**I. Tổng quan**

**1. Giới thiệu phương pháp**

Trong lập trình cũng như trong thực tế, chắc hẳn chúng ta đều đã gặp những bài toán với yêu cầu tìm kết quả ***tốt nhất*** thỏa mãn một hoặc một số điều kiện nào đó. Sự thật là chúng ta gặp các bài toán này khá thường xuyên, thậm chí vô cùng thực tiễn, chẳng hạn như:

* Tìm cách trả số tiền T với ntờ tiền có mệnh giá cho trước, sao cho số tờ tiền cần sử dụng là nhỏ nhất?
* Trong các nhà máy sản xuất vỏ lon đồ uống, các nhà thiết kế luôn luôn tìm cách thiết kế sao cho diện tích toàn phần của vỏ lon là nhỏ nhất nhằm giảm thiểu chi phí nguyên liệu. Bài toán đặt ra là với thể tích cần thiết là V, làm sao để tạo ra vỏ lon hình trụ có diện tích toàn phần nhỏ nhất? ......

Có rất nhiều bài toán như vậy trong Tin học, và chúng được gọi là các ***bài toán tối ưu***. Thông thường, người ta sẽ hay nghĩ đến phương pháp ***Quy hoạch động*** khi giải các bài toán tối ưu, tuy nhiên, phương pháp này chỉ có thể áp dụng nếu như bài toán đang xét có bản chất đệ quy mà thôi. Thực tế có nhiều bài toán tối ưu không có thuật toán nào thực sự hữu hiệu để giải quyết, mà vẫn phải sử dụng mô hình xem xét tất cả các phương án rồi đánh giá chúng để chọn ra phương án tốt nhất.

Phương pháp ***Nhánh Cận (Branch and Bound)*** chính là một phương pháp cải tiến từ phương pháp ***Quay lui***, được sử dụng để tìm nghiệm của bài toán tối ưu.

**2. Ý tưởng**

Bước đầu tiên của phương pháp vẫn giống với ý tưởng của quay lui: Tìm cách biểu diễn nghiệm của bài toán dưới dạng một vector (x1­, x2, …, xn), mỗi thành phần xi​ được chọn ra từ tập các ứng cử viên Si.

Bước tiếp theo sẽ hơi khác một chút: Nếu như ở phương pháp Quay lui, chỉ cần tuần tự chọn các ứng cử viên cho từng thành phần của vector nghiệm, thì ở phương pháp Nhánh và cận, mỗi nghiệm X = (x1­, x2, …, xn) của bài toán sẽ được đánh giá ***độ tốt*** bằng một hàm . Vì đây là bài toán tối ưu, nên mục tiêu của chúng ta là đi tìm nghiệm có hàm tốt nhất, thường là ***lớn nhất*** hoặc ***nhỏ nhất***.

Bước thứ 3 là xây dựng nghiệm của bài toán. Giả sử, các bạn đã xây dựng được i thành phần của nghiệm là (x1­, x2, …, xi) và chuẩn bị mở rộng nghiệm thành (x1­, x2, …, xi, xi+1) . Nếu như bằng một cách nào đó, các bạn đánh giá được độ tốt của toàn bộ các nghiệm mở rộng của nhánh này (x1­, x2, …, xi, xi+1, …)  và biết rằng không có nghiệm nào trong nhánh này "tốt hơn" nghiệm tốt nhất tại thời điểm đó, thì việc mở rộng tiếp từ (x1­, x2, …, xi) sẽ là không cần thiết nữa, mà thay vào đó ta sẽ chuyển qua chọn ứng cử viên tiếp theo cho thành phần xi ​luôn.

Bằng phương pháp trên, ta sẽ loại bỏ được những nhánh không cần thiết để không duyệt vào các phương án đó, từ đó việc tìm ra nghiệm tối ưu sẽ nhanh hơn. Tuy nhiên, việc đánh giá được "độ tốt" của các nghiệm mở rộng không phải việc đơn giản, nhưng nếu làm được như vậy thì giải thuật sẽ thực thi nhanh hơn nhiều so với quay lui.

**3. Lược đồ giải thuật**

Để thực hiện giải thuật Nhánh và Cận, các bạn có thể sử dụng một hàm đệ quy giống như giải thuật Quay lui, nhưng thêm phần đánh giá nghiệm trước khi xây dựng thành phần thứ i*i*. Ngoài ra, ta cần sử dụng thêm một biến \text{best\_solution} để ghi nhận nghiệm tốt nhất hiện tại.

Dưới đây là mô hình cài đặt bằng C++:

void branch\_and\_bound(i)

{

// Đánh giá các nghiệm mở rộng

if ({Các\_nghiệm\_mở\_rộng\_đều\_không\_tốt\_hơn\_best\_solution})

return;

for (value in S[i])

{

x[i] = value; // Ghi nhận thành phần thứ i.

if ({Tìm\_thấy\_nghiệm})

best\_solution = X; // Cập nhật lại best\_solution bằng nghiệm vừa tìm được.

else

branch\_and\_bound(i + 1); // Chưa tìm thấy nghiệm thì xây dựng tiếp.

{Loại\_bỏ\_thành\_phần\_thứ\_i}

}

}

# II. Một số bài toán minh họa

## 1. Rút tiền ATM

### Đề bài

Một máy ATM hiện có n tờ tiền có giá trị lần lượt là t1, t2, ..., tn ​. Hãy tìm ra cách trả số tiền S sao cho số tờ tiền phải sử dụng là ***ít nhất***?

***Input:***

* Dòng đầu tiên chứa hai số nguyên dương n và S (1 ≤ n ≤ 20; 1 ≤ S ≤ 1000)*.*
* Dòng thứ hai chứa n*n* số nguyên dương t1, t2, ..., tn ​ phân tách nhau bởi dấu cách (1 ≤ t\_i ≤ 1000; ∀i: 1 ≤ i ≤ n).

***Output:***

* Nếu như có thể trả số tiền S thì in ra số tờ tiền ít nhất cần sử dụng và các tờ tiền được chọn, ngược lại in ra -1.

***Sample Input:***

10 390

200 10 20 20 50 50 50 50 100 100

***Sample Output:***

5

20 20 50 100 200

### Phân tích ý tưởng

Nghiệm của bài toán có thể biểu diễn dưới dạng một vector gồm toàn các bit nhị phân 0 - 1là x1, x2, ..., xn ​ với ý nghĩa: xi = 0 là tờ tiền thứ i không được chọn, xi = 1 là tờ tiền thứ iđược chọn.

Mục tiêu chúng ta đang cần tìm một bộ nghiệm sao cho:

Giả sử các ban đã xây dựng được i*i* thành phần của nghiệm là (x1, x2, ..., xi), tổng số tờ tiền đã sử dụng là cnt và số tiền đã trả được là sum, thì ta nhận xét thấy:

* Số tiền còn lại cần trả là S - sum.
* Nếu gọi tmax[i + 1] là giá trị của tờ tiền lớn nhất trong các tờ tiền còn lại (tmax[i + 1] = max(ti+1, ti + 2, ..., tn)), thì ít nhất ta cần sử dụng thêm \frac{S - sum}{t\_{max}[i + 1]}*tmax*​[*i*+1]*S*−*sum*​ tờ tiền nữa, tức là tổng số tờ tiền tối thiểu cần dùng của nhánh phương án này là cnt + \frac{S - sum}{t\_{max}[i + 1]}*cnt*+*tmax*​[*i*+1]*S*−*sum*​.

Gọi số tờ tiền của cách trả tốt nhất hiện tại là \text{cnt\_best}, thì nếu như cnt + \frac{S - sum}{t\_{max}[i]} \ge \text{best\_cnt}, ta sẽ không cần phải mở rộng các nghiệm từ (x1, x2, ..., x\_i)(*x*1​,*x*2​,…,*xi*​) nữa.

Để kiểm soát các tờ tiền được chọn, mình sử dụng thêm hai mảng là \text{mark}mark để đánh dấu các tờ tiền được chọn trong một phương án, và \text{mark\_best} để đánh dấu các tờ tiền được chọn trong phương án tốt nhất.

### Cài đặt

#include <bits/stdc++.h>

using namespace std;

const int maxn = 21;

int n, S, cnt, cnt\_best, sum, a[maxn], mark[maxn], mark\_best[maxn], maxmoney[maxn];

void enter()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(0); cout.tie(0);

cin >> n >> S;

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

cin >> t[i];

}

void create\_data()

{

// cnt\_best là số tờ tiền sử dụng trong phương án tốt nhất.

// Ban đầu chưa có phương án nào, gán cnt\_best = n + 1.

best\_cnt = n + 1;

// t\_max[i] lưu giá trị của tờ tiền lớn nhất từ i tới n.

t\_max[n] = t[n];

for (int i = n - 1; i >= 0; --i)

t\_max[i] = max(t\_max[i + 1], t[i]);

}

// Nếu tìm được một phương án tốt hơn thì cập nhật lại kết quả.

void update\_best\_solution()

{

if (sum == S && cnt < cnt\_best)

{

cnt\_best = cnt;

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

mark\_best[i] = mark[i];

}

}

// In kết quả.

void printf\_result()

{

// Không tìm được cách trả tiền, in ra -1.

if (cnt\_best == n + 1)

cout << -1;

else // Tìm được thì in ra cách trả đó.

{

cout << cnt\_best << endl;

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

if (mark\_best[i])

cout << t[i] << ' ';

}

}

void branch\_and\_bound(int i)

{

// Nếu nghiệm mở rộng của nhánh này không tốt hơn thì return.

if (cnt + (S - sum) / t\_max[i + 1] >= cnt\_best)

return;

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

{

// Ghi nhận thành phần thứ i.

sum = sum + a[i] \* j;

mark[i] = j;

cnt += j;

if (i == n)

update\_best\_solution();

else if (sum <= S)

branch\_and\_bound(i + 1);

// Loại bỏ thành phần thứ i.

sum -= t[i] \* j;

cnt -= j;

}

}

int main()

{

enter();

create\_data();

branch\_and\_bound(1);

printf\_result();

return 0;

}

## 2. Bài toán Người du lịch

### Đề bài

Có n*n* thành phố đánh số từ 11 tới n*n*. Giữa các cặp thành phố có thể có hoặc không có đường nối hai chiều, mạng lưới đường này được mô tả bằng một ma trận C\_{u, v},*Cu*,*v*​, với ý nghĩa C\_{u, v} = C\_{v, u}*Cu*,*v*​=*Cv*,*u*​ là chi phí để di chuyển giữa hai thành phố u*u* và v*v*.

Một người du lịch xuất phát từ thành phố 1,1, người này muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng một lần rồi quay trở lại thành phố 11.

***Yêu cầu:*** Hãy tìm một hành trình cho người đó sao cho chi phí di chuyển là ít nhất?

***Input:***

* Dòng đầu tiên chứa số nguyên dương n \ (1 ≤ n ≤ 20)*n* (1≤*n*≤20).
* n*n* dòng tiếp theo, mỗi dòng chứa n*n* số nguyên dương không vượt quá 100100 biểu thị ma trận C*C*.

***Output:***

* Dòng đầu tiên ghi chi phí nhỏ nhất.
* Dòng thứ hai ghi một hành trình tìm được.

***Sample Input:***

4

0 20 35 42

20 0 34 30

35 34 0 12

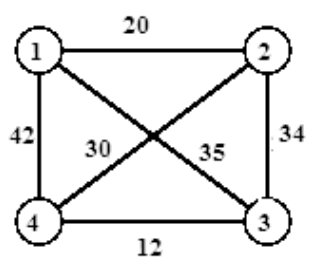
42 30 12 0

***Sample Output:***

97

1 2 4 3 1

***Hình minh họa:***



### Phân tích ý tưởng

Vector nghiệm của bài toán là một dãy (x1 = 1, x2, x\_3, ..., x\_n, x\_{n + 1} = 1);(*x*1​=1,*x*2​,*x*3​,…,*xn*​,*xn*+1​=1); với điều kiện giữa hai thành phố x\_i*xi*​ và x\_{i + 1}*xi*+1​ phải có đường đi trực tiếp. Ngoài ra, chỉ có thành phố 11 được phép lặp lại 22 lần. Vì thế, có thể thấy dãy (x1, x2, ..., x\_n)(*x*1​,*x*2​,…,*xn*​) là một hoán vị của (1, 2, ..., n)(1,2,…,*n*).

Ý tưởng duyệt quay lui như sau: Khi đã xây dựng được (x1, x2,..., x\_i),(*x*1​,*x*2​,...,*xi*​), thì x\_{i + 1}*xi*+1​ có thể chọn một trong các thành phố mà có đường nối trực tiếp với nó, đồng thời chưa được chọn. Tuy nhiên, ta có thể áp dụng Nhánh và Cận để giảm độ phức tạp như sau:

* Gọi chi phí tốt nhất hiện tại là \text{best\_cost}.
* Với mỗi bước thử chọn x\_i,*xi*​, kiểm tra xem chi phí đường đi tính tới lúc đó có lớn hơn hoặc bằng chi phí tốt nhất hiện tại hay không. Nếu đã lớn hơn thì chọn ngay giá trị khác cho x\_i,*xi*​, bởi vì có đi tiếp theo nhánh này cũng sẽ chỉ tạo ra chi phí lớn hơn mà thôi.
* Tới khi chọn được một giá trị x\_n*xn*​ thì cần kiểm tra xem chi phí tới x\_n*xn*​ cộng thêm chi phí từ x\_n*xn*​ về 11 có tốt hơn chi phí tốt nhất hiện tại không? Nếu có thì cập nhật lại cách đi tốt nhất.

### Cài đặt

Trong cài đặt dưới đây, giả thiết rằng giữa mọi cặp thành phố đều tồn tại đường đi, và chi phí c\_{u, v}*cu*,*v*​ luôn bằng 00 nếu như u = v*u*=*v*.

Mảng \text{visited}visited dùng để đánh dấu một thành phố đã được thăm hay chưa trong một cấu hình X*X*. Mảng x*x* sử dụng để lưu cấu hình hiện tại, còn mảng \text{x\_best} sử dụng để lưu cấu hình tốt nhất với \text{best\_cost} là chi phí tốt nhất tìm được.

#include <bits/stdc++.h>

#define int long long

#define task "tsp."

#define inf 1e9 + 7

using namespace std;

const int maxn = 21;

int n, current\_cost, best\_cost;

int visited[maxn], x\_best[maxn], x[maxn], c[maxn][maxn];

void enter()

{

cin >> n;

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

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

cin >> c[i][j];

// Khởi tạo trước thành phố đầu tiên là 1, đồng thời đánh dấu nó đã thăm.

x[1] = 1;

visited[1] = 1;

// Khởi tạo chi phí tối ưu bằng +oo, giả sử phương án hiện tại đang rất tệ.

best\_cost = inf;

}

// Cập nhật kết quả tốt nhất.

void update\_best\_solution(int current\_cost)

{

if (current\_cost + c[x[n]][1] < best\_cost)

{

best\_cost = current\_cost + c[x[n]][1];

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

x\_best[i] = x[i];

}

}

// In ra phương án tốt nhất tìm được.

void print\_best\_solution()

{

cout << best\_cost << endl;

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

cout << x\_best[i] << "->";

cout << 1;

}

// Giải thuật nhánh và cận.

void branch\_and\_bound(int i)

{

if (current\_cost >= best\_cost)

return;

for (int j = 2; j <= n; ++j)

if (!visited[j])

{

visited[j] = 1;

x[i] = j;

current\_cost += c[x[i - 1]][j];

// Đã sinh xong một cấu hình, cập nhật chi phí tốt nhất.

if (i == n)

update\_best\_solution(current\_cost);

// Chưa sinh xong, tiếp tục sinh thành phần tiếp theo với chi phí tăng thêm.

else

branch\_and\_bound(i + 1);

visited[j] = 0;

current\_cost -= c[x[i - 1]][j];

}

}

main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(nullptr);

enter();

branch\_and\_bound(2);

print\_best\_solution();

return 0;

}

# III. Tài liệu tham khảo

* <https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/>
* <https://www.geeksforgeeks.org/branch-and-bound-algorithm/>
* [Sách Giải thuật và Lập trình - thầy Lê Minh Hoàng](https://cuongquach.com/ebook-giai-thuat-va-lap-trinh-lmh-pdf.html).
* [Tài liệu giáo khoa chuyên Tin quyển 1 - thầy Hồ Sĩ Đàm](https://downloadsachmienphi.com/tai-lieu-giao-khoa-chuyen-tin-quyen-1/).