Contents

Chương 3 : Liệt kê

Câu :Trình bày phương pháp liệt kê nhị phân bằng phương pháp sinh

- Xâu $X=(x_1,x_2,...,x_n)$: $x_i=0,1; i = 1,2,…,n$ được gọi là xâu nhị phân có độ dài n. Ví dụ với $n = 4$, ta có 16 xâu nhị phân

- cấu hình đầu tiên sẽ là (00…0) và cấu hình cuối cùng là (11…1).

- nếu cấu hình x[1..n] là cấu hình đang có và không phải cấu hình cuối cùng cần liệt kê thì xâu kế tiếp sẽ nhận được bằng cách cộng thêm 1 (theo cơ số 2 có nhớ) vào cấu hình hiện tại.

***Phương pháp :**

-Giả sử cấu hình hiện tại $X=(x_{1,2},...,x_n)$:

-Nếu $x_i=1$ với mọi i, thì x là cấu hình cuối cùng, thuật toán liệt kê kết thúc

-Gọi x_k là chữ số 0 đầu tiên tính từ bên phải của x, như vậy $x = x_1x_2...x_{k-1}011...1$

-Cấu hình tiếp theo $y=y_1y_2...y_n$ được tạo ra như sau:

$y_i=x_i$ với $1\le i\le k-1$
$y_i=1-x_i$ ới $k\le i\le n$
$=> y=x_1x_2\ldots x_{k-1}100\ldots0$

//sinh nhị phân

```
#include <bits/stdc++.h>
using namespace std;
int main(){
      int n; cin>>n;
      int x[n+1];
      memset(x,0,sizeof(x));
          while(1){
        for(int i=1;i<=n;i++) cout<<x[i]; cout<<' ';
        int i=n;
        while(x[i]==1){ x[i]=1-x[i];i--;}
        if(i==0) break;
        else x[i]=1-x[i];
    }
    return 0;
}
```

Câu :Trình bày phương pháp liệt kê tổ hợp bằng phương pháp sinh

- Mỗi tổ hợp chập k của 1, 2, … , n là một tập con k phần tử khác nhau của 1, 2, … , n. mỗi tổ hợp có dạng $X=(x_1,x_2,\ldots,x_n)$
- Tập con (cấu hình) đầu tiên là $X=(1,2,\ldots,k)$, tập con (cấu hình) cuối cùng là $(n-k+1,\ldots,n)$.
- Ta gọi tập con $X=(x_1,x_2,\ldots,x_k)$ là đứng trước tập con $Y=(y_1,y_2,\ldots,y_k)$ nếu tìm được chỉ số t sao cho: $x_1=y_1$, $x_2=y_2,\ldots,x_{t-1}=y_{t-1}, x_t<y_t$.

***Phương pháp :**

-Giả sử cấu hình tại X= $x=x_1x_2\ldots x_n$
-Nếu X là cấu hình cuối cùng =>thuật toán kết thúc
- Tìm từ phải sang trái của X phần tử $x_i \neq n-k+i$
- Thay x_i bởi x_i+1,
Thay x_j bởi x_i+j-i, với $j=i+1,i+2,\ldots,k$

//sinh tổ hợp

```
#include <bits/stdc++.h>
using namespace std;
int main() {
```

```
    int n,k;cin>>n>>k;
    int x[k+1];
    for(int i=1;i<=k;i++) x[i]=i;
    while(1){
        for(int i=1;i<=k;i++) cout<<x[i]; cout<<' ';
        int i=k;
        while(x[i]==n-k+i&&i>0) i--;
        if(i==0) break;
        else{
            x[i]++;
            for(int j=i+1;j<=k;j++) x[j]=x[i]-i+j;;
        }
    }
    return 0;
}
```

Câu :Trình bày pp liệt kê hoán vị bằng phương pháp sinh

- Số các hoán vị là $n!$. mỗi hoán vị có dạng $X=(x_1,x_2,...,x_n)$
- Cấu hình đầu tiên là $(1,2, \dots , n)$Cấu hình cuối cùng là $(n, n-1, \dots , 1)$

-Hoánvị$X=(x_1,x_2,...,x_n)$ được gọi là đứng trước hoán vị $Y=(y_1,y_2,...,y_n)$nếu tồn tại chỉ số k sao cho $x_1=y_1, x_2=y_2,\dots,x_{k-1}=y_{k-1}, x_k<y_k$.

***Phương pháp :**

-Giả sử cấu hình tại $X=(x_1,x_2,\dots,x_n)$ Nếu X là cấu hình cuối cùng =>thuật toán kết thúc

- Tìm từ phải sang trái hoán vị có chỉ số j đầu tiên thỏa mãn: $x_j<x_{j+1}$
- Tìmx_k là số nhỏ nhất mà còn lớn hơn x_j trong các số bênphải x_j
- Đổi chỗ x_j và x_k
- Lật ngược đoạn từx_{j+1}đến x_n

//sinh hoán vị

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;cin>>n;
    int x[n+1];
    for(int i=1;i<=n;i++) x[i]=i;
    while(1){
        for(int i=1;i<=n;i++) cout<<x[i]; cout<<' ';
        int j=n-1,k=n;
        while(x[j]>x[j+1]&&j>0) j--;
        if(j==0) break;
```

```
            else {
                while(x[k]<x[j]) k--;
                swap(x[j],x[k]);
                int l=j+1,r=n;
                while(l<r){swap(x[l],x[r]);l++;r--;}
            }
        }
        return 0;
}
```

Câu :Trình bày phương pháp liệt kê nhị phân bằng quay lui

Giả sử ta cần xác định bộ $X=(x_{1,2},\ldots,x_n)$biểu diễn các xâu nhị phân.Ứng với mỗi thành phần x_i ta có 2 khả năng cần lựa chọn là 0 và 1. Các giá trị này mặc nhiên được chấp nhận mà không cần phải thoả mãn điều kiện gì.

Ứng với mỗi khả năng j trong 2 khả năng dành cho thành phần x_i ta thực hiện:

chấp thuận từng khả năng j cho thành phần x_i

nếu i là thành phần cuối cùng ($i == n$) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ $i + 1$.

***Thuật toán:**

```
void Try ( int i ) {
        for (int j =0;  j<=1;j++){
                X[i] = j;
                if ( i ==n) Result();
                elseTry  (i+1);
        }
}
```

Khi đó, để duyệt các xâu nhị phân có độ dài n ta chỉ cần gọi đến thủ tục Try(1).

//**quay lui nhị phân**

```
#include <bits/stdc++.h>
using namespace std;
int x[100],n;
void out(){
        for(int i=1;i<=n;i++) cout<<x[i];
        cout<<" ";
}
void Try(int i){
        for(int j=0;j<=1;j++){
                x[i]=j;
                if(i==n) out();
                else Try(i+1);
        }
```

```
}
int main() {
      cin>>n;
      Try(1);
      return 0;
}
```

Câu :Trình bày phương pháp liệt kê tổ hợp bằng quay lui

Giả sử ta cần xác định bộ $X=(x_1,x_2,\dots,x_n)$ biểu diễn các tổ hợp gồm k phần tử.Ứng với mỗi thành phần x_i ta có các khả năng cần lựa chọn từ x_{i-1} + 1 cho đến n - k + i. Cần thêm vào x_0 = 0. Các giá trị đề cử này mặc nhiên được chấp nhận mà không cần phải thêm điều kiện gì.

Ứng với mỗi khả năng *j* trong các khả năng dành cho thành phần x_i ta thực hiện:

chấp thuận từng khả năng j cho thành phần x_i

nếu *i* là thành phần cuối cùng (*i* == k) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ *i* + 1.

***Thuật toán:**

```
void Try ( int i ){
          for (int j =X[i-1]+1; j<=n-k+ i; j++){
                    X[i] = j;
                    if(i==k) Result();
                    else Try (i+1);
          }
}
```

Khi đó, để duyệt các tập con *k* phần tử của 1, 2, … , *n* ta chỉ cần gọi đến thủ tục Try(1).

//quay lui tổ hợp

```
#include <bits/stdc++.h>
using namespace std;
int x[100],n,k;
void out(){
          for(int i=1;i<=k;i++) cout<<x[i]; cout<<" ";
}
void Try(int i){
          for(int j=x[i-1]+1;j<=n-k+i;j++){
                    x[i]=j;
                    if(i==k) out();
                    else Try(i+1);
          }
}
int main() {
      cin>>n>>k;
```

```
    Try(1);
return 0;
}
```

Câu :Trình bày phương pháp liệt kê hoán vị bằng quay lui

-Giả sử ta cần xác định bộ X=(x_{12}..,x_n) biểu diễn các hoán vị gồm n phần tử

Mỗi hoán vị X=(x_{12}..,x_n) là bộ có tính đến thứ tự của1,2, … ,n. Mỗi $x_i \in X$ có n lựa chọn. Khi $x_i = j$ được lựa chọn thì giá trị này sẽkhông được chấp thuận cho các thành phần còn lại.

-Để ghi nhận điều này, ta sử dụng mảng $unus$[] gồm n phần tử.

Nếu $unus[i] = True$ điều đó có nghĩa giá trị i được chấp thuận .$unused[i]$ = $False$tương ứng với giá trị i không được phép sử dụng-

-Ứng với mỗi khả năng j trong các khả năng dành cho thành phần x_i ta thực hiện: Nếu khả năng j được chấp thuận thì nếu i là thành phần cuối cùng ($i = n$) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ $i + 1$. Nếu không có khả năng j nào được chấp thuận cho thànhphần x_i thì ta quay lại bước trước đó ($i - 1$) để thử lại các khả năng tiếp theo của ($i - 1$).

***Thuật Toán:**

```
void Try ( int i ){
        for (int j =1; j<=n; j++){
                if (unused[j] ) {
                X[i] = j; unused[j] = false;
                if ( i ==n)Result();
                else Try (i+1);
                unused[j] = true;}}}
```

Khi đó, để duyệt cáchoán vị của 1, 2, …, n ta chỉ cần gọi đến thủ tục Try(1).

//**quay lui hoán vị**

```
#include<bits/stdc++.h>
using namespace std;
int n,x[100]; bool check[100];
void out(){
        for(int i=1;i<=n;i++)cout<<x[i]; cout<<" ";}
void Try(int i){
        for(int j=1;j<=n;j++){
                if(check[j]==0){
                        check[j]=1;
                        x[i]=j;
                        if(i==n) out();
                        else Try(i+1);
                        check[j]=0;
                }
        }
```

```
}
int main(){
      cin>>n;Try(1);
          return 0;
}
```

Chương 4:Bài toán tối ưu:

Câu :Thuật toán duyệt toàn bộ giải bài toán tối ưu:

***Bài toán tối ưu:** Cho tập dữ liệu D = {X = (x_1, x_2, …, x_n)}

Yêu cầu: Tìm x ∈ D sao cho f(x) → Min(Max);

- Hàm f(x) được gọi là hàm mục tiêu của bài toán.
- Mỗi phần tử x∈D gọi là một phương án của bài toán.
- Phương án x*∈D phương án tối ưu của bài toán (làm cho hàm mục tiêu có giá trị nhỏ nhất (lớn nhất)
- f* = f(x*) được gọi là giá trị tối ưu của bài toán.

*** Thuật Toán:**

Giả sử D là tập phương án. Xét mọi X∈D, tìm X* để f(X*)→max (min).

Bước 1 (Khởi tạo): XOPT=∅; //Phương án tối ưu

FOPT= - ∞ (+ ∞);//Giá trị tối ưu

Bước 2(Lặp):

```
for  each  X∈D do { //lấy mỗi phần tử trên tập phương án
          S = f(X);// tính giá trị hàm mục tiêu cho phương án X
          if ( FOPT<S ) { //Cập nhật phương án tối ưu
          FOPT = S; //Giá trị tối ưu mới được xác lập
          XOPT = X;// Phương án tối ưu mới     }
}
```

Bước 3 (Trả lại kết quả): Return(XOPT, FOPT);

Ưu điểm: - Đơn giản dễ cài đặt.- Có thể thực hiện trên mọi bài toán tối ưu.

Nhược điểm: Chi phí tính toán lớn khi D lớn.⇒ trong thực tế sẽ tính gần đúng theo chi phí thời gian chấp nhận được.

Câu :Thuật toán duyệt toàn bộ giải bài toán cái túi:

***Bài toán cái túi :**

Một nhà thám hiểm có một cái túi có thể chứa các đồ vật với trọng lượng không quá B. Có N đồ vật cần đem theo. Đồ vật thứ i có trọng lượng a_i, có giá trị sử dụng c_i

Hãy tìm cách chọn đồ vật vào túi sao cho tổng giá trị sử dụng các đồ vật trong túi là lớn nhất

=> có thể phát biểu dưới dạng sau :

Giả sử D là tập phương án. Xét mọi X∈D, tìm X* để g(X*) ≤ b và f(X*)→max trong đó :

$$D = \left\{ X = (x_1, x_2, ..., x_n) : \sum_{i=1}^{n} a_i x_i \le b, x_i \in \{0,1\}, i = 1,2,...,n \right\}$$

$$f^* = \max\left\{ f(X) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \le b, x_i \in \{0,1\}, i = 1,2,\ldots,n \right\}$$

***Thuật Toán**

Bước 1 (Khởi tạo): XOPT=∅; //Phương án tối ưu

FOPT= - ∞ ;//Giá trị tối ưu

Bước 2(Lặp):

```
for  each  X∈D do { //lấy mỗi phần tử trên tập phương án
        W = g(X);// tính giá trị hàm khối lượng cho phương án X
        S = f(X);// tính giá trị hàm mục tiêu cho phương án X
        if ( W<=b && FOPT<S ) { //Cập nhật phương án tối ưu
        FOPT = S; //Giá trị tối ưu mới được xác lập
        XOPT = X;// Phương án tối ưu mới     }
}
```

Bước 3 (Trả lại kết quả): Return(XOPT, FOPT);

Câu :Thuật toán nhánh cận giải bài toán tối ưu:

***Bài toán tối ưu:**

Tìm min { f(X) : X∈D }. Trong đó, D là tập hữu hạn các phần tử.

D = { X =(x_1, x_2,..,x_n) ∈ $A_1 \times A_2 \times \ldots$ xA_n: X thỏa mãn tính chất P }

***Ý tưởng:** (Hạn chế các phương án duyệt)

- Thuật toán nhánh cận ⇔ tìm được một hàm g thỏa mãn bất đẳng thức (*):

$g(a_1,a_2, ..,a_k) \le \min\{f(X): X\in D, x_i = a_i, i=1, 2,..,k\}$ (*)

⇒ giá trị $g(a_1, a_2,..,a_k)$ không vượt quá giá trị nhỏ nhất của hàm mục tiêu trên tập con các phương án:

$D(a_1, a_2,..,a_k) = \{ X\in D: x_i = a_i, i= 1, 2, .., k\}$

Hàm g gọi là hàm cận dưới, $g(a_1, a_2, ..,a_k)$ gọi là cận dưới của tập $D(a_1, a_2,..,a_k)$.

-Giả sử có hàm g. Để giảm bớt khối lượng duyệt trên tập phương án trong quá trình liệt kê bằng thuật toán quay lui ta xác định được X* là phương án làm cho hàm mục tiêu có giá trị nhỏ nhất trong số các phương án tìm được f* =f(X*).

Nếu $f^* < g(a_1, a_2,..,a_k)$

thì $f^* < g(a_1, a_2,..,a_k) \le \min \{ f(X):X\in D, x_i = a_i, i=1, 2, ..,k\}$.

⇒ tập $D(a_1, a_2,.., a_k)$ chắc chắn không chứa phương án tối ưu.

⇒ không cần phải tiếp tục phương án bộ phận $(a_1,a_2,..,a_k)$. Tập $D(a_1,a_2,..a_k)$ cũng bị loại bỏ khỏi quá trình duyệt. Nhờ đó, số các tập cần duyệt nhỏ đi trong quá trình tìm kiếm.

***Thuật toán**

```
Branch_And_Bound (k) {
        for ak∈Ak do {
                if (<chấp nhận ak>){
                        xk = ak;
```

if (k==n) <Cập nhật kỷ lục nếu f(X) < f_{OPT}>;
else if (g(a_1, a_2,..,a_k) < f_{OPT}) Branch_And_Bound (k+1);
}
}}

Câu :Thuật toán nhánh cận giải bài toán cái túi :

Bài toán cái túi có thể phát biểu dưới dạng sau :

Giả sử D là tập phương án. Xét mọi X∈D, tìm X* để f(X*)→max trong đó :

$$D = \left\{ X = (x_1, x_2, ..., x_n): \sum_{i=1}^{n} a_i x_i \le b, x_i \in \{0,1\}, i = 1,2,...,n \right\}$$

$$f^* = \max \left\{ f(X) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \le b, x_i \in \{0,1\}, i = 1,2,...,n \right\}$$

(note nếu 1 vật có thể lấy lại nhiều lần thì x_i=Z_+)

Ý tưởng:

Bước 1. Sắp xếp các đồ vật thỏa mãn: $\frac{c_1}{a_1} \ge \frac{c_2}{a_2} \ge ... \ge \frac{c_n}{a_n}$

Fopt = - ∞; Xopt = ∅; b_0 = b; δ_0 = 0;

Bước 2 (Lặp): Xét các bài toán bộ phận cấp k =1, 2,..,n: (Chọn x_k bằng 1 hoặc 0;)

- Trọng lượng còn lại của túi: $$b_k = b - \sum_{i=1}^{k} a_i x_i = b_{k-1} - a_k x_k$$

- Giá trị sử dụng của k đồ vật trong túi: $$\delta_k = \sum_{i=1}^{k} c_i x_i = \delta_{k-1} + c_k x_k$$

- Cận trên của phương án bộ phận cấp k (k < n): $$g_k = \delta_k + b_k . \frac{c_{k+1}}{a_{k+1}}$$

- Nếu b_k < 0 thì dừng và quay lui

Bước 3:(Trả lại kết quả): Phương án tối ưu Xopt; và Giá trị tối ưu Fopt .

Mô hình:Thuật toán

Branch_And_Bound (k) {
for (j = b_k/a_k ; j>=0; j--){
x[k] =j;
δ_k=δ_k+$c_k x_k$; b_k= b_k-$a_k x_k$;
if (k==n) <Ghi nhận kỷ lục>;
else if (δ_k+ (c_{k+1}*b_k)/a_{k+1} >FOPT) Branch_And_Bound(k+1);
δ_k=δ_k-$c_k x_k$; b_k= b_k+$a_k x_k$;
}
}

Câu : thuật toán nhánh cận giải bài toán người du lịch :

Bài toán người du lịch có thể được phát biểu tổng quát dưới dạng sau:

Giả sử D là tập phương án. Xét mọi $X \in D$, tìm X* để $f(X^*) \rightarrow \min$ trong đó :

$$f^* = \min\left\{ f(X) = \sum_{i=1}^{n-1} c[x_i, x_{i+1}] + c[x_n, x_1] : X \in D \right\}$$

$$D = \left\{ X = (x_1, x_2, \ldots, x_n) : x_1 = 1 \wedge x_i \neq x_j, i, j = 1, 2, .., n \right\}$$

Ý tưởng:

-Gọi c_{min} = min {c[i, j], i , j = $1, 2,.., n, i \neq j$ } là giá trị nhỏ nhất của ma trận chi phí. - Phương pháp đánh giá cận dưới của mỗi bài toán bộ phận cấp k được tiến hành như sau. Giả sử ta đang có hành trình bộ phận qua k thành phố:

$T_1 \rightarrow T_{u2} \rightarrow .. \rightarrow T_{uk}$ (T_1 =1).

Khi đó, chi phí của phương án bộ phận cấp k là:

$\delta = c[1, u_2] + c[u_2, u_3] + \ldots + c[u_{k-1}, u_k]$.

Để phát triển hành trình bộ phận này thành hành trình đầy đủ, ta cần phải qua n-k thành phố nữa rồi quay trở về thành phố số 1. Như vậy, ta cần phải qua n-k+1 đoạn đường nữa. Vì mỗi đoạn đường đều có chi phí không nhỏ hơn c_{min}, nên cận dưới của phương án bộ phận có thể đước xác định:

$g(u_1, u_2, .. u_k) = \delta + (n-k+1).c_{min}$.

Thuật toán Brach_And_Bound (k) {

```
        for ( j = 2; j<=n; j++){
                if ( chuaxet[j] ) {
                        x[k] = j; chuaxet[j] = False;
                        δ = δ + c[ x[k-1], x[k]];
                        if ( k==n) Ghi_Nhan();
                        else if ( δ + (n-k+1)*cmin < FOPT)
                                Brach_And_Bound (k+1);
                        chuaxet[j] = True; δ = δ - c[ x[k-1], x[k]];
                }
        }
}
```