B18DCCN424 – LÊ KHẮC NAM

Mục lục

[**I.** **Mô hình thuật toán sinh kế tiếp (Next Generation Algorithm)** 1](#_Toc133358699)

[**1.** **Ý nghĩa và cách hoạt động của thuật toán** 1](#_Toc133358700)

[**2.** **Ví dụ** 2](#_Toc133358701)

[**3.** **Ững dụng thuật toán.** 5](#_Toc133358702)

[**4.** **Đánh giá thuật toán.** 5](#_Toc133358703)

[**5.** **CODE PTIT** 5](#_Toc133358704)

[**II.** **Mô hình thuật toán đệ qui (Recusion Algorithm)** 5](#_Toc133358705)

[**1.** **Ý nghĩa và cách hoạt động của thuật toán** 5](#_Toc133358706)

[**2.** **Ví dụ** 6](#_Toc133358707)

[**3.** **Ững dụng thuật toán.** 6](#_Toc133358708)

[**4.** **Đánh giá thuật toán.** 6](#_Toc133358709)

[**5.** **CODE PTIT** 7](#_Toc133358710)

[**III.** **Mô hình thuật toán quay lui ( Back-track Algorithm)** 7](#_Toc133358711)

[**1.** **Ý nghĩa và cách hoạt động của thuật toán.** 7](#_Toc133358712)

[**2.** **Ví dụ** 7](#_Toc133358713)

[**3.** **Ứng dụng thuật toán.** 9](#_Toc133358714)

[**4.** **Đánh giá thuật toán.** 9](#_Toc133358715)

[**5.** **CODE PTIT** 9](#_Toc133358716)

[**IV.** **Mô hình nhánh cận (Branch-and Bounch Algorithm)** 9](#_Toc133358717)

[**1.** **Ý nghĩa và cách hoạt động của thuật toán** 9](#_Toc133358718)

[**2.** **Ví dụ** 11](#_Toc133358719)

[3. **ứng dụng thuật toán.** 14](#_Toc133358720)

[4. **Đánh giá thuật toán.** 14](#_Toc133358721)

[5. **CODE PTIT** 14](#_Toc133358722)

**BÁO CÁO THU HOẠCH 1**

1. **Mô hình thuật toán sinh kế tiếp (Next Generation Algorithm)**
2. **Ý nghĩa và cách hoạt động của thuật toán**

Mô hình thuật toán sinh được dùng để giải lớp các bài toán liệt kê, bài toán đếm, bài toán tối ưu, bài toán tồn tại thỏa mãn hai điều kiện:

• **Điều kiện 1:** Có thể xác định được một thứ tự trên tập các cấu hình cần liệt kê của bài toán. Biết cấu hình đầu tiên, biết cấu hình cuối cùng.

• **Điều kiện 2:** Từ một cấu hình chưa phải cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình đứng ngay sau nó.

Mô hình thuật toán sinh được biểu diễn thành hai bước: bước khởi tạo và bước lặp.

Tại bước khởi tạo, cấu hình đầu tiên của bài toán sẽ được thiết lập. Điều này bao giờ cũng thực hiện được theo giả thiết của bài toán. Tại bước lặp, quá trình lặp được thực hiện khi gặp phải cấu hình cuối cùng. Điều kiện lặp của bài toán bao giờ cũng tồn tại theo giả thiết của bài toán. Hai chỉ thị cần thực hiện trong thân vòng lặp là đưa ra cấu hình hiện tại và sinh ra cấu hình kế tiếp. Mô hình sinh kế tiếp được thực hiện tùy thuộc vào mỗi bài toán cụ thể.

Tổng quát, mô hình thuật toán sinh được thể hiện như dưới đây.

**Thuật toán Generation;**

**begin**

**Bước1 (Khởi tạo):**

<Thiết lập cấu hình đầu tiên>;

**Bước 2 (Bước lặp):**

**while** (<Lặp khi cấu hình chưa phải cuối cùng>) **do**

<Đưa ra cấu hình hiện tại>;

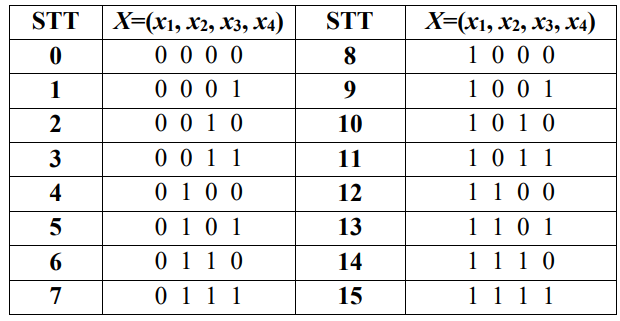
<Sinh ra cấu hình kế tiếp>;

**endwhile;**

**End.**

1. **Ví dụ**

Vector X = (x1, x2, .., xn), trong đó xi = 0, 1 được gọi là một xâu nhị phân có độ dài n. Hãy liệt kê các xâu nhị phân có độ dài n. Ví dụ với n=4, ta sẽ liệt kê được 24 xâu nhị phân độ dài 4 như trong Bảng.



Lời giải:

**Điều kiện 1:** Gọi thứ tự của xâu nhị phân X=(x1, x2,.., xn) là f(X). Trong đó, f(X)= k là số chuyển đồi xâu nhị X thành số ở hệ cơ số 10. Ví dụ, xâu X = (1, 0, 1, 1) được chuyển thành số hệ cơ số 10 là 11 thì ta nói xâu X có thứ tự 11. Với cách quan niệm này, xâu đứng sau xâu có thứ tự 11 là 12 chính là xâu đứng ngay sau xâu X = (1, 0, 1, 1). Xâu đầu tiên có thứ tự là 0 ứng với xâu có n số 0. Xâu cuối cùng có thứ tự là 2^n-1 ứng với xâu có n số 1.

Như vậy, điều kiện 1 của thuật toán sinh đã được thỏa mãn.

**Điều kiện 2:** Về nguyên tắc ta có thể lấy k = f(X) là thứ tự của một xâu bất kỳ theo nguyên tắc ở trên, sau đó lấy thứ tự của xâu kế tiếp là (k + 1) và chuyển đổi (k+1) thành số ở hệ cơ số 10 ta sẽ được xâu nhị phân tiếp theo. Xâu cuối cùng sẽ là xâu có n số 1 ứng với thứ tự k = 2^n-1. Với cách làm này, ta có thể coi mỗi xâu nhị phân là một số, mỗi thành phần của xâu là một bít và chỉ cần cài đặt thuật toán chuyển đổi cơ số ở hệ 10 thành số ở hệ nhị phân. Ta có thể xây dựng thuật toán tổng quát hơn bằng cách xem mỗi xâu nhị phân là một mảng các phần tử có giá trị 0 hoặc 1. Sau đó, duyệt từ vị trí bên phải nhất của xâu nếu gặp số 1 ta chuyển thành 0 và gặp số 0 đầu tiên ta chuyển thành 1. Ví dụ với xâu X = (0, 1, 1, 1) được chuyển thành xâu X= (1, 0, 0, 0), xâu X = (1,0,0,0) được chuyển thành xâu X =(1, 0, 0, 1). Lời giải và thuật toán sinh xâu nhị phân kế tiếp được thể hiện trong chương trình dưới đây.Trong đó, thuật toán sinh xâu nhị phân kế tiếp từ một xâu nhị phân bất kỳ là hàm Next\_Bits\_String().

#include <iostream>

#include <iomanip>

#define MAX 100

using namespace std;

int X[MAX], n, dem = 0; //sử dụng các biến toàn cục X[], n, OK, dem

bool OK =true;

void Init(void){ //khởi tạo xâu nhị phân đầu tiên

cout<<"Nhập n="; cin>>n;

for(int i = 1; i<=n; i++) //thiết lập xâu với n số 0

X[i]=0;

}void Result(void){ //đưa ra xâu nhị phân hiện tại

cout<<"\n Xâu thứ "<<++dem<<":";

for(int i=1; i<=n; i++)

cout<<X[i]<<setw(3);

}

void Next\_Bits\_String(void){ //thuật toán sinh xâu nhị phân kế tiếp

int i=n;

while(i>0 && X[i]){ //duyệt từ vị trí bên phải nhất

X[i]=0; //nếu gặp X[i] = 1 ta chuyển thành 0

i--; //lùi lại vị trí sau

}

if (i>0) X[i]=1; //gặp X[i] =0 đầu tiên ta chuyển thành 1

else OK = false; //kết thúc khi gặp xâu có n số 1

}

int main(void){ //đây là thuật toán sinh

Init(); //thiết lập cấu hình đầu tiên

while(OK){//lặp khi chưa phải cấu hình cuối cùng

Result(); //đưa ra cấu hình hiện tại

Next\_Bits\_String(); //sinh ra cấu hình kế tiếp

}

}

1. **Ững dụng thuật toán.**

* Sinh xâu nhị phân
* Sinh hoán vị
* Sinh tổ hợp

1. **Đánh giá thuật toán.**

**Ưu điểm:**

* Có thể sinh được tất cả các cấu hình, trường hợp của bài toán,

**Nhược điểm:**

* Phải biết cấu hình đầu tiên và cấu hình tiếp theo và cấu hình cuối cùng của bài toán.

1. **CODE PTIT**
2. **Mô hình thuật toán đệ qui (Recusion Algorithm)**
3. **Ý nghĩa và cách hoạt động của thuật toán**

Một đối tượng được định nghĩa trực tiếp hoặc gián tiếp thông qua chính nó được gọi là phép định nghĩa bằng đệ qui. Thuật toán giải bài toán P một cách trực tiếp hoặc gián tiếp thông qua bài toán P’ giống như P được gọi là thuật toán đệ qui giải bài toán P. Một hàm được gọi là đệ qui nếu nó được gọi trực tiếp hoặc gián tiếp đến chính nó.

Tổng quát, một bài toán có thể giải được bằng đệ qui nếu nó thỏa mãn hai điều kiện:

**• Phân tích được:** Có thể giải được bài toán P bằng bài toán P’ giống như P. Bài tóa P’ và chỉ khác P ở dữ liệu đầu vào. Việc giải bài toán P’ cũng được thực hiện theo cách phân tích giống như P.

• **Điều kiện dừng:** Dãy các bài toán P’ giống như P là hữu hạn và sẽ dừng tại một bài toán xác định nào đó.

Thuật toán đệ qui tổng quát có thể được mô tả như sau:

*Thuật toán Recursion ( P )* {

1. Nếu P thỏa mãn điều kiện dừng:

<Giải P với điều kiện dừng>;

2. Nếu P không thỏa mãn điều kiện dừng:

<Giải P’ giống như P:Recursion(P’)>;

}

1. **Ví dụ**

Tìm tổng của n số tự nhiên đầu tiên bằng phương pháp đệ qui.**Lời giải.** Gọi Sn là tổng của n số tự nhiên. Khi đó:

* **Bước phân tích:** dễ dàng phận thấy tổng n số tự nhiên Sn = n + Sn-1, với n>=1.
* **Điều kiện dừng:** S0 = 0 nếu n = 0;

Từ đó ta có lời giải của bài toán như sau: int Tong (int n ) {

if (n ==0 ) return(0); //Điều kiện dừng

else return(n + Tong(n-1)); //Điều kiện phân tích được

}

Chẳng hạn ta cần tìm tổng của 5 số tự nhiên đầu tiên, khi đó:

S = Tong(5)

= 5 + Tong(4)

= 5 + 4 + Tong(3)

= 5 + 4 + 3 + Tong(2)

= 5 + 4 + 3 + 2+ Tong(1)

= 5 + 4 + 3 + 2 + 1 + Tong(0)

= 5 + 4 + 3 + 2 + 1 + 0

= 15

1. **Ững dụng thuật toán.**

* Tính tổng,
* Tính giai thừa
* Tính số fibonacci
* Duyệt cây và đồ thị
* Cài đặt thuật toán chia để trị

1. **Đánh giá thuật toán.**

**Nhược điểm:**

* Mỗi cuộc gọi đệ quy cần thêm dung lượng trong bộ nhớ ngăn xếp.

**Ưu điểm:**

* Kích thước code nhỏ hơn.

1. **CODE PTIT**
2. **Mô hình thuật toán quay lui ( Back-track Algorithm)**
3. **Ý nghĩa và cách hoạt động của thuật toán.**

Giả sử ta cần xác định bộ X =(x1, x2,..,xn) thỏa mãn một số ràng buộc nào đó. Ứng với mỗi thành phần xi ta có ni khả năng cần lựa chọn. Ứng với mỗi khả năng jni dành cho thành phần xi ta cần thực hiện:

* Kiểm tra xem khả năng j có được chấp thuận cho thành phần xi hay không?

Nếu khả năng j được chấp thuận thì ta xác định thành phần xi theo khả năng j.

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 thành phần xi thì ta quay lại bước trước đó (i-1) để thử lại các khả năng còn lại.

Thuật toán quay lui được mô tả như sau:

**Thuật toán Back-Track ( int i )** {

for ( j =<Khả năng 1>; j <=ni; j++ ){

if (<chấp thuận khả năng j>) {

X[i] = <khả năng j>;

if ( i ==n) Result();

else Back-Track(i+1);

}

}

}

1. **Ví dụ**

Duyệt các xâu nhị phân có độ dài n.Lời giải. Xâu nhị phân X = (x1, x2,..,xn)| xi =0, 1. Mỗi xiX có hai lựa chọn xi=0, 1. Cả hai giá trị này đều được chấp thuận mà không cần có thêm bất kỳ điều kiện gì. Thuật toán được mô tả như sau:

void Try ( int i ) {

for (int j =0; j<=1; j++){

X[i] = j;

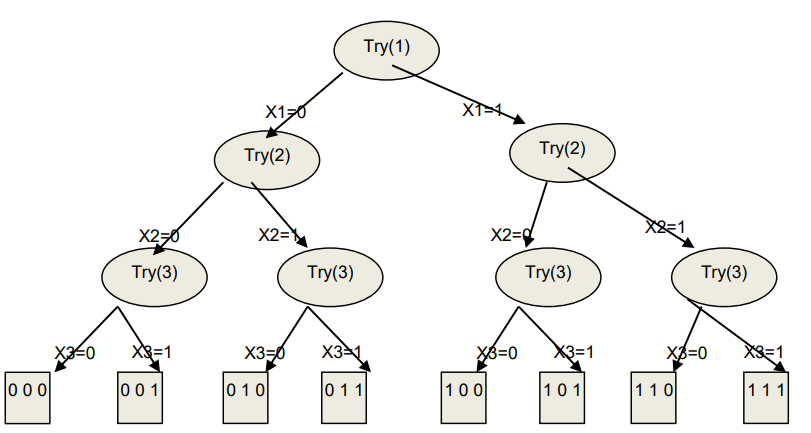
if ( i ==n) Result();

else Try (i+1);

}

}

Khi đó, việc duyệt các xâu nhị phân có độ dài n ta chỉ cần gọi đến thủ tục Try(1). Cây quay lui được mô tả như hình dưới đây.



Chương trình duyệt các xâu nhị phân có độ dài n bằng thuật toán quay lui được thể hiện như dưới đây.

#include <iostream>

#include <iomanip>

#define MAX 100

using namespace std;

int X[MAX], n, dem=0;

void Init(){ //thiết lập độ dài xâu nhị phân

cout<<"\n Nhập n="; cin>>n;

}void Result(void){ //In ra xâu nhị phân X[] = x1, x2,.., xn

cout<<"\n Kết quả "<<++dem<<":";

for(int i =1; i<=n; i++)

cout<<X[i]<<setw(3);

}

void Try(int i){ //thuật toán quay lui

for (int j=0; j<=1; j++){ //duyệt các khả năng j dành cho xi

X[i]=j; //thiết lập thành phần xi là j

if(i==n) //nếu i là thành phần cuối cùng

Result(); // ta đưa ra kết quả

else //trong trường hợp khác

Try(i+1); //ta xác định tiếp thành phần xi+1

}

}

int main(void){

Init();

Try(1);

}

1. **Ứng dụng thuật toán.**

* Bài toán người đi du lịch
* Chuột trong mê cung
* Xếp hậu
* Sudoku

1. **Đánh giá thuật toán.**
2. **CODE PTIT**
3. **Mô hình nhánh cận (Branch-and Bounch Algorithm)**
4. **Ý nghĩa và cách hoạt động của thuật toán**

Thuật toán nhánh cận thường được dùng trong việc giải quyết các bài toán tối ưu tổ hợp. Bài toán được phát biểu dưới dạng sau:

Tìm min { f(X) : XD }, với D = { X =(x1, x2,..,xn) A1xA2 xAn: X thỏa mãn tính chất P }.

* XD được gọi là một phương án của bài toán.
* Hàm f(X) được gọi là hàm mục tiêu của bài toán.
* Miền D được gọi là tập phương án của bài toán.

Để giải quyết bài toán trên, ta có thể dùng thuật toán quay lui duyệt các phần tử XD, phần tử X\* làm cho F(X\*) đạt giá trị nhỏ nhất (lớn nhất) là phương án tối ưu của bài toán. Thuật toán nhánh cận có thể giải được bài toán đặt ra nếu ta xây dựng được một hàm g xác định trên tất cả phương án bộ phận cấp k của bài toán sao cho:

g(a1, a2, ..,ak)≤ min { f(X): XD, xi=ai, i=1, 2,..,k}

Nói cách khác, giá trị của hàm g tại phương án bộ phận cấp k là g(a1, a2,..,ak) không vượt quá giá trị nhỏ nhất của hàm mục tiêu trên tập con các phương án.

D(a1, a2,..,ak) = { XD: xi = ai, i= 1, 2, .., k}

Giá trị của hàm g(a1, a2, ..,ak) là cận dưới của hàm mục tiêu trên tập

D(a1, a2,..,ak).

Hàm g được gọi là hàm cận dưới, g(a1, a2, ..,ak) gọi là cận dưới của tập D(a1,a2,..,ak).

Giả sử ta đã có hàm g. Để giảm bớt khối lượng duyệt trên tập phương án, 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\*). Ta gọi X\* là phương án tốt nhất hiện có, f\* là kỷ lục hiện tại.

Nếu

f\* < g(a1, a2,..,ak)

thì

f\*< g(a1, a2,..,ak) ≤ min { f(X):XD, xi = ai, i=1, 2, ..,k}.

Điều này có nghĩa tập D(a1, a2,.., ak) chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần phải triển khai phương án bộ phận (a1,a2,..,ak). Tập D(a1,a2,..ak) cũng bị loại bỏ khỏi quá trình duyệt. Nhờ đó, số các phương án cần duyệt nhỏ đi trong quá trình tìm kiếm.

Thuật toán Branch\_And\_Bound (k) {

for akAk do {

if (<chấp nhận ak>){

xk = ak;

if ( k==n )

<Cập nhật kỷ lục>;

else if ( g(a1, a2,..,ak) ≤ f\*)

Branch\_And\_Bound (k+1) ;

}

}

}

1. **Ví dụ**

Giải bài toán cái túi bằng thuật toán nhánh cận. Một nhà thám hiểm cần đem theo một cái túi trọng lượng không quá b. Có n đồ vật cần đem theo. Đồ vật thứ i có trọng lượng ai và giá trị sử dụng ci. Hãy tìm cách đưa đồ vật vào túi cho nhà thám hiểm sao cho tổng giá trị sử dụng các đồ vật trong túi là lớn nhất.

**Lời giải.** Bạn đọc tự tìm hiểu lời giải của bài toán trong các tài liệu liên quan. Dưới đây là chương trình cài đặt bài toán với dữ liệu vào trong file caitui.in và kết quả ra trong file ketqua.out theo khuôn dạng:**Caitui.in:**

* Dòng đầu tiên ghi lại hai số n và b tương ứng với số lượng đồ vật và trọng

lượng túi.

* N dòng kế tiếp, mỗi dòng ghi lại bộ đôi ai, ci tương ứng với trọng lượng và giá trị sử dụng đồ vật i.

**Ketqua.out:**

* Dòng đầu tiên ghi lại giá trị tối ưu của bài toán.
* Dòng kế tiếp ghi lại phương án tối ưu của bài toán.

Ví dụ:

Caitui.in

4 8

5 10

3 5

2 3

4 6

ketqua.out

15

1 1 0 0

Chương trình giải quyết bài toán được thực hiện như dưới đây:

#include <iostream>

#include <fstream>

#define MAX 100

using namespace std;

int X[MAX]; // phương án tối ưu của bài toán

int n; //số lượng đồ vật

float b, weight=0; //trọng lượng túi

float cost=0; // giá trị sử dụng phương án hiện tại

int XOPT[MAX]; //phương án tối ưu của bài toán

float FOPT=0; //giá trị tối ưu của bài toán

float A[MAX], C[MAX];//vector trọng lượng và giá trị sử dụng

void Init (void) { //đọc dữ liệu

ifstream fp("caitui.in"); fp>>n;fp>>b;

for(int i=1; i<=n; i++){

fp>>A[i]>>C[i];

}

fp.close();

}void Result(void) {//đưa ra giá trị và phương án tối ưu

cout<<"Giá trị tối ưu:"<<FOPT<<endl;

cout<<"Phương án tối ưu:";

for(int i=1; i<=n; i++)

cout<<XOPT[i]<<" ";

}

void Update(void) { //cập nhật phương án tối ưu

if (cost> FOPT){

FOPT =cost;

for (int i=1; i<=n; i++)

XOPT[i]=X[i];

}

}

void Back\_Track(int i){

int j, t = (int)((b-weight)/A[i]); //giá trị khởi đầu

for(int j= t; j>=0; j--){

X[i] = j;

weight = weight+A[i]\*X[i]; //trọng lượng phương án bộ phận

cost = cost + C[i]\*X[i]; //giá trị phương án bộ phận

if (i==n) Update(); //ghi nhận kỷ lục

else if ( cost + ( C[i+1]\*((b- weight)/A[i+1]))>FOPT) //kiểm tra cận

Back\_Track(i+1);

weight = weight-A[i]\*X[i];

cost = cost - C[i]\*X[i];

}

}

int main (void ){ //chương trình chính

Init(); Back\_Track(1);Result();

}

1. **ứng dụng thuật toán.**

* Bài toán cái túi

1. **Đánh giá thuật toán.**
2. **CODE PTIT**