Nhánh và cận

1. Phương pháp

Trong thực tế, có nhiều bài toán yêu cầu tìm ra một phương án thoả mãn một số điều kiện nào đó, và phương án đó là tốt nhất theo một tiêu chí cụ thể. Các bài toán như vậy được gọi là bài toán tối ưu. Có nhiều bài toán tối ưu không có thuật toán nào thực sự hữu hiệu để giải quyết, mà cho đến nay vẫn phải dựa trên mô hình xem xét toàn bộ các phương án, rồi đánh giá để chọn ra phương án tốt nhất.

Phương pháp nhánh và cận là một dạng cải tiến của phương pháp quay lui, được áp dụng để tìm nghiệm của bài toán tối ưu.

Giả sử nghiệm của bài toán có thể biểu diễn dưới dạng một vector $(x_1, x_2,., x_n)$, mỗi thành phần x_i (i=1, 2,….,n) được chọn ra từ tập S_i . Mỗi nghiệm của bài toán

$X= (x_1, x_2,., x_n)$, được xác định "độ tốt" bằng một hàm f(X) và mục tiêu cần tìm nghiệm có giá trị f(X) đạt giá trị nhỏ nhất (hoặc đạt giá trị lớn nhất).

Tư tưởng của phương pháp nhánh và cận như sau: Giả sử, đã xây dựng được k thành phần $(x_1, x_2,., x_k)$ của nghiệm và khi mở rộng nghiệm $(x_1, x_2,., x_{k+1})$, nếu biết rằng tất cả các nghiệm mở rộng của nó $(x_1, x_2,., x_{k+1},…)$ đều không tốt bằng nghiệm tốt nhất đã biết ở thời điểm đó, thì ta không cần mở rộng từ $(x_1, x_2,., x_k)$ nữa. Như vậy, với phương pháp nhánh và cận, ta không phải duyệt

toàn bộ các phương án để tìm ra nghiệm tốt nhất mà bằng cách đánh giá các nghiệm mở rộng, ta có thể cắt bỏ đi những phương án (nhánh) không cần thiết, do đó việc tìm nghiệm tối ưu sẽ nhanh hơn. Cái khó nhất trong việc áp dụng phương pháp nhánh và cận là đánh giá được các nghiệm mở rộng, nếu đánh giá được tốt sẽ giúp bỏ qua được nhiều phương án không cần thiết, khi đó thuật toán nhánh cận sẽ chạy nhanh hơn nhiều so với thuật toán vét cạn.

Thuật toán nhánh cận có thể mô tả bằng mô hình đệ quy sau:

```
procedure BranchBound(i);// xây dựng thành phần thứ i begin
  <Đánh giá các nghiệm mở rộng>;
  if(các nghiệm mở rộng đều không tốt hơn
BestSolution)then exit;
  <Xác định Si>;
  for xi $ Si do begin
<ghi nhận thành phần thứ i>;
if (tìm thấy nghiệm) then <Cập nhật BestSolution> else
BranchBound(i+1);
<loại thành phần i>; end;
end;
```

Trong thủ tục trên, `BestSolution` là nghiệm tốt nhất đã biết ở thời điểm đó. Thủ tục `<cập nhật BestSolution>` sẽ xác định "độ tốt" của nghiệm mới tìm thấy, nếu nghiệm mới tìm thấy tốt hơn `BestSolution` thì `BestSolution` sẽ được cập nhật lại là nghiệm mới tìm được.

2. Giải bài toán người du lịch bằng phương pháp nhánh cận.

Bài toán. Cho n thành phố đánh số từ 1 đến n và các tuyến đường giao thông hai chiều giữa chúng, mạng lưới giao thông này được cho bởi mảng C[1..n,1..n], ở đây $C_{ij} = C_{ji}$ là chi phí đi đoạn đường trực tiếp từ thành phố i đến thành phố j.

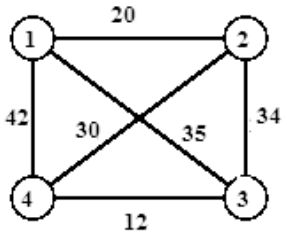
Một người du lịch xuất phát từ thành phố 1, muốn đi thăm tất cả các thành phố còn lại mỗi thành phố đúng 1 lần và cuối cùng quay lại thành phố 1. Hãy chỉ ra cho người đó hành trình với chi phí ít nhất. Bài toán được gọi là bài toán người du lịch hay bài toán người chào hàng (Travelling Salesman Problem - TSP)

Dữ liệu vào trong file "TSP.INP" có dạng:

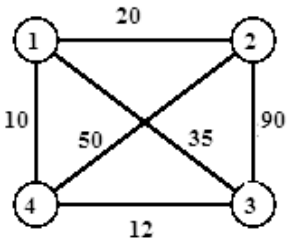
- Dòng đầu chứa số n(1<n≤20), là số thành phố. - n dòng tiếp theo, mỗi dòng n số mô tả mảng C

Kết quả ra file "TSP.OUT" có dạng: - Dòng đầu là chi phí ít nhất

- Dòng thứ hai mô tả hành trình Ví dụ 1:

TSP.INP	TSP.OUT	Hình minh họa
4 0 20 35 42 20 0 34 30 35 34 0 12 42 30 12 0	97 1->2->4->3->1	

Ví dụ 2:

TSP.INP	TSP.OUT	Hình minh họa
4 0 20 35 10 20 0 90 50 35 90 0 12 10 50 12 0	117 1->2->4->3->1	

Giải

1) Hành trình cần tìm có dạng ($x_1 = 1$, x_2, ..., x_n, $x_{n+1} = 1$), ở đây giữa x_i và x_{i+1}: hai thành phố liên tiếp trong hành trình phải có đường đi trực tiếp; trừ thành phố 1, không thành phố nào được lặp lại hai lần, có nghĩa là dãy (x_1, x_2, ..., x_n) lập thành một hoán vị của (1, 2, ..., n).
2) Duyệt quay lui: x_2 có thể chọn một trong các thành phố mà x_1 có đường đi trực tiếp tới, với mỗi cách thử chọn x_2 như vậy thì x_3 có thể chọn một trong các thành phố mà x_2 có đường đi tới (ngoài x_1). Tổng quát: x_i có thể chọn 1 trong các thành phố chưa đi qua mà từ x_{i-1} có đường đi trực tiếp tới.($2 \le i \le n$).

3) Nhánh cận: Khởi tạo cấu hình BestSolution có chi phí = +∞. Với mỗi bước thử chọn x_i xem chi phí đường đi cho tới lúc đó có nhỏ hơn chi phí của cấu hình BestSolution không? nếu không nhỏ hơn thì thử giá trị khác ngay bởi có đi tiếp cũng chỉ tốn thêm. Khi thử được một giá trị x_n ta kiểm tra xem x_n có đường đi trực tiếp về 1 không ? Nếu có đánh giá chi phí đi từ thành phố 1 đến thành phố x_n cộng với chi phí từ x_n đi trực tiếp về 1, nếu nhỏ hơn chi phí của đường đi BestSolution thì cập nhật lại BestSolution bằng cách đi mới.

```
program TSP;
const      MAX              =20;
           oo               =1000000;  fi    ='TSP.INP';
           fo               ='TSP.OUT';
var        c           :array[1..MAX,1..MAX]of longint;
           x,bestSolution  :array[1..MAX]of longint;
           d               :array[1..MAX]of longint; n
                           :longint;
           sum,best        :longint; procedure input;
var f      :text; i,j,k  :longint;
 begin
    assign(f,fi); reset(f); read(f,n);
    for i:=1 to n do
      for j:=1 to n do read(f,C[i,j]); close(f);
```

```
 end;
procedure update; begin
   if sum+C[x[n],x[1]]<best then begin
       best:=sum+C[x[n],x[1]]; bestSolution:=x;
   end; end;
procedure branchBound(i:longint); var j   :longint;
 begin
    if sum>=best then exit; for j:=1 to n do
      if d[j]=0 then begin x[i]:=j;  d[j]:=1;
         sum:=sum + C[x[i-1],j]; if i=n then update
          else branchBound(i+1); sum:=sum - C[x[i-1],j];
         d[j]:=0;
      end; end;
procedure init; begin
    fillchar(d,sizeof(d),0); d[1]:=1;
```

```
    x[1]:=1; best:=oo;
 end;
procedure output; var f   :text;
    i   :longint; begin
    assign(f,fo); rewrite(f); writeln(f,best);
    for i:=1 to n do write(f,bestSolution[i],'->');
    write(f,bestSolution[1]);
    close(f);
end;
BEGIN
  input;
  init;
  branchBound(2);
  output;
END.
```

Chương trình trên là một giải pháp nhánh cận rất thô sơ giải bài toán TSP, có thể có nhiều cách đánh giá nhánh cận chặt hơn nữa làm tăng hiệu quả của chương trình.

3. Bài toán máy rút tiền tự động ATM

Bài toán

Một máy ATM hiện có n ($n \leq 20$) tờ tiền có giá $t_1, t_2, ., t_n$. Hãy đưa ra cách trả ít tờ nhất với số tiền đúng bằng S.

Dữ liệu vào từ file ATM.INP có dạng:

- Dòng đầu là 2 số n và S.
- Dòng thứ 2 gồm n số $t_1, t_2, ., t_n$.

Kết quả ra file ATM.OUT có dạng: Nếu có thể trả tiền đúng bắng S thì đưa ra số tờ ít nhất cần trả và đưa ra cách trả, nếu không ghi -1.

ATM.INP	ATM.OUT

10 390 200 10 20 20 50 50 50 50 100 100	5 20 20 50 100 200

Giải

Như ta đã biết, nghiệm của bài toán là một dãy nhị phân độ dài n, giả sử đã xây dựng được k thành phần (x_1, x_2, ..., x_k), đã trả được sum và sử dụng c tờ. Để đánh giá được các nghiệm mở rộng của (x_1, x_2, ..., x_k), ta nhận thấy:

- Còn phải trả S-sum;
- Gọi tmax[k] là giá cao nhất trong các tờ tiền còn lại
  (tmax[k]=MAX{ t_{k+1}, ….., t_n})

thì ít nhất cần sử dụng thêm $\left\lceil \frac{S-sum}{t\max[k]} \right\rceil$ tờ nữa.

Do đó, nếu c+ $\frac{S-sum}{t\max[k]}$ mà lớn hơn hoặc bằng số tờ của cách trả tốt nhất hiện có

thì không cần mở rộng các nghiệm của (x_1, x_2, ..., x_k) nữa.

```
 const       MAX         =20;
             fi          ='ATM.INP'; fo     ='ATM.OUT';
 type        vector      =array[1..MAX]of   longint;   var
             t,tmax      :array[1..MAX]of longint;
             x,xbest     :vector; c,cbest   :longint;
             n,s,sum     :longint;
 procedure input; var   f   :text;
       i   :longint; begin
    assign(f,fi); reset(f); readln(f,n, s);
    for i:=1 to n do read(f,t[i]); close(f);
  end;  procedure init; var i :longint;
begin tmax[n]:=t[n];
    for i:=n-1 downto 1 do begin tmax[i]:=tmax[i+1];
         if tmax[i]<t[i] then tmax[i]:=t[i]; end;
    sum:=0; c:=0; cbest:=n+1;
  end;
 procedure update; var i   :longint;
     f   :text; begin
    if (sum = s) and (c<cbest) then begin xbest:=x;
       cbest:=c; end;
  end;
procedure printResult;
 var i   :longint; f   :text;
  begin
    assign(f,fo); rewrite(f); if cbest<n+1 then begin
      writeln(f,cbest); for i:=1 to n do
        if xbest[i]=1 then write(f,t[i],' '); end
    else write(f,'-1'); close(f);
```

```
  end;
 procedure branchBound(i:longint); var j :longint;
  begin
    if c + (s-sum)/tmax[i] >= cbest then exit; for j:=0
    to 1 do begin
       x[i]:=j;
       sum:=sum + x[i]*t[i]; c:=c + j;
       if (i=n) then update
       else if sum<=s then branchBound(i+1); sum:=sum -
       x[i]*t[i];
       c:=c - j; end;
end;
BEGIN
   input; init;
   branchBound(1); PrintResult;
END.
```