P,' den ', Q ,' dai ',A[P,Q]) ;
       Writeln(f,'Duong di ngan nhat tu',P,' den',Q,' dai ',A[P,Q]) ;
       Write(P);
       XDduong(P,Q);
      Close(f);
   end;
 End;
{-------------------------------------------------------------------------------------}
 Procedure Process;
 Begin
  Clrscr;
  For k:=1 to N do
    For i:=1 to N do
    If A[i,k] >0 Then
    For j:=1 To N do
    If (A[j,k] >0) And (i<>j) Then
    If (A[i,j]=0) or (A[i,j]>A[i,k]+A[k,j]) Then
    Begin
      A[i,j]:=A[i,k]+A[k,j];     C[i,j]:=k;
    End;
End;
{-------------------------------------------------------------------------------------}
BEGIN
         Input;
         Process;
         Output;
         Readln;
END.
```

3.4 Bài toán mắc dây điện

```
Program MAC_DAY_DIEN;
Uses crt;
Const    fi        = 'Daydien.inp';          fo        = 'Daydien.out';
       nm          = 501;
       mm          = 100000000;
```

```
Type xyz          = Record
     x,y   :real; z  : Integer;
     end;
Var a             : Array[1..nm]of xyz;
    i, j, n, vti,vtj  : Integer;
    d, min,t      : Real;
    f             : Text;
{-------------------------------------------------------------------------------------------}
Function kt(x,y:real):boolean;
begin
     for j:=1 to i-1 do
      if (a[j].x=x)and(a[j].y=y)then
       begin
            kt:=false;
            exit;
       end;
       kt:=true;
end;
{-------------------------------------------------------------------------------------------}
Procedure nhap1;
begin
     randomize;
     write ('Vao n='); readln(n);
     assign(f,fi); rewrite(f);
     writeln(f,n);
     for i:=1 to n do
        Repeat
           a[i].x:=random(10*n);
           a[i].y:=random(10*n);
           a[i].z:=random(2);
       Until kt(a[i].x,a[i].y);
         for i:=1 to n do
         writeln(f,a[i].x:0:0,' ',a[i].y:0:0,' ',a[i].z);
     close(f);
end;
{-------------------------------------------------------------------------------------------}
Procedure nhap;
  begin
     assign(f,fi); reset(f);     readln(f,n);
     for i:=1 to n do
      begin
           readln(f,a[i].x,a[i].y,a[i].z);
      end;
       close(f);
   end;
Function kc(i,j:integer):real;
  begin
     kc:=1.0*sqrt(sqr(a[i].x-a[j].x)+sqr(a[i].y-a[j].y));
end;
{-------------------------------------------------------------------------------------------}
Procedure xuly;
```

```
begin
    d:=0;
    Repeat
         min:=mm;
         for i:=1 to n do
         for j:=1 to n do
        if ((a[i].z=0)and(a[j].z=1))or((a[i].z=1)and(a[j].z=0))then
            begin
                t:=kc(i,j);
                if t<min then
                   begin
                     min:=t;
                     vti:=i; vtj:=j;
                   end;
             end;
          if min<mm then
            begin
              d:=d+min;
              a[vti].z:=1; a[vtj].z:=1;
           end else break;
    Until false;
       assign(f,fo);   rewrite(f);  write(f,d:0:3);
       close(f);
  end;
{----------------------------------------------------------------------------------------}
Begin
    Nhap;
    xuly;
End.
```

3.5 Bài toán quân cờ Domino

```
Program Domino;
Const   nmax    = 50;
        fi      = ‘domino.inp';
        fo      = ‘domino.out';
        inf     =  nmax*100;

Var  up, down   : Array[1..nmax] of longint;
     n,sum      : Longint;
     c          : Array[1..nmax,1..nmax*10] of longint;
     flip       : Array[1..nmax,1..nmax * 10] of boolean;
     f          : Text;
 {c[i,j] la so phep lat toi thieu cac quan domini tu 1 den i, de co tong hang tren bang j}
{----------------------------------------------------------------------------------------}
Procedure readf;
  Var i : longint;
   begin
        assign(f,fi);      reset(f);          readln(f,n);
        sum := 0;
        for i:= 1 to n do
```

```
            begin
               read(f,up[i]); sum := sum + up[i];
            end;
               readln(f);
               for i := 1 to n do begin
               read(f,down[i]); sum := sum + down[i];
      end;
    close(f);
 end;
{-----------------------------------------------------------------------------------------}
Procedure CalC;
  Var i,j,tmp : Longint;
      begin
            for i := 1 to n do
            for j := 1 to n * 10 do
           begin
             c[i,j] := inf;
             flip[i,j] := false;
           end;
            tmp := 0;
            for i := 1 to n do
       begin
            tmp := tmp + up[i];
            c[i,tmp] := 0;
       end;
           c[1,down[1]] := 1;          flip[1,down[1]] := true;
           c[1,up[1]] := 0;            flip[1,up[1]] := false;
           for i := 2 to n do
           for j :=1 to n*10 do
     begin
         if j-up[i] > 0 then
         c[i,j] := c[i-1,j-up[i]];
         if (j-down[i] > 0) and (c[i,j] > c[i-1,j-down[i]] + 1 ) then
           begin
             c[i,j] := c[i-1,j-down[i]] + 1;
             flip[i,j] := true;
           end;
   end;
end;
{-----------------------------------------------------------------------------------------}
Procedure writef;
  var i, j, min, lj, t          : longint;
                tmp             : array[1..nmax] of longint;
  begin
      min := inf;
      for j := 1 to n * 10 do
      if (c[n,j] < inf) and (abs(sum - j - j ) < min) then begin
      min := abs(sum - j - j);
      lj := j;
  end;
      assign(f,fo);    rewrite(f);  write(f,min,' ');
```

```
          t := 0;
          j := lj;
          i := n;
         while j > 0 do
   begin
        if flip[i,j] = true then begin
        inc(t); tmp[t] := i;
        j := j - down[i];
   end
    else
      j := j-up[i];
      i := i-1;
  end;
      writeln(f,t);
      for i:= t downto 1  do
      write(f,tmp[i], ' ');
      close(f);
 end;
{-----------------------------------------------------------------------------------------}
Begin
   Readf;
   Calc;
   Writef;
End.
```

3.6 Bài toán mạng giao thông

```
Program  MANG_GIAO_THONG;
Const   fi                  = 'TRANSPO.INP';
          Fo                = 'TRANSPO.OUT';
       Type       link      = ^dinh;
          dinh               = Record
          x,p                : Word;
          next               : Link;
        end;
          mm                 = Array[1..1000] of word;
Var
          A                  : Array[1..1000] of link;
          ht,q,truoc         : mm;
  n,m,min,max,ds,s           : Word;
  p                          : Pointer;
  f:      text;     t        : Char;
{-----------------------------------------------------------------------------------------}
Procedure add(i,j,p:word);
 Var
    z:link;
Begin
          new(z);
          z^.x:=j; z^.p:=p;  z^.next:=a[i];
          a[i]:=z;
      end;
```

```
Procedure nhap;
    Var     i,j,k,p :Word;
begin
           assign(f,fi);reset(f);
              readln(f,n,m);
              for i:=1 to n do a[i]:=nil;
              min:=65535;max:=0;
          for k:=1 to m do
     begin
            read(f,i,j,p);
               if min>p then min:=p;
               if max<p then max:=p;
               add(i,j,p);add(j,i,p);
             end;
       close(f);
  end;
{----------------------------------------------------------------------------------------}
Function ke(u,v:word):word;
  Var    z  :link;
begin
          z:=a[u];
          While z<>nil do
     begin
               if z^.x=v then begin ke:=z^.p;exit;end;
               z:=z^.next;
     end;
  ke:=0;
   end;
{----------------------------------------------------------------------------------------}
Function bfs(r:word;var truoc:mm):boolean;
  Var    dau,cuoi,i,j:word;
begin
          fillchar(truoc,sizeof(truoc),0);
          dau:=1;cuoi:=1;q[1]:=1;truoc[1]:=1;
          While dau<=cuoi do
     begin
            i:=q[dau];inc(dau);
            for j:=1 to n do if (truoc[j]=0) and (ke(i,j)>=r) then
        begin
               truoc[j]:=i;
               if j=n then begin bfs:=true;exit;end;
               inc(cuoi);q[cuoi]:=j;
         end;
     end;
          bfs:=false;
    end;
{----------------------------------------------------------------------------------------}
Function giua(x,y:word):word;
     begin
          if (x mod 2 = 0) and (y mod 2 = 0) then
     begin giua:= x div 2 + y div 2;exit;    end;
```

```
          if (odd(x)) and (y mod 2 = 0) then
     begin giua:= x div 2 + y div 2;    exit;     end;
          if (x mod 2 = 0) and (odd(y)) then
     begin giua:= x div 2 + y div 2;    exit;    end;
          if (odd(x)) and (odd(y)) then
     begin giua:= x div 2 + y div 2 +1;  exit;   end;
end;
{-------------------------------------------------------------------------------------------}
Procedure lam;
  Var    r,l,k:word;
begin
          l:=min;r:=max;
          While l<r do
    begin
          k :=giua(l,r);
          if bfs(k,truoc) then l:=k else r:=k;
          if (r=l+1) then
          if bfs(r,truoc) then
      begin
          ds :=r;    bfs(ds,truoc);  exit;
      end
    else
          begin
                    ds :=l;   bfs(ds,truoc);   exit;
    end;
    end;
          ds :=l;
          bfs(ds,truoc);
    end;
{-------------------------------------------------------------------------------------------}
Procedure viet;
  Var    i   :word;
begin
          s :=1;    i :=n;    ht[s] :=i;
          while i<>1 do
     begin
          i :=truoc[i];
          inc(s);  ht[s] :=i;
     end;
          assign(f,fo+t);    rewrite(f);   writeln(f,ds);   writeln(f,s);
          for i:=s downto 1 do write(f,ht[i],' ');
          close(f);
   end;
{-------------------------------------------------------------------------------------------}
BEGIN
   new(p);
   nhap;lam;viet;
   dispose(p);
END.
```