THIẾT KẾ GIẢI THUẬT

Nội dung của chương này trình bày hai chiến lược thiết kế thuật giải thông dụng là vét cạn và tham lam. Nội dung của chương, ngoài phần trình bày về các phương pháp còn có những ví dụ cụ thể, cả thuật giải và cài đặt, để người đọc có một cái nhìn chi tiết về việc từ thuật toán đến chương trình.

1. Vét cạn (Exhausted search)

Vét cạn, duyệt, quay lui… là một số tên gọi tuy không đồng nghĩa nhưng cùng chỉ một phương pháp rất đơn giản trong tin học: **tìm nghiệm của một bài toán bằng cách xem xét tất cả các phương án có thể**. Đối với con người phương pháp này thường là không khả thi vì số phương án cần kiểm tra quá lớn. Tuy nhiên đối với máy tính, nhờ tốc độ xử lí nhanh, máy tính có thể giải rất nhiều bài toán bằng phương pháp vét cạn.

Ưu điểm lớn nhất của phương pháp vét cạn là **luôn đảm bảo tìm ra nghiệm chính xác**. Ngoài ra phương pháp vét cạn còn có một số ưu điểm so với các phương pháp khác là đòi hỏi rất ít bộ nhớ và cài đặt đơn giản. Hạn chế duy nhất của phương pháp này là **thời gian thực thi rất lớn**, độ phức tạp thường ở bậc mũ. Do đó vét cạn thường chỉ áp dụng tốt với các bài toán có kích thước nhỏ.

1.1. Bài toán tìm cấu hình tổ hợp

Thường những bài toán trong Tin học có yêu cầu dạng: tìm các đối tượng x thoả mãn những điều kiện nhất định trong một tập S các đối tượng cho trước. Bài toán tìm cấu hình tổ hợp là bài toán yêu cầu tìm các đối tượng x có dạng là một vector thoả mãn các điều kiện sau:

1. Đối tượng x gồm n phần tử: $x = (x_1,x_2,\ldots x_n)$.
2. Mỗi phần tử x_i có thể nhận một trong các giá trị rời rạc $a_1, a_2, \ldots a_m$.
3. x thoả mãn các ràng buộc có thể cho bởi hàm logic G(x).

Tuỳ từng trường hợp mà bài toán có thể yêu cầu: tìm một nghiệm, tìm tất cả nghiệm hoặc đếm số nghiệm.

Trước hết chúng ta nhắc lại một số cấu hình tổ hợp cơ bản.

a) Tổ hợp

Một tổ hợp chập k của n là một tập con k phần tử của tập n phần tử.

Chẳng hạn tập {1,2,3,4} có các tổ hợp chập 2 là: {1,2}, {1,3, {1,4, {2,3}, {2,4}, {3,4}. Vì trong tập hợp các phần tử không phân biệt thứ tự nên tập {1,2} cũng là tập {2,1} và do đó, ta coi chúng chỉ là một tổ hợp.

Bài toán đặt ra cho chúng ta là **hãy xác định tất cả các tổ hợp chập k của tập n phần tử**. Để đơn giản ta chỉ xét bài toán tìm các tổ hợp của tập các số nguyên từ 1 đến n. Đối với một tập hữu hạn bất kì, bằng cách đánh số thứ tự của các phần tử, ta cũng đưa được về bài toán đối với tập các số nguyên từ 1 đến n.

Nghiệm cần tìm của bài toán tìm các tổ hợp chập k của n phần tử phải thoả mãn các điều kiện sau:

1. Là một vector $x = (x_1, x_2, \ldots x_k)$
2. x_i lấy giá trị trong tập $\{1,2,\ldots n\}$
3. Ràng buộc: $x_i < x_{i+1}$ với mọi giá trị i từ 1 đến k-1.

Có ràng buộc 3 là vì tập hợp không phân biệt thứ tự phần tử nên ta sắp xếp các phần tử theo thứ tự tăng dần.

b) Chỉnh hợp lặp

Chỉnh hợp lặp chập k của n là một dãy k thành phần, mỗi thành phần là một phần tử của tập n phần tử, có xét đến thứ tự và không yêu cầu các thành phần khác nhau.

Một ví dụ dễ thấy nhất của chỉnh hợp lặp là các dãy nhị phân. Một dãy nhị phân độ dài m là một chỉnh hợp lặp chập m của tập 2 phần tử {0,1}. Chẳng hạn 101 là một dãy nhị phân độ dài 3. Ngoài ra ta còn có 7 dãy nhị phân độ dài 3 nữa là 000, 001, 010, 011, 100, 110, 111. Vì có xét thứ tự nên dãy 101 và dãy 011 là 2 dãy khác nhau.

Như vậy, bài toán **xác định tất cả các chỉnh hợp lặp chập k của tập n phần tử** yêu cầu tìm các nghiệm như sau:

1. Là một vector $x = (x_1, x_2, \ldots x_k)$
2. x_i lấy giá trị trong tập $\{1,2,\ldots n\}$
3. Không có ràng buộc nào giữa các thành phần.

Chú ý là cũng như bài toán tìm tổ hợp, ta chỉ xét đối với tập n số nguyên từ 1 đến n. Nếu tập hợp cần tìm chỉnh hợp không phải là tập các số nguyên từ 1 đến n thì ta có thể đánh số các phần tử của tập đó để đưa về tập các số nguyên từ 1 đến n

c) Chỉnh hợp không lặp

Khác với chỉnh hợp lặp là các thành phần được phép lặp lại, tức là có thể giống nhau, chỉnh hợp không lặp chập k của tập n phần tử cũng là một dãy k thành phần lấy từ tập n phần tử có xét thứ tự nhưng các thành phần không được phép giống nhau.

Chẳng hạn có n người, một cách chọn ra k người để xếp thành một hàng là một chỉnh hợp không lặp chập k của n.

Một trường hợp đặc biệt của chỉnh hợp không lặp là hoán vị. Hoán vị của một tập n phần tử là một chỉnh hợp không lặp chập n. Nói một cách trực quan thì hoán vị của tập n phần tử là phép thay đổi vị trí của các phần tử (do đó mới gọi là hoán vị).

Nghiệm của bài toán **tìm các chỉnh hợp không lặp chập k của tập n số nguyên từ 1 đến n** là các vector x thoả mãn các điều kiện:

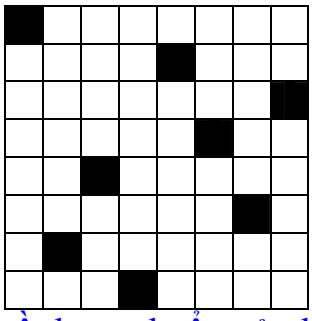
1. x có k thành phần: $x = (x_1,x_2,\ldots xk)$
2. Các giá trị xi lấy trong tập {1,2,..n}
3. Ràng buộc: các giá trị xi đôi một khác nhau, tức là $xi \neq xj$ với mọi $i \neq j$.

Đó là một số bài toán tìm cấu hình tổ hợp cơ bản. Chúng ta sẽ xem xét một số bài toán khác để thấy tính phổ biến của lớp các bài toán dạng này.

d) Bài toán xếp hậu

Cho bàn cờ vua nxn. Hãy xếp n con hậu lên bàn cờ sao cho không con nào khống chế con nào. Hai 2 con hậu khống chế nhau nếu chúng ở trên cùng một hàng, một cột hoặc một đường chéo.

Chẳng hạn ta có một cách đặt sau, các ô đen là các vị trí đặt hậu:



Để chuyển bài toán này về dạng chuẩn của bài toán tìm cấu hình tổ hợp, ta có có nhận xét: mỗi con hậu phải ở trên một hàng và một cột. Do đó ta coi con hậu thứ i ở hàng i và nếu biết x[i] là cột đặt con hậu thứ i thì ta suy ra được lời giải. Vậy nghiệm của bài toán có thể coi là một vector x gồm n thành phần với ý nghĩa:

1. Con hậu thứ i được đặt ở hàng i và cột x[i].
2. x[i] lấy giá trị trong tập {1,2…n}
3. Ràng buộc: các giá trị x[i] khác nhau từng đôi một và không có 2 con hậu ở trên cùng một đường chéo.

Trong phần cài đặt, chúng ta sẽ phân tích chi tiết về các ràng buộc trên.

e) Bài toán từ đẹp (xâu ABC)

Một từ đẹp là một xâu độ dài n chỉ gồm các kí tự A,B,C mà không có 2 xâu con liên tiếp nào giống nhau. Chẳng hạn ABAC là một từ đẹp độ dài 4, BABCA là một từ đẹp độ dài 5.

Bài toán tìm tất cả các từ đẹp độ dài n cho trước yêu cầu tìm nghiệm là các vector x có n thành phần:

1. xi nhận giá trị trong tập {A,B,C}
2. x không có 2 đoạn con liên tiếp nào bằng nhau.

Trước khi trình bày về phương pháp vét cạn giải các bài toán tìm cấu hình tổ hợp, chúng ta xem xét các bài toán tối ưu tổ hợp, vì các bài toán tối ưu tổ hợp thực chất là sự mở rộng của bài toán tìm cấu hình tổ hợp.

1.2. Bài toán tối ưu tổ hợp

Bài toán tối ưu tổng quát có thể phát biểu như sau: Cho tập B khác rỗng và một hàm f:B→R gọi là hàm mục tiêu. Cần tìm phần tử x thuộc B sao cho f(x) đạt giá trị nhỏ nhất hoặc lớn nhất. Phần tử x là nghiệm của bài toán còn được gọi là phương án tối ưu.

Bài toán tối ưu tổ hợp là bài toán tìm phương án tối ưu trên tập các cấu hình tổ hợp. Nghiệm của bài toán cũng là một vector x gồm n thành phần sao cho:

1. $x = (x_1,x_2,\ldots xn)$
2. xi lấy giá trị trong tập $\{a_1,a_2,\ldots am\}$
3. x thoả mãn các ràng buộc cho bởi hàm G(x).
4. $f(x) \rightarrow \min/\max$.

Chúng ta sẽ phân tích một số bài toán tối ưu tổ hợp điển hình. Phần lớn đều là các bài toán NPC.

a) Bài toán xếp balô

Có một balô có tải trọng m và n đồ vật, đồ vật i có trọng lượng wi và có giá trị vi. Hãy lựa chọn các vật để cho vào balô sao cho tổng trọng lượng của chúng không quá M và tổng giá trị của chúng là lớn nhất.

Mỗi cách chọn các đồ vật cho vào balô đều tương ứng với một vector x gồm n thành phần mà xi=1 nếu chọn đưa vật thứ i vào balô, và xi=0 nếu vật thứ i không được chọn.

Khi đó ràng buộc tổng trọng lượng các đồ vật không quá tải trọng của balô được viết thành:

$$\sum_{i=1}^{n} x_i w_i \le m$$

Hàm mục tiêu là tổng giá trị của các đồ vật được chọn:

$$f(x) = \sum_{i=1}^{n} x_i v_i \to \max$$

Nghiệm của bài toán cũng là một vector x gồm n thành phần sao cho:

1. $x = (x_1, x_2, \ldots x_n)$
2. xi lấy giá trị trong tập {0,1}
3. Ràng buộc: $\sum_{i=1}^{n} x_i w_i \le m$
4. $f(x) = \sum_{i=1}^{n} x_i v_i \to \max$.

b) Bài toán người du lịch

Có n thành phố, d[i,j] là chi phí để di chuyển từ thành phố i đến thành phố j. (Nếu không có đường đi thì $d[i,j] = \infty$). Một người muốn đi du lịch qua tất cả các thành phố, mỗi thành phố một lần rồi trở về nơi xuất phát sao cho tổng chi phí là nhỏ nhất. Hãy xác định một đường đi như vậy.

Phương án tối ưu của bài toán cũng là một vector x, trong đó xi là thành phố sẽ đến thăm tại lần di chuyển thứ i. Các điều kiện của x như sau:

1. $x = (x_1, x_2, \ldots x_n)$
2. xi lấy giá trị trong tập {1,2,…n}
3. Ràng buộc: $x_i \ne x_j$ với mọi $i \ne j$ và $d[x_i, x_{i+1}] < \infty$ với mọi i=1,2,..n, coi $x_{n+1} = x_1$.
4. $f(x) = \sum_{i=1}^{n} d[x_i, x_{i+1}] \to \min$

Trên đây ta đã xét một số bài toán tìm cấu hình tổ hợp và bài toán tối ưu tổ hợp. Trong phần tiếp chúng ta sẽ tìm hiểu phương pháp vét cạn giải các bài toán đó.

1.3. Phương pháp vét cạn giải các bài toán cấu hình tổ hợp và tối ưu tổ hợp

Phương pháp vét cạn là phương pháp rất tổng quát để đơn giản để giải các bài toán cấu hình tổ hợp và tối ưu tổ hợp. ý tưởng cơ bản là: bằng một cách nào đó sinh ra tất cả các cấu hình có thể rồi phân tích các cấu hình bằng các hàm ràng buộc và hàm mục tiêu để tìm phương án tối ưu (do đó phương pháp này còn được gọi là duyệt toàn bộ).

Dựa trên ý tưởng cơ bản đó, người ta có 2 cách tiếp cận khác nhau để duyệt toàn bộ các phương án.

Phương pháp thứ nhất là phương pháp sinh tuần tự. Phương pháp này cần xác định một quan hệ thứ tự trên các cấu hình (gọi là thứ tự từ điển) và một phép biến đổi để biến một cấu hình thành cấu hình ngay sau nó. Mỗi lần sinh được một cấu hình thì tiến hành định giá, so sánh với cấu hình tốt nhất đang có và cập nhật nếu cấu hình mới tốt hơn.

Giả mã của thuật toán tìm cấu hình tối ưu bằng phương pháp sinh như sau:

```
Procedure sinh;
begin
   x := cau_hinh_dau_tien;
   best := x;
   Repeat
      x := Cau_hinh_tiep_theo(x);
      if f(x) "tốt hơn" f(best) then best := x;
   Until x = cau_hinh_cuoi_cung;
end;
```

Thuật toán thực hiện như sau: tìm cấu hình đầu tiên và coi đó là cấu hình tốt nhất. Sau đó lần lượt sinh các cấu hình tiếp theo, mỗi lần sinh được một cấu hình ta so sánh nó với cấu hình tốt nhất hiện có (best) và nếu nó tốt hơn thì cập nhật best. Quá trình dừng lại khi ta sinh được cấu hình cuối cùng. Kết quả ta được phương án tối ưu là best.

Phương pháp sinh tuần tự thường rất khó áp dụng. Khó khăn chủ yếu là do việc xác định thứ tự từ điển, cấu hình đầu tiên, cấu hình cuối cùng và phép biến đổi một cấu hình thành cấu hình tiếp theo thường là không dễ dàng.

Phương pháp thứ hai là phương pháp quay lui đệ quy. Tư tưởng cơ bản của phương pháp là xây dựng từng thành phần của cấu hình, tại mỗi bước xây dựng đều kiểm tra các ràng buộc và chỉ tiếp tục xây dựng các thành phần tiếp theo nếu các thành phần hiện tại là thoả mãn. Nếu không còn phương án nào để xây dựng thành phần hiện tại thì quay lui, xây dựng lại các thành phần trước đó.

Mô hình cơ bản của phương pháp quay lui đệ quy như sau:

```
Procedure Search;
begin
```

```
    Try(1);
end;
procedure Try(i);
var j;
Begin
    for j := 1 to m do
        if <chọn được a[j]> then begin
            x[i] := a[j];
            <ghi nhận trạng thái mới>;
            if i=n then Update
            else Try(i+1);
            <trả lại trạng thái cũ>;
        end;
end;
procedure Update;
begin
    if f(x) "tốt hơn" f(best) then best := x;
end;
```

Để duyệt toàn bộ các cấu hình, ban đầu ta gọi đến Try(1). Try(1) sẽ lựa chọn cho x_1 một giá trị thích hợp đầu tiên, ghi nhận trạng thái rồi gọi đệ quy đến Try(2). Try(2) lại lựa chọn một giá trị cho x_2, ghi nhận trạng thái và gọi đến Try(3). Cứ như vậy ở bước thứ i, thuật toán tìm một giá trị cho xi, ghi nhận trạng thái rồi gọi đệ quy để sinh thành phần xi_{+1}. Khi sinh đủ n thành phần của x thì dừng lại để cập nhật phương án tối ưu. Nếu mọi khả năng của xi_{+1} đều đã xét qua thì vòng for của Try(i+1) thực hiện xong, theo cơ chế đệ quy chương trình sẽ quay về điểm gọi đệ quy của Try(i). Trạng thái cũ trước khi chọn xi được phục hồi và vòng for của Try(i) sẽ tiếp tục để chọn giá trị phù hợp tiếp theo của xi, đó chính là thao tác quay lui. Khi quay lui về đến Try(1) và xét hết mọi khả năng của x_1 thì chương trình con đệ quy kết thúc và ta đã duyệt được toàn bộ các cấu hình.

Trên đây là các thuật toán vét cạn đối với bài toán tìm cấu hình tối ưu. Trong trường hợp bài toán cần tìm một cấu hình, tìm mọi cấu hình hay đếm số cấu hình thì thuật toán cũng tương tự, chỉ khác ở phần cập nhật (Update) khi sinh được một cấu hình mới.

Chẳng hạn thủ tục Update đối với bài toán tìm và đếm mọi cấu hình sẽ tăng số cấu hình và in ra cấu hình vừa tìm được:

```
procedure Update;
begin
   count := count + 1;
   print(x);
end;
```

Chúng ta sẽ dùng thuật toán quay lui đệ quy để giải các bài toán cấu hình tổ hợp và tối ưu tổ hợp đã trình bày ở trên.

a) Sinh các tổ hợp chập k của n

Đây là bài toán sinh tổ hợp đã được chúng ta trình bày ở phần trên. Ta sẽ giải bằng thuật toán tìm cấu hình tổ hợp bằng đệ quy quay lui.

Về cấu trúc dữ liệu ta chỉ cần một mảng x để biểu diễn tổ hợp. Ràng buộc đối với giá trị x[i] là: $x[i-1] < x[i] \le n-k+i$. Thủ tục đệ quy sinh tổ hợp như sau:

```
procedure Try(i);
var j;
begin
   for j := x[i–1]+1 to n–k+i do begin
      x[i] := j;
      if i=k then Print(x)
      else Try(i+1);
   end;
end;
```

Dưới đây là toàn văn chương trình sinh tổ hợp viết bằng ngôn ngữ Pascal. Để đơn giản, các giá trị n,k được nhập từ bàn phím và các tổ hợp được in ra màn hình. Người đọc có thể cải tiến chương trình để nhập/xuất ra file.

```
program SinhTohop;
uses crt;
const
   max = 20;
var
   n,k : integer;
   x : array[0..max] of integer;
```

```
{================================}
  procedure input;
  begin
     clrscr;
     write('n,k = '); readln(n,k);
     writeln('Cac to hop chap ',k,' cua ',n);
  end;

  procedure print;
  var
     i : integer;
  begin
     for i := 1 to k do write(' ',x[i]);
     writeln;
  end;

  procedure try(i:integer);
  var
     j : integer;
  begin
     for j := x[i-1]+1 to n-k+i do begin
        x[i] := j;
        if i = k then Print
        else try(i+1);
     end;
  end;

  procedure solve;
  begin
     x[0] := 0;
     try(1);
  end;
{================================}
```

```
BEGIN
   input;
   solve;
END.
```

Chú ý trong phần cài đặt là có khai báo thêm phần tử x[0] để làm "lính canh", vì vòng lặp trong thủ tục đệ quy có truy cập đến x[i−1], và khi gọi Try(1) thì sẽ truy cập đến x[0].

b) Sinh các chỉnh hợp lặp chập k của n

Xem lại phân tích của bài toán sinh chỉnh hợp lặp chập k của n ta thấy hoàn toàn không có ràng buộc nào đối với cấu hình sinh ra. Do đó, cấu trúc dữ liệu của ta chỉ gồm một mảng x để lưu nghiệm. Thuật toán sinh chỉnh hợp lặp như sau:

```
procedure Try(i);
var j;
begin
   for j := 1 to n do begin
      x[i] := j;
      if i=k then Print(x)
      else Try(i+1);
   end;
end;
```

Dưới đây là chương trình sinh tất cả các dãy nhị phân độ dài n. Để đơn giản, chương trình nhập n từ bàn phím và in các kết quả ra màn hình.

```
program SinhNhiphan;
uses crt;
const
   max = 20;
var
   n : integer;
   x : array[1..max] of integer;
{================================}
   procedure input;
   begin
      clrscr;
```

```
    write('n = '); readln(n);
    writeln('Cac day nhi phan do dai ',n);
  end;

  procedure print;
  var
    i : integer;
  begin
    for i := 1 to n do write(' ',x[i]);
    writeln;
  end;
  procedure try(i:integer);
  var
    j : integer;
  begin
    for j := 0 to 1 do begin
      x[i] := j;
      if i = n then Print
      else try(i+1);
    end;
  end;

  procedure solve;
  begin
    try(1);
  end;
{================================}
BEGIN
  input;
  solve;
END.
```

c) Sinh các chỉnh hợp không lặp chập k của n

Chỉnh hợp không lặp yêu cầu các phần tử phải khác nhau. Để đảm bảo điều đó, ngoài mảng x, ta sẽ dùng thêm một cấu trúc dữ liệu nữa là mảng d để đánh dấu. Khi một giá trị được chọn, ta đánh dấu giá trị đó, và khi chọn, ta chỉ chọn các giá trị chưa đánh dấu. Mảng d sẽ là "trạng thái" của thuật toán. Bạn đọc xem phần giả mã dưới đây để thấy rõ hơn ý tưởng đó.

```
procedure Try(i);
var j;
begin
   for j := 1 to n do
      if d[j]=0 then begin
         x[i] := j; d[j] := 1;
         if i=k then Print(x)
         else Try(i+1);
         d[i] := 0;
      end;
end;
```

Chương trình dưới đây sẽ sinh toàn bộ các hoán vị của tập n số nguyên từ 1 đến n. Giá trị n được nhập từ bàn phím và các hoán vị được in ra màn hình.

```
program SinhHoanvi;
uses crt;
const
   max = 20;
var
   n : integer;
   x,d : array[1..max] of integer;
{================================}
   procedure input;
   begin
      clrscr;
      write('n = '); readln(n);
      writeln('Cac hoan vi cua day ',n);
   end;
```

```
    procedure print;
    var
       i : integer;
    begin
       for i := 1 to n do write(' ',x[i]);
       writeln;
    end;

    procedure try(i:integer);
    var
       j : integer;
    begin
       for j := 1 to n do
          if d[j] = 0 then begin
             x[i] := j; d[j] := 1;
             if i = n then Print
             else try(i+1);
             d[j] := 0;
       end;
    end;

    procedure solve;
    begin
       try(1);
    end;
{================================}
BEGIN
    input;
    solve;
END.
```

d) Bài toán xếp hậu

Khác với những bài toán sinh các cấu hình đơn giản ở phần trước, sinh các cấu hình của bài toán xếp hậu đòi hỏi những phân tích chi tiết hơn về các điều kiện ràng buộc.

Ràng buộc thứ nhất là các giá trị x[i] phải khác nhau. Ta có thể dùng một mảng đánh dấu như ở thuật toán sinh hoán vị để đảm bảo điều này.

Ràng buộc thứ 2 là các con hậu không được nằm trên cùng một đường chéo chính và phụ. Các bạn có thể dễ dàng nhận ra rằng 2 vị trí (x_1,y_1) và (x_2,y_2) nằm trên cùng đường chéo chính nếu:

$$x_1-y_1=x_2-y_2=\text{const.}$$

Tương tự, 2 vị trí (x_1,y_1) và (x_2,y_2) nằm trên cùng đường chéo phụ nếu:

$$x_1+y_1=x_2+y_2=\text{const}$$

Do đó, con hậu i đặt tại vị trí (i,x[i]) và con hậu j đặt tại vị trí (j,x[j]) phải thoả mãn ràng buộc:

$$i-x[i] \neq j-x[j] \text{ và } i+x[i] \neq j+x[j] \text{ với mọi } i \neq j$$

Ta có thể viết riêng một hàm Ok để kiểm tra các ràng buộc đó. Nhưng giải pháp tốt hơn là dùng thêm các mảng đánh dấu để mô tả rằng một đường chéo chính và phụ đã có một con hậu khống chế. Tức là khi ta đặc con hậu i ở vị trí (i,j), ta sẽ đánh dấu đường chéo chính i-j và đường chéo phụ i+j.

Như vậy về cấu trúc dữ liệu, ta dùng 4 mảng:

1. Mảng x với ý nghĩa: x[i] là cột ta sẽ đặt con hậu thứ i.
2. Mảng cot với ý nghĩa: cot[j]=1 nếu cột j đã có một con hậu được đặt, ngược lại thì cot[j]=0.
3. Mảng dcc với ý nghĩa: dcc[k]=1 nếu đường chéo chính thứ k đã có một con hậu được đặt, tức là ta đã đặt một con hậu tại vị trí (i,j) mà i–j=k; ngược lại thì dcc[k]=0.
4. Tương tự ta dùng mảng dcp với ý nghĩa: dcp[k]=1 nếu đường chéo phụ thứ k đã có một con hậu được đặt.

Giả mã của thuật toán xếp hậu như sau:

```
procedure Try(i);
var j;
begin
   for j := 1 to n do
```

```
      if (cot[j]=0) and (dcc[i-j]=0) and (dcp[i+j]=0) then begin
         x[i] := j;
               cot[j]:=1; dcc[i-j]:=1; dcp[i+j]:=1;     {ghi nhận trạng thái mới}
         if i=n then Update
         else Try(i+1);
               cot[j]:=0; dcc[i-j]:=0; dcp[i+j]:=0;     {phục hồi trạng thái cũ}
      end;
end;
procedure Update;
begin
   count := count + 1;
   print(x);
end;
```

Phần dưới là toàn bộ chương trình tìm các phương án xếp hậu trên bàn cờ 8x8. Chương trình tìm được 92 phương án khác nhau.

e) Bài toán từ đẹp

Tất cả các bài toán ta đã giải ở trên đều có cấu hình có thành phần là các số nguyên. Riêng bài toán từ đẹp thì cần tìm cấu hình là một xâu. Ta có thể dùng một mảng kí tự để thay thế, tuy nhiên điều đó không cần thiết vì ngôn ngữ Pascal cũng có khả năng xử lí xâu kí tự rất tốt.

Mô hình quay lui của bài toán từ đẹp có thể viết như sau:

```
procedure Try(i)
var c;
begin
   for c := 'A' to 'C' do begin
      x := x + c;
      if Ok(i) then
         if i=n then Update
         else Try(i+1);
      delete(x,i,1);
   end;
end;
procedure Update;
```

```
begin
   count := count + 1;
   print(x);
end;
```

Các thủ tục Try, Update khá tương tự các bài toán khác. Riêng để viết hàm Ok kiểm tra lựa chọn hiện tại cho x[i] có phù hợp không, chúng ta phân tích sâu hơn như sau:

Trước hết ta thấy rằng khi lựa chọn đến x[i] thì xâu x[1..i-1] đã thoả mãn tính chất của từ đẹp. Như vậy nếu x[1..i] không thoả mãn tính chất của từ đẹp thì chỉ có khả năng là do kí tự thứ i mới được chọn không phù hợp. Vậy hàm Ok(i) chỉ cần kiểm tra các xâu con có chứa x[i] có giống một xâu con liền kề trước nó hay không? Nếu có thì giá trị x[i] đó không thoả mãn và ta phải chọn giá trị khác. Ngược lại nếu giá trị x[i] thoả mãn thì ta cập nhật kết quả hoặc đệ quy tiếp tuỳ thuộc vào việc ta đã chọn đủ n kí tự chưa.

Hàm Ok có thể viết như sau:

```
function Ok(l)
begin
        Ok := false;
        for k := 1 to l div 2 do
                if copy(x,l-k+1,k) = copy(x,l-2*k+1,k) then exit;
        Ok := true;
end;
```

Nếu độc giả thấy hàm Ok khó hiểu thì chúng tôi có thể giải thích như sau: ta cần kiểm tra mọi xâu con có chứa kí tự cuối cùng có bằng xâu con liền kề trước nó hay không? Độ dài xâu đang có là l, do đó các xâu con có chứa kí tự thứ l có khả năng bằng xâu liền kề trước nó chỉ có độ dài từ 1 đến l/2. Biểu thức copy(x,l-k+1,k) cho kết quả là xâu con gồm k kí tự cuối cùng của x và biểu thức copy(x,l-2*k+1,k) cho xâu con k kí tự ngay trước xâu con có k kí tự cuối cùng.

Phần cài đặt chương trình cụ thể xin dành cho độc giả. Phần tiếp theo chúng tôi xin đề cập đến bài toán tối ưu tổ hợp.

f) Bài toán người du lich.

Độc giả dễ dàng nhận thấy mỗi phương án của bài toán người du lịch là một hoán vị của n thành phố. Do đó ta có thể dùng mô hình vét cạn của bài toán sinh

hoán vị để tìm các phương án. Và ta sử dụng thêm ràng buộc: $d[xi_{-1},xi]<\infty$. Mặt khác vì phương án là một chu trình nên ta có thể coi thành phố xuất phát là thành phố 1.

Thuật giải bài toán người du lịch bằng vét cạn như sau:

```
procedure Search;
begin
   min := ∞;
   x[1] := 1; dd[1] := 1;
   try(2);
end;
procedure Try(i)
var j;
begin
   for j := 1 to n do
      if (dd[j]=0) and (d[x[i-1],j] < ∞) then begin
         x[i] := j; dd[j] := 1;
         if i=n then Update
         else Try(i+1);
         dd[j] := 0;
      end;
end;
procedure Update;
var s,i;
begin
   s := d[x[n],1];
   for i := 1 to n-1 do s := s + d[x[i],x[i+1]];
   if s < min then begin
      min := s;
      best := x;
   end;
end;
```

Lớp các bài toán tối ưu tổ hợp rất rộng. Phần lớn các bài toán đó trong trường hợp tổng quát chỉ có thuật toán tối ưu duy nhất là vét cạn. Tuy nhiên,

nhược điểm của phương pháp vét cạn là độ phức tạp tính toán rất lớn do hiện tượng bùng nổ tổ hợp. Các bạn nhớ lại rằng số hoán vị của tập n phần tử là n!. Do đó trong trường hợp xấu nhất thuật toán vét cạn đối với bài toán người du lịch là O(n!).

Có 2 giải pháp khắc phục vấn đề này. Giải pháp thứ nhất cải tiến phương pháp vét cạn bằng kỹ thuật nhánh cận, tức là loại bỏ ngay các hướng đi chắc chắn không dẫn đến phương án tối ưu. Giải pháp thứ 2 là sử dụng các phương pháp khác, mà hai phương pháp nổi bật nhất là phương pháp quy hoạch động và phương pháp tham lam.

Phần tiếp theo, chúng tôi sẽ trình bày sơ lược về kỹ thuật nhánh cận.

1.4. Kỹ thuật nhánh cận

Nguyên nhân dẫn đến độ phức tạp của các bài toán tối ưu tổ hợp là hiện tượng bùng nổ tổ hợp. Đó là hiện tượng số cấu hình tổ hợp tăng theo hàm mũ đối với số thành phần tổ hợp n. Đơn giản nhất là các dãy nhị phân, mỗi thành phần tổ hợp chỉ có 2 khả năng là 0 và 1 thì số các dãy nhị phân độ dài n đã là 2^n. Do đó việc sinh toàn bộ các cấu hình tổ hợp sẽ không khả thi khi n lớn.

Quá trình vét cạn kiểu quay lui là một quá trình tìm kiếm phân cấp, tức là các thành phần x_1, x_2… sẽ được chọn trước. Nếu tại bước i ta chọn một giá trị xi không tối ưu thì toàn bộ quá trình chọn xi_{+1}, xi_{+2}… sẽ hoàn toàn vô nghĩa. Ngược lại, nếu ta xác định được rằng giá trị xi đó không dẫn đến cấu hình tối ưu thì ta sẽ tiết kiệm được toàn bộ các bước chọn xi_{+1}, xi_{+2}… Tiết kiệm đó đôi khi là đáng kể. Chẳng hạn nếu đối với bài toán duyệt nhị phân (tối ưu các cấu hình là dãy nhị phân) ta xác định được x_1=0 không hợp lí thì ta đã tiết kiệm được 2^{n-1} bước duyệt phía sau. Đó chính là tư tưởng của phương pháp nhánh cận.

Mô hình quay lui có nhánh cận như sau:

```
Procedure Search;
begin
   Try(1);
end;
procedure Try(i);
var j;
Begin
   for j := 1 to m do
```

```
    if <chọn được a[j]> then begin
       x[i] := a[j];
       <ghi nhận trạng thái mới>;
       if i=n then Update
       else
                    if Ok(i) then Try(i+1);
       <trả lại trạng thái cũ>;
    end;
end;
```

Cải tiến so với phương pháp vét cạn thuần tuý là ở câu lệnh **if Ok(i) then Try(i+1);**. Hàm Ok ở đây được dùng để đánh giá tình trạng của cấu hình hiện tại. Thứ nhất là có đảm bảo dẫn đến cấu hình tối ưu hay không. Nếu không thì ít nhất cũng phải đảm bảo cho giá trị hàm mục tiêu tốt hơn phương án tốt nhất ta đang có.

Kĩ thuật nhánh cận rất đa dạng, phụ thuộc vào từng bài toán và tư duy của người lập trình. Chúng ta sẽ xem xét một số bài toán tối ưu giải bằng phương pháp nhánh cận.

Đầu tiên là bài toán người du lịch. Ta có nhận xét: tại lần di chuyển thứ i, nếu tổng chi phí đang có ≥ chi phí của phương án tốt nhất ta đang có thì rõ ràng việc đi tiếp không mang đến kết quả tốt hơn. Do đó ta có thể đặt một nhánh cận đơn giản như sau:

```
procedure Try(i)
var j;
begin
   for j := 1 to n do
      if (dd[j]=0) and (d[x[i-1],j] < ∞) then begin
         x[i] := j; dd[j] := 1; s := s + d[x[i-1],j];
         if i=n then Update
         else
            if s < min then Try(i+1);
         dd[j] := 0; s := s - d[x[i-1],j];
      end;
end;
```

Hai biến s, min là các biến toàn cục, trong đó min dùng để lưu chi phí của phương án tốt nhất còn s lưu tổng chi phí hiện tại.

Ta có thể tiếp tục cải tiến cận này bằng việc không chỉ xét chi phí đến thời điểm hiện tại mà còn xét luôn cả chi phí tối thiểu để kết thúc hành trình. Gọi dmin là giá trị nhỏ nhất của bảng d, tương đương với chi phí nhỏ nhất của việc di chuyển từ thành phố này đến thành phố kia. Tại bước thứ i thì ta còn phải thực hiện n–i+1 bước di chuyển nữa thì mới kết thúc hành trình (đi qua n–i thành phố còn lại và quay về thành phố 1). Do đó chi phí của cả hành trình sẽ tối thiểu là s + (n–i+1)*dmin. Nếu chi phí này còn lớn hơn chi phí của phương án tốt nhất thì rõ ràng lựa chọn hiện tại cũng không thể dẫn đến một phương án tốt hơn. Chương trình con vét cạn đệ quy có thể sửa thành:

```
procedure Try(i)
var j;
begin
   for j := 1 to n do
      if (dd[j]=0) and (d[x[i–1],j] < ∞) then begin
         x[i] := j; dd[j] := 1; s := s + d[x[i–1],j];
         if i=n then Update
         else
            if s + (n–i+1)*dmin < min then Try(i+1);
         dd[j] := 0; s := s – d[x[i–1],j];
      end;
end;
```

Nhìn chung những cận có cải thiện tình hình đôi chút nhưng cũng không thực sự hiệu quả. Người ta cũng đã nghiên cứu nhiều cận chặt hơn, độc giả có thể tìm đọc ở các tài liệu khác.

Ta xét tiếp bài toán từ đẹp nhất. Định nghĩa **từ đẹp** đã được mô tả ở bài toán từ đẹp. **Từ đẹp nhất** là từ có ít kí tự C nhất. Rõ ràng bài toán tìm từ đẹp nhất là một bài toán tối ưu tổ hợp.

Chúng ta xây dựng nhánh cận với nhận xét: nếu x[1..n] là từ đẹp thì trong 4 kí tự liên tiếp của x phải có ít nhất một kí tự C. Vậy, nếu ta đã xây dựng i kí tự thì phần còn lại gồm n–i kí tự sẽ có ít nhất (n–i)/4 kí tự C. Do đó số kí tự C tối thiểu

của cả xâu sẽ là t + (n–i)/4, trong đó t là số kí tự C của x[1..i]. Ta có thể dùng t+(n–i)/4 làm cận. Chương trình con đệ quy như sau:

```
procedure Try(i)
var c;
begin
   for c := 'A' to 'C' do begin
      x := x + c;
      if c = 'C' then t := t + 1;
      if Ok(i) then
         if i=n then Update
         else
            if t + (n-i) div 4 < minC then Try(i+1);
      delete(x,i,1);
      if c = 'C' then t := t - 1;
   end;
end;
```

Biến minC ở đây dùng để lưu số kí tự C của phương án tốt nhất đang có.

Nhánh cận là một kĩ thuật mạnh và đòi hỏi tư duy sâu sắc. Chọn được một cận tốt thường không đơn giản, đòi hỏi phải có những phân tích sâu sắc và tỉ mỉ. Một số chú ý khi chọn cận:

1. Cận phải đánh giá chính xác tình trạng cấu hình hiện tại. Nếu quá lỏng thì số cấu hình loại bỏ không đáng kể, nếu quá chặt thì sẽ dẫn đến bỏ sót nghiệm.
2. Cận phải tính toán đơn giản. Vì thao tác tính cận thực hiện tại tất cả các bước nên nếu tính toán cận quá phức tạp thì thời gian rút ngắn nhờ đặt cận tiết kiệm được thì lại mất đáng kể cho việc tính cận.

Mặc dù nhánh cận là kĩ thuật mạnh nhưng muốn để áp dụng tốt đòi hỏi những phân tích rất chi tiết. Hơn nữa nhiều trường hợp có đặt cận thì số phương án cần duyệt vẫn quá nhiều. Trong những trường hợp như vậy chúng ta cần phải có những cách tiếp cận khác. Phần tiếp theo trình bày về một phương pháp cực kì hiệu quả, đó là phương pháp quy hoạch động.

2. Phương pháp tham lam (Gready)

Tìm nghiệm của bài toán tối ưu thường đòi hỏi chi phí lớn về thời gian tính toán và không gian bộ nhớ. Tuy nhiên trong nhiều trường hợp ta chỉ tìm được một

nghiệm đủ tốt, khá gần với nghiệm tối ưu là đạt yêu cầu, nhất là khi có hạn chế về mặt thời gian và bộ nhớ.

Phương pháp tham lam xây dựng các thuật toán giải các bài toán tối ưu dựa trên tư tưởng tối ưu cục bộ theo một chiến lược tư duy kiểu con người, nhằm nhanh chóng đạt đến một lời giải "tốt".

Có một số thuật toán dựa trên tư tưởng của phương pháp tham lam thực sự tìm được phương án tối ưu (chẳng hạn thuật toán Kruscal tìm cây khung cực tiểu), còn lại đa số các thuật toán dựa trên phương pháp tham lam thường là thuật toán gần đúng, chỉ cho một lời giải xấp xỉ lời giải tối ưu.

2.1. Một số chiến lược tham lam

Phương pháp tham lam tìm nghiệm tối ưu dựa trên các chiến lược tối ưu cục bộ của con người. Trước khi trình bày những thuật giải tham lam cho những bài toán cụ thể, chúng tôi đề cập đến hai chiến lược tối ưu cục bộ cơ bản:

- Chọn cái tốt nhất trước (còn gọi là "chọn miếng ngon trước". Đây là lí do vì sao phương pháp này được gọi là "tham lam" hay "tham ăn").
- Cải tiến cái đang có thành cái tốt hơn.

Chiến lược thứ nhất thường được áp dụng khi xây dựng dần từng thành phần của nghiệm tối ưu. Thuật giải sẽ đánh giá các lựa chọn theo một tiêu chuẩn nào đó và sắp xếp từ nhỏ đến lớn rồi tiến hành chọn theo trình tự đó.

Chiến lược thứ hai thường bắt đầu bằng một hay một vài phương án. Sau đó, bằng một số cách thức nào đó, các phương án được điều chỉnh để có giá trị tốt hơn. Quá trình điều chỉnh sẽ dừng lại khi không điều chỉnh được thêm hoặc sự cải thiện rất nhỏ hoặc hết thời gian cho phép… Phần lớn các thuật toán hiện nay áp dụng cho bộ dữ liệu lớn được thiết kế theo chiến lược này: chẳng hạn tìm kiếm leo đồi, giải thuật di truyền… Độc giả có thể tham khảo trong những tài liệu khác.

Chúng ta sẽ đi vào một số bài toán cụ thể và vận dụng những chiến lược trên để thiết kế các thuật giải tham lam.

2.2. Bài toán xếp balô

Giải thuật tham lam giải bài toán xếp balô dựa trên chiến lược "chọn cái tốt nhất trước". Việc chọn các vật đưa vào balô có thể theo 3 chiến lược như sau:

- Ưu tiên vật nhẹ trước (với hi vọng chọn được nhiều đồ vật).
- Ưu tiên vật có giá trị cao trước.
- Ưu tiên chọn những vật có tỉ số giá trị/trọng lượng lớn trước.

Ta thấy chiến lược thứ 3 tổng quát hơn nhất. Do vậy việc tìm nghiệm của chúng ta sẽ tiến hành theo 2 bước:

1. Sắp xếp các đồ vật giảm dần theo tỉ lệ: giá trị/trọng lượng.
2. Lần lượt đưa vào balô những đồ vật nào có thể đưa được theo trình tự đã sắp xếp đó.

Thuật giải chi tiết như sau:

```
procedure Gready;
begin
   for i := 1 to n do begin a[i]:=v[i]/w[i];id[i] := i; end;
   sắp xếp w,v,a,id theo a;

   for i:=1 to n do begin
      if m >= w[i] then begin
         m := m - w[i];
         t := t + v[i];
         s := s + [id[i]];
      end;
   end;
end;
```

Kết quả: S là tập các đồ vật được chọn, T là tổng giá trị của chúng. Thuật giải này có độ phức tạp O(nlogn) do thao tác chủ yếu ở phần sắp xếp.

2.3. Bài toán người du lịch

Có nhiều giải thuật tham lam khác nhau giải bài toán này. Chúng tôi xin trình bày một giải thuật theo chiến lược chọn cái tốt trước và một giải thuật theo chiến lược cải tiến cái hiện có.

Giải thuật theo chiến lược chọn cái tốt nhất trước có ý tưởng rất đơn giản: tại mỗi bước ta sẽ chọn thành phố tiếp theo là thành phố chưa đến thăm mà chi phí từ thành phố hiện tại đến thành phố đó là thấp nhất.

Giải thuật theo chiến lược cải tiến cái hiện có cũng có ý tưởng rất đơn giản: xuất phát từ một lộ trình nào đó (1,2,..n chẳng hạn), ta cải tiến lộ trình hiện có bằng cách tìm 2 thành phố mà đổi chỗ chúng cho nhau thì tổng chi phí giảm đi. Quá trình đó dừng lại khi không còn cải tiến được hơn nữa.

Dựa trên 2 ý tưởng đó, bạn đọc có thể dễ dàng xây dựng được các thuật giải. Thuật giải thứ nhất có độ phức tạp tính toán là $O(n^2)$, thuật giải thứ hai có độ phức tạp tính toán là $O(n^3)$.

Ngoài 2 giải thuật trên, người ta còn xây dựng được giải thuật di truyền cho bài toán này. ý tưởng cơ bản là thay vì xuất phát từ một phương án, chúng ta xử lí nhiều phương án đồng thời, cải tiến chúng bằng việc mô phỏng quá trình di truyền, thích nghi và tiến hoá của sinh vật để có được những phương án ngày một tốt hơn. Tuy nhiên vấn đề đó nằm ngoài khuôn khổ của cuốn giáo trình này.

Đối với đồ thị metric (tức là có bất đẳng thức tam giác d[i,j]<=d[i,k]+d[k,j] với mọi i,j,k) người ta còn tìm được thuật giải tham lam cho nghiệm có tổng chi phí <=2 tổng chi phí của nghiệm tối ưu. Thuật giải tiến hành cũng theo 2 bước:

1. Tìm cây khung cực tiểu của đồ thị.
2. Duyệt cây khung theo chiều sâu.
3. Xây dựng lộ trình là thứ tự duyệt của các đỉnh, loại bỏ các đỉnh trùng nhau.

2.4. Bài toán đóng thùng (Bin-Packing)

Cho N đồ vật, mỗi đồ vật có trọng lượng là ai. Có rất nhiều thùng giống nhau và mỗi thùng chỉ có khả năng chứa được các đồ vật có tổng khối lượng không quá M (tất nhiên ai<=M) . Hãy xếp các đồ vật vào các thùng sao cho số thùng sử dụng là ít nhất.

Bài toán đóng thùng cũng là một bài toán NPC, tức là chưa có thuật giải đa thức. Thuật giải tham lam của bài toán này dựa trên ý tưởng:

1. Sắp xếp các đồ vật theo thứ tự trọng lượng giảm dần.
2. Lần lượt đưa các đồ vật vào các thùng bằng cách tìm thùng đầu tiên vẫn còn đủ để chứa vật ấy. Nếu không có thì thêm một thùng mới để chứa vật ấy.

```
procedure Gready;
begin
   for i:=1 to n do id[i] := i;;
   sxep a,id on a;
   k:=0;
   for i:=1 to n do begin
      for j := 1 to k+1 do
        if t[j]+a[i]<=M then break; {tìm thùng j đầu tiên}
```

```
        if j=k+1 then k:=k+1;       {k thùng đầu tiên không thể  chứa được => dùng thùng mới}
        t[j] := t[j] + a[i];
        s[j] := s[j] + [id[i]];
    end;
end;
```

ý nghĩa của các cấu trúc dữ liệu như sau: id[i] là chỉ số ban đầu đồ vật i sau khi sắp xếp, k là số thùng cần dùng, t[j] là tổng trọng lượng các vật đang có trong thùng j, s[j] là tập hợp các vật được đưa vào thùng j.

Độ phức tạp tính toán của thuật toán là $O(n^2)$.

Johnson đã chứng minh được thuật giải này cho nghiệm x thoả mãn:

$f(x)$ xấp xỉ $11*f(x_0)/9+4$ với x_0 là nghiệm tối ưu.

Trước khi kết thúc trình bày về phương pháp tham lam, chúng tôi đề cập đến thuật toán Kruscal, một thuật toán tham lam thực sự tối ưu.

2.5. Thuật toán Kruscal

Thuật toán Kruscal giải bài toán tìm cây khung cực tiểu của đồ thị vô hướng có trọng số. Bài toán cây khung cực tiểu đã được trình bày ở chương Đồ thị. Nói một cách đơn giản, cây khung cực tiểu của một đồ thị N đỉnh là một đồ thị con N đỉnh, N-1 cạnh, liên thông và có tổng trọng số các cạnh là nhỏ nhất.

Lý thuyết đồ thị đã chứng minh một đồ thị liên thông không có chu trình sẽ là một cây.

Tư tưởng tham lam trong thuật toán Kruscal là "chọn cái tốt nhất trước". Chúng ta sẽ tiến hành chọn N-1 cạnh ưu tiên các cạnh có trọng số nhỏ trước sao cho khi chọn cạnh được chọn phải không tạo thành chu trình với các cạnh đã chọn.

Như vậy thuật toán sẽ tiến hành qua 2 bước: sắp xếp và chọn. Để kiểm tra một cạnh khi được chọn có tạo thành chu trình hay không, ta sẽ lưu trữ các đỉnh vào các cây con. Nếu hai đỉnh đầu mút của một cạnh mà cùng thuộc một cây thì thêm cạnh đó sẽ tạo thành chu trình. Đồng thời, khi thêm một cạnh ta cũng hợp nhất 2 cây của 2 đỉnh tương ứng.

Về cấu trúc dữ liệu: ta biểu diễn đồ thị bằng danh sách cạnh, gồm các bộ ba (u,v,d) với ý nghĩa trọng số cạnh (u,v) là d. Để biểu diễn các cây con ta dùng mảng T, trong đó T[u] là đỉnh cha của đỉnh u trong cây. Nếu T[u] bằng 0 thì u là đỉnh gốc. Hai đỉnh cùng thuộc một cây nếu chúng cùng đỉnh gốc.

Thuật toán chi tiết như sau:

```
procedure Kruscal;
begin
        sắp xếp u,v,d tăng dần theo d; {sắp xếp danh sách cạnh tăng dần}
        for i := 1 to m do begin
                x := Root(u[i]); y := Root(v[i]);        {tìm gốc của mỗi đỉnh }
                if x <> y then begin                {thuộc 2 cây khác nhau  }
                        k := k + 1;
                        s := s + [i];           {chọn cạnh i}
                T[y] := x;                      {hợp nhất 2 cây}
                        if k=n-1 then exit; {chọn đủ n-1 cạnh thì xong}
                end;
        end;
end;
function Root(u);           {tìm gốc của đỉnh u, là đỉnh có T = 0}
begin
        while T[u] <> 0 do u := T[u];
        Root := u;
end;
```

Kết quả ta được s là danh sách các cạnh được chọn. Dễ dàng chứng minh được độ phức tạp của thuật toán là O(mlogm), chủ yếu là ở thời gian sắp xếp các cạnh.

Qua các thuật giải tham lam đã trình bài, chúng ta có thể kết luận:

1. Phương pháp tham lam có độ phức tạp tính toán thấp, thường nhanh chóng tìm được lời giải .
2. Lời giải của phương pháp tham lam thường chỉ là một lời giải tốt chứ không phải lời giải tối ưu.

3. Kết luận

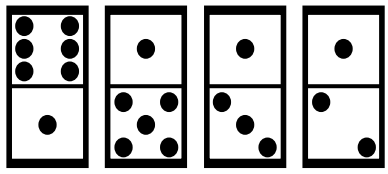
Trong chương này chúng ta đã tìm hiểu về hai phương pháp thiết kế thuật toán phổ biến: vét cạn, tham lam. Mỗi phương pháp có những ưu điểm và nhược điểm riêng. Phương pháp vét cạn có ưu điểm là đơn giản và chắc chắn tìm được lời giải tối ưu. Bù lại nhược điểm của nó là độ phức tạp quá lớn. Phương pháp

tham lam thì nhanh chóng, đơn giản và không đòi hỏi nhiều bộ nhớ, nhưng lại không chắc chắn tìm được lời giải tối ưu.

Do mỗi phương pháp đều có ưu, nhược điểm riêng nên tuỳ vào yêu cầu của bài toán và khả năng cho phép mà ta có thể lựa chọn hay phối hợp các phương pháp khác nhau để tìm kiếm lời giải tốt nhất. Chẳng hạn nếu bộ nhớ hạn hẹp thì ta có thể dùng vét cạn với bộ dữ liệu nhỏ và tham lam với bộ dữ liệu lớn, hoặc dùng phương pháp tham lam để xác định một số cận cho vét cạn.

Bài tập

Bài 1: Dominoes



Có N quân Domino xếp thành một hàng như hình vẽ

Mỗi quân Domino được chia làm hai phần, phần trên và phần dưới. Trên mặt mỗi phần có từ 1 đến 6 dấu chấm.

Ta nhận thấy rằng:

Tổng số dấu chấm ở phần trên của N quân Domino bằng: 6+1+1+1=9, tổng số dấu chấm ở phần dưới của N quân Domino bằng 1+5+3+2=11, độ chênh lệch giữa tổng trên và tổng dưới bằng |9-11|=2

Với mỗi quân, bạn có thể quay 180^0 để phần trên trở thành phần dưới, phần dưới trở thành phần trên, và khi đó độ chênh lệch có thể được thay đổi. Ví dụ như ta quay quân Domino cuối cùng của hình trên thì độ chênh lệch bằng 0

Bài toán đặt ra là: Cần quay ít nhất bao nhiêu quân Domino nhất để độ chênh lệch giữa phần trên và phần dưới là nhỏ nhất.

Dữ liệu vào trong file: "Dom.in" có dạng:

- Dòng đầu là số nguyên dương N ($1 \le N \le 20$)
- N dòng sau, mỗi dòng hai số a_i, b_i là số dấu chấm ở phần trên, số dấu chấm ở phần dưới của quân Domino thứ i ($1 \le a_i, b_i \le 6$)

Kết quả ra file: "Dom.out" có dạng:

- Gồm 1 dòng duy nhất chứa 2 số nguyên cách nhau một dấu cách là độ chênh lệch nhỏ nhất và số quân Domino cần quay ít nhất để được độ chênh lệch đó.

Bài 2: Tham quan

Trong đợt tổ chức đi tham quan danh lam thắng cảnh của thành phố Hồ Chí Minh, Ban tổ chức hội thi Tin học trẻ tổ chức cho N đoàn (đánh từ số 1 đến N) mỗi đoàn đi thăm quan một địa điểm khác nhau. Đoàn thứ i đi thăm địa điểm ở cách Khách sạn Hoàng Đế di km (i=1,2,...., N). Hội thi có M xe taxi đánh số từ 1 đến M ($M \ge N$) để phục vụ việc đưa các đoàn đi thăm quan. Xe thứ j có mức tiêu thụ xăng là vj đơn vị thể tích/km.

Yêu cầu: Hãy chọn N xe để phục vụ việc đưa các đoàn đi thăm quan, mỗi xe chỉ phục vụ một đoàn, sao cho tổng chi phí xăng cần sử dụng là ít nhất.

Dữ liệu: File văn bản TQ.INP:

- Dòng đầu tiên chứa hai số nguyên dương N, M (N≤M≤200);
- Dòng thứ hai chứa các số nguyên dương d_1, d_2, ..., dN;
- Dòng thứ ba chứa các số nguyên dương v_1, v_2, ..., vM.
- Các số trên cùng một dòng được ghi khác nhau bởi dấu trắng.

Kết quả: Ghi ra file văn bản TQ.OUT:

- Dòng đầu tiên chứa tổng lượng xăng dầu cần dùng cho việc đưa các đoàn đi thăm quan (không tính lượt về);
- Dòng thứ i trong số N dòng tiếp theo ghi chỉ số xe phục vụ đoàn i (i=1, 2, ..., N).

Bài 3: Biểu thức

Cho một xâu S (chỉ gồm các ký tự ‘0’ đến ‘9’, độ dài nhỏ hơn 10) hãy tìm cách chèn vào S các dấu '+' hoặc '-' để thu được số M cho trước (nếu có thể). Xâu S và số M (M thuộc kiểu longint) nguyên được đọc từ file “BT.INP”. Trong file BT.OUT ghi tất cả các phương án chèn (nếu có) và ghi "-1" nếu như không thể thu được M từ cách làm trên.

Ví dụ: M = 8, S=’123456789’ một trong các phương án đó là: '-1+2-3+4+5-6+7';