**BÁO CÁO CHUYÊN ĐỀ**

**MỘT SỐ BÀI TOÁN**

**QUY HOẠCH ĐỘNG ĐIỂN HÌNH**

MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG ĐIỂN HÌNH

# A. Khái niệm về phương pháp quy hoạch động

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức là việc tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con.

Ðối với một số bài toán đệ quy, nguyên lý chia để trị (divide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Ðể giải quyết một bài toán lớn, ta chia nó thành nhiều bài toán con cùng dạng với nó để có thể giải quyết độc lập.

Trong phương án quy hoạch động, nguyên lý chia để trị càng được thể hiện rõ: Khi không biết phải giải quyết những bài toán con nào, ta sẽ đi giải quyết toàn bộ các bài toán con và lưu trữ những lời giải hay đáp số của chúng với mục đích sử dụng lại theo một sự phối hợp nào đó để giải quyết những bài toán tổng quát hơn.

Ðó chính là điểm khác nhau giữa Quy hoạch động và phép phân giải đệ quy và cũng là nội dung phương pháp quy hoạch động:

- Phép phân giải đệ quy bắt đầu từ bài toán lớn phân ra thành nhiều bài toán con và đi giải từng bài toán con đó. Việc giải từng bài toán con lại đưa về phép phân ra tiếp thành nhiều bài toán nhỏ hơn và lại đi giải các bài toán nhỏ hơn đó bất kể nó đã được giải hay chưa.

- Quy hoạch động bắt đầu từ việc giải tất cả các bài toán nhỏ nhất (bài toán cơ sở) để từ đó từng bước giải quyết nhưng bài toán lớn hơn, cho tới khi giải được bài toán lớn nhất (bài toán ban đầu).

Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động.

Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi của quy hoạch động.

Tập các bài toán có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là cơ sở quy hoạch động.

Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động.

Trước khi áp dụng phương pháp quy hoạch động ta phải xét xem phương pháp đó có thỏa mãn những yêu cầu dưới đây không:

- Bài toán lớn phải phân rã được thành nhiều bài toán con, mà sự phối hợp lời giải của các bài toán con đó cho ta lời giải của bài toán lớn.

- Vì quy hoạch động là đi giải tất cả các bài toán con, nên nếu không đủ không gian vật lý lưu trữ lời giải (bộ nhớ, đĩa, …) để phối hợp chúng thì phương pháp quy hoạch động cũng không thể thực hiện được.

- Quá trình từ bài toán cơ sở tìm ra lời giải bài toán ban đầu phải qua hữu hạn bước. Các bước cài đặt một chương trình sử dụng quy hoạch động:

- Giải tất cả các bài toán cơ sở (thông thường rất dễ), lưu các lời giải vào bảng phương án.

- Dùng công thức truy hồi phối hợp những lời giải của các bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải của các bài toán lớn hơn rồi lưu chúng vào bảng phương án. Cho tới khi bài toán ban đầu tìm được lời giải.

- Dựa vào bảng phương án, truy vết tìm ra nghiệm tối ưu.

Cho tới nay, vẫn chưa có một định lý nào cho biết một cách chính xác những bài toán nào có thể giải quyết hiệu quả bằng quy hoạch động. Tuy nhiên để biết được bài toán có thể giải bằng quy hoạch động hay không, ta có thể đặt câu hỏi:

1. “Một nghiệm tối ưu của bài toán lớn có phải là sự phối hợp các nghiệm tối ưu của các bài toán con hay không?”

2. “Liệu có thể nào lưu trữ được nghiệm các bài toán con dưới một hình thức nào đó để phối hợp tìm được ngiệm bài toán lớn?”.

# B. Một số bài tập điển hình

# I. Dãy con tăng dài nhất

## 1. Mô hình

Cho dãy số nguyên a1, a2, a3, …, aN. Hãy xác định dãy con tăng có nhiều phần tử nhất của dãy này. Giới hạn 2 ≤ N ≤ 103; |ai | ≤ 109

## 2. Công thức QHĐ

Gọi L[i] là độ dài dãy con tăng dần dài nhất các phần tử xét từ a[1..i] và phần tử cuối cùng phải là a[i]

Vậy yêu cầu của bài toán ban đầu: max(L[i])

Tính L[i]

L[i] = max(1, L[j] + 1), với điều kiện: j < i, a[j] <= a[i]

*Bài toán con cơ bản:*

L[1]: là độ dài dãy con tăng dần dài nhất các phần tử xét từ a[1..1] và phần tử cuối cùng phải là a[1]

Vậy: L[1] = 1

## 3. Cài đặt

#include <fstream>

**using namespace** std;

**const** **int** MAXN = 1000;

#define fi "DAYCON.INP"

#define fo "DAYCON.OUT"

**int** n;

**float** a[MAXN];

**int** L[MAXN], Truoc[MAXN];

fstream f;

**void** Nhap() {

f.open(fi, ios::in);

f >> n;

**int** i;

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

f >> a[i];

}

f.close();

}

**void** QHD() {

L[1] = 1;

Truoc[1] = -1;

**int** i, j;

**for** (i=2; i<=n; i++) {

L[i] = 1;

Truoc[i] = -1;

**for** (j=1; j < i; j++)

**if** (a[j] < a[i])

**if** (L[i] < L[j] + 1) {

L[i] = L[j] + 1;

Truoc[i] = j;

}

}

}

**void** TruyVet(**int** i) {

**if** (Truoc[i] == -1) f << a[i];

**else** {

TruyVet(Truoc[i]);

f << " -> " << a[i];

}

}

**void** InKQ() {

f.open(fo, ios::out);

**int** i;

**int** maxLen = -1;

**int** i0;

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

**if** (maxLen < L[i]) {

maxLen = L[i];

i0 = i;

}

f << maxLen << endl;

TruyVet(i0);

f.close();

}

**int** main() {

Nhap();

QHD();

InKQ();

}

## 4. Một số bài toán khác

### a) Bố trí phòng họp

***Đề bài***

Có n cuộc họp, cuộc họp thứ i bắt đầu vào thời điểm ai và kết thúc ở thời điểm bi. Do chỉ có một phòng hội thảo nên 2 cuộc họp bất kì sẽ được cùng bố trí phục vụ nếu khoảng thời gian làm việc của chúng chỉ giao nhau tại đầu mút. Hãy bố trí phòng họp để phục vụ được nhiều cuộc họp nhất.

***Công thức QHĐ***

Cuộc họp j < i thì: b[j] <= b[i]

b[j] <= a[i]

- Sắp xếp tăng dần theo thứ tự của thời điểm kết thúc (sắp xếp mảng b tăng dần)

- Sau khi sắp xếp:

Gọi L[i]: là số lượng cuộc họp được bố trí nhiều nhất xét từ cuộc họp 1..i và cuộc họp cuối cùng phải là cuộc họp i

Vậy:

L[i] = max(1, L[j] + 1), với điều kiện: j < i, b[j] <= a[i]

***Cài đặt***

#include <fstream>

**using namespace** std;

**const** **int** MAXN = 1000;

#define fi "HOP.INP"

#define fo "HOP.OUT"

**int** n;

**float** a[MAXN]; b[MAXN];

**int** L[MAXN], Truoc[MAXN];

fstream f;

**void** Nhap() {

f.open(fi, ios::in);

f >> n;

**int** i;

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

f >> a[i] >> b[i];

}

f.close();

}

**void** DoiCho(**float** & a, **float** & b) {

**float** tg = a;

a = b;

b = tg;

}

**void** SapXep(){

**int** i, j;

**for** (i=1; i<=n-1; i++)

**for** (j=i+1; j<=n; j++)

**if** (b[i] > b[j]) {

DoiCho(b[i], b[j]);

DoiCho(a[i], a[j]);

}

}

**void** QHD() {

L[1] = 1;

Truoc[1] = -1;

**int** i, j;

**for** (i=2; i<=n; i++) {

L[i] = 1;

Truoc[i] = -1;

**for** (j=1; j < i; j++)

**if** (b[j] <= a[i])

**if** (L[i] < L[j] + 1) {

L[i] = L[j] + 1;

Truoc[i] = j;

}

}

}

**void** TruyVet(**int** i) {

**if** (Truoc[i] == -1) f << a[i] << " " << b[i] << endl;

**else** {

TruyVet(Truoc[i]);

f << a[i] << " " << b[i] << endl;

}

}

**void** InKQ() {

f.open(fo, ios::out);

**int** i;

**int** maxLen = -1;

**int** i0;

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

**if** (maxLen < L[i]) {

maxLen = L[i];

i0 = i;

}

f << maxLen << endl;

TruyVet(i0);

f.close();

}

**int** main() {

Nhap();

SapXep();

QHD();

InKQ();

}

### b) Cho thuê máy

***Đề bài***

Trung tâm tính toán hiệu năng cao nhận được đơn đặt hàng của n khách hàng. Khách hàng i muốn sử dụng máy trong khoảng thời gian từ ai đến bi và trả tiền thuê là ci. Hãy bố trí lịch thuê máy để tổng số tiền thu được là lớn nhất mà thời gian sử dụng máy của 2 khách hàng bất kì được phục vụ đều không giao nhau (cả trung tâm chỉ có 1 máy cho thuê)

***Công thức QHĐ***

Gọi L[i] là tổng số tiền lớn nhất thu được khi xét các đơn hàng từ 1..i và đơn hàng cuối cùng là đơn hàng i

Vậy:

L[i] = max(c[i], L[j] + c[i]), với điều kiện: j < i, b[j] <= a[i]

*Bài toán con cơ bản:*

L[1]: là tổng số tiền lớn nhất thu được khi xét các đơn hàng từ 1..1 và đơn hàng cuối cùng là đơn hàng 1

Vậy:

L[1] = c[1]

# ***Cài đặt***

include <fstream>

**using namespace** std;

**const** **int** MAXN = 1000;

#define fi "THUEMAY.INP"

#define fo "THUEMAY.OUT"

**int** n;

**float** a[MAXN], b[MAXN], c[MAXN], L[MAXN];

**int** Truoc[MAXN];

fstream f;

**void** Nhap(){

f.open(fi, ios::in);

f >> n;

**int** i;

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

f >> a[i] >> b[i] >> c[i];

}

f.close();

}

**void** DoiCho(**float** & a, **float** & b){

**float** tg = a;

a = b;

b = tg;

}

**void** Sapxep(){

**int** i, j;

**for** (i=1; i<=n-1; i++)

**for** (j=i+1; j<=n; j++)

**if** (b[i] > b[j]){

DoiCho(b[i],b[j]);

DoiCho(a[i],a[j]);

DoiCho(c[i],c[j]);

}

}

**void** QHD(){

L[1] = c[1];

Truoc[1] = -1;

**int** i, j;

**for** (i=2; i<=n; i++){

L[i] = c[i];

Truoc[i] = -1;

**for** (j=1; j<i; j++)

**if** (b[j] <= a[i])

**if** (L[i] < L[j] + c[i]){

L[i] = L[j] + c[i];

Truoc[i] = j;

}

}

}

**void** TruyVet(**int** i){

**if** (Truoc[i] == -1) f << a[i] << " " << b[i] << " " << c[i] << endl;

**else**{

TruyVet(Truoc[i]);

f << a[i] << " " << b[i] << " " << c[i] << endl;

}

}

**void** InKQ(){

f.open(fo, ios::out);

**int** i;

**int** max = -1;

**int** k;

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

**if** (max < L[i]){

max = L[i];

k = i;

}

f << max << endl;

TruyVet(k);

f.close();

}

**int** main(){

Nhap();

Sapxep();

QHD();

InKQ();

}

# II. Vali (B)

## 1. Mô hình

Có n đồ vật, vật thứ i có trọng lượng a[i] và giá trị b[i]. Hãy chọn ra một số các đồ vật, mỗi vật một cái để xếp vào 1 vali có trọng lượng tối đa W sao cho tổng giá trị của vali là lớn nhất.

## 2. Công thức QHĐ

Gọi L[i, j] là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..i và trọng lượng không vượt quá j

Kết quả của bài toán là: L[n, W]: là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..n và trọng lượng không vượt quá W

Tính L[i, j]

Có 2 trường hợp xảy ra:

\* TH1: w[i] > j

L[i, j] = L[i-1, j]

\* TH2: w[i] <= j

Có 2 trường hợp xảy ra:

- Chọn đồ vật thứ i

L[i, j] = v[i] + L[i-1, j - w[i]]

- Không chọn đồ vật thứ i

L[i, j] = L[i-1, j]

L[i, j] = max2(v[i] + L[i-1, j - w[i]], L[i-1, j])

*Bài toán con cơ bản:*

L[0, j]: là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..0 và trọng lượng không vượt quá j

L[0, j] = 0

L[i, 0]: là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..i và trọng lượng không vượt quá 0

L[i, 0] = 0

## 3. Cài đặt

#include <fstream>

#include <string.h>

**using namespace** std;

#define fi "VALI.INP"

#define fo "VALI.OUT"

**const** **int** MAXN = 1000;

**const** **int** MAXW = 1000;

fstream f;

**int** n, W;

**int** w[MAXN];

**float** v[MAXN];

**float** L[MAXN][MAXW];

**void** Nhap() {

f.open(fi, ios::in);

f >> n >> W;

**int** i;

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

f >> w[i] >> v[i];

f.close();

}

**void** QHD() {

memset(L, 0, **sizeof**(L));

**int** i, j;

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

**for** (j = 1; j <= W; j++)

**if** (w[i] > j)

L[i][j] = L[i-1][j];

**else**

L[i][j] = max(L[i-1][j], v[i] + L[i-1][j - w[i]]);

}

**void** TruyVet(**int** i, **int** j) {

**if** (i==0) **return**;

**if** (j==0) **return**;

/// i > 0 va j > 0

**if** (w[i] > j) TruyVet(i-1, j);

**else** { /// w[i] <= j

**if** (L[i][j] == L[i-1][j]) TruyVet(i-1, j);

**else** {/// L[i][j] == v[i] + L[i-1][j]

TruyVet(i-1, j - w[i]);

f << "Chon do vat thu " << i << " voi trong luong " << w[i]

<< ", va gia tri la: " << v[i] << endl;

}

}

}

**void** InKQ() {

f.open(fo, ios::out);

f << L[n][W] << endl;

TruyVet(n, W);

f.close();

}

**int** main() {

Nhap();

QHD();

InKQ();

}

## 4. Một số bài toán khác

### a) Dãy con có tổng bằng S

***Đề bài***

Cho dãy a1, a2, a3, …, aN. Tìm một dãy con của dãy đó có tổng bằng S

***Công thức QHĐ***

Gọi L[i, j] = true khi có thể chọn ra được các số từ dãy a[1..i] và có tổng bằng j

L[i, j] = true khi L[i-1, j] = true hoặc L[i-1, j - a[i]] = true

L[i, j] = (L[i-1, j] == true) OR (L[i-1, j - a[i]] = true)

L[i, j] = L[i-1, j] OR (L[i-1, j - a[i]] = true)

*Bài toán con cơ bản:*

L[i, 0] = true: khi có thể chọn ra được các số từ dãy a[1..i] và có tổng bằng 0

Vậy: L[i, 0] = true

L[0, j] = true: khi có thể chọn ra được các số từ dãy a[1..0] và có tổng bằng j

L[0, j] = false, với j != 0

L[0, 0] = true

-(|a[1]| + |a[2]| + ... + |a[n]|) <= j <= (|a[1]| + |a[2]| + ... + |a[n]|)

Đặt P = |a[1]| + |a[2]| + ... + |a[n]|

Vậy: -P <= j <=P

Mảng trong thực tế (hoặc Pascal) có j chạy từ -P đến P

Mảng trong C++ chạy từ 0 trở đi: cộng thêm cho chỉ số j một lượng là P, thì: 0 -> 2\*P

***Cài đặt***

#include <fstream>

#include <string.h>

**using namespace** std;

#define fi "TONG.INP"

#define fo "TONG.OUT"

**const** **int** MAXN = 1000;

**const** **int** MAXS = 1000;

fstream f;

**int** n, S;

**int** a[MAXN];

**float** L[MAXN][MAXS];

**void** Nhap() {

f.open(fi, ios::in);

f >> n >> S;

**int** i;

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

f >> a[i];

f.close();

}

**void** QHD() {

memset(L, 0, **sizeof**(L));

**int** i, j;

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

**for** (j = 1; j <= S; j++)

**if** (a[i] > j)

L[i][j] = L[i-1][j];

**else**

L[i][j] = max(L[i-1][j], a[i] + L[i-1][j - a[i]]);

}

**void** TruyVet(**int** i, **int** j) {

**if** (i==0) **return**;

**if** (j==0) **return**;

/// i > 0 va j > 0

**if** (a[i] > j) TruyVet(i-1, j);

**else** { /// a[i] <= j

**if** (L[i][j] == L[i-1][j]) TruyVet(i-1, j);

**else** {/// L[i][j] == a[i] + L[i-1][j]

TruyVet(i-1, j - a[i]);

f << "Chon phan tu o vi tri " << i << endl;

}

}

}

**void** InKQ() {

f.open(fo, ios::out);

f << L[n][S] << endl;

TruyVet(n, S);

f.close();

}

**int** main() {

Nhap();

QHD();

InKQ();

}

### b) Chia kẹo

***Đề bài***

Cho n gói kẹo, gói thứ i có ai viên. Hãy chia các gói thành 2 phần sao cho chênh lệch giữa 2 phần là ít nhất.

***Công thức QHĐ***

Gọi T1 là số kẹo của phần 1

Gọi T2 là số kẹo của phần 2

P = T1 + T2

Không mất tính tổng quát giả sử T1 <= T2

Vậy: chênh lệnh giữa hai phần là: T2 - T1 = (P - T1) - T1 = P - 2\*T1

Chênh lệch trên nhỏ nhất khi T1 lớn nhất

Mặt khác, ta lại có: T1 <= T2 = P - T1

Vậy: 2T1 <= P

Suy ra: T1 <= P/2

***Cài đặt***

#include <bits/stdc++.h>

**using namespace** std;

**const** **int** maxn = 1000;

**int** n, P, a[1000];

bool Phan1[maxn];

bool L[maxn][2\*maxn];

**void** enter()

{

freopen("CHIAKEO.INP","r",stdin);

freopen("CHIAKEO.OUT","w",stdout);

cin >> n;

P = 0;

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

{

cin >> a[i];

P = P + a[i];

}

}

**void** Init() {

memset(Phan1, false, **sizeof**(Phan1));

}

**void** QHD()

{

**for** (**int** i=0;i<=n;i++) L[i][0] = true;

**for** (**int** j=0;j<=P;j++) L[0][j] = false;

L[0][0] = true;

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

**for** (**int** j=0;j<=P;j++)

{

L[i][j] = L[i-1][j];

**if** ((0 <= j - a[i]) && (j - a[i] <= P))

L[i][j] = L[i][j] || L[i-1][j-a[i]];

}

}

**void** TruyVet(**int** i, **int** j)

{

**if** (i==0) **return**;

**if** (j==0) **return**;

**if** ((0 <= j - a[i]) && (j - a[i] <= P))

{

**if** (L[i-1][j-a[i]])

{

cout << a[i] << " ";

Phan1[i] = true;

TruyVet(i-1, j-a[i]);

}

**else** TruyVet(i-1, j);

}

**else** TruyVet(i-1, j);

}

**void** InKQ()

{

**int** T1;

**for** (T1 = P/2; T1>=0; T1--)

**if** (L[n][T1] == true)

break;

cout << "Chenh lenh giua hai phan la: " << P - 2\*T1 << endl;

cout << T1 << " " << P - T1;

cout << "\nPhan 1: ";

TruyVet(n, T1);

cout << "\nPhan 2:";

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

**if** (Phan1[i] == false) cout << a[i] << " ";

}

**int** main()

{

enter();

Init();

QHD();

InKQ();

**return** 0;

}

### c) Market

***Đề bài***

Người đánh cá Clement bắt được n con cá, khối lượng mỗi con là ai, đem bán ngoài chợ. Ở chợ cá, người ta không mua cá theo từng con mà mua theo một **lượng** nào đó. Chẳng hạn 3kg, 5kg,…

Ví dụ: có 3 con cá, khối lượng lần lượt là 3, 2, 4. Mua lượng 6kg sẽ phải lấy con cá thứ 2 và thứ 3. Mua lượng 3kg thì lấy con cá thứ nhất. Không thể mua lượng 8kg.

Nếu bạn là người đầu tiên mua cá, có bao nhiêu lượng bạn có thể chọn?

***Cài đặt***

#include <bits/stdc++.h>

**using namespace** std;

**const** **int** maxn = 1000;

**int** n, P, a[maxn];

bool L[maxn][maxn\*2];

**void** enter()

{

freopen("market.inp","r",stdin);

freopen("market.out","w",stdout);

cin >> n;

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

{

cin >> a[i];

P = P + a[i];

}

}

**void** QHD()

{

**for** (**int** i=0;i<=n;i++) L[i][0] = true;

**for** (**int** i=0;i<=P;i++) L[0][i] = false;

L[0][0] = true;

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

**for** (**int** j=1;j<=P;j++)

{

L[i][j] = L[i-1][j];

**if** ((0 <= j - a[i]) && (j - a[i] <= P))

L[i][j] = L[i][j] || L[i-1][j-a[i]];

}

}

**void** TruyVet(**int** i, **int** j)

{

**if** (i==0) **return**;

**if** (j==0) **return**;

**if** ((-P <= j - a[i]) && (j - a[i] <= P))

{

**if** (L[i-1][j-a[i]])

{

cout << a[i] << " ";

TruyVet(i-1, j-a[i]);

}

TruyVet(i-1, j);

}

**else**

TruyVet(i-1, j);

}

**void** InKQ()

{

**int** dem = 0, k = 0;

**for** (**int** i=1;i<=P;i++)

**if** (L[n][i]) dem++;

cout << dem << "\n";

**for** (**int** S=1;S<=P;S++)

**if** (L[n][S] == true)

{

k++;

cout << "Cach " << k << " : " << "\n";

TruyVet(n, S);

cout << "\n";

}

}

**int** main()

{

enter();

QHD();

InKQ();

**return** 0;

}

### d) Điền dấu

***Đề bài***

Cho n số tự nhiên a1, a2, a3, …, aN. Ban đầu các số được đặt liên tiếp theo đúng thứ tự cách nhau bởi dấu “?”: a1?a2?a3?...?aN. Cho trước số nguyên S, có cách nào thay các dấu “?” bằng dấu “+” hay dấu “-” để được một biểu thức số học cho giá trị là S hay không?

***Công thức QHĐ***

Gọi L[i, j] = true nếu có thể điền các dấu + hoặc - vào các dấu ?

để biểu thức a1 ? a2 ? a3 ? .... ? ai-1 ? ai = j

Ngược lại thì: L[i, j] = false

Ta có: L[i, j] = true khi và chỉ khi:

xảy ra: a1 ? a2 ? a3 ? .... ? ai-1 ? ai = j

Có 2 trường hợp xảy ra:

. Điền dấu +

a1 ? a2 ? a3 ? .... ? ai-1 + ai = j

a1 ? a2 ? a3 ? .... ? ai-1 = j - ai

L[i-1, j - a[i]] = true

. Điền dấu -

a1 ? a2 ? a3 ? .... ? ai-1 - ai = j

a1 ? a2 ? a3 ? .... ? ai-1 = j + ai

L[i-1, j + a[i]] = true

Vậy: L[i, j] = true khi và chỉ khi:

(L[i-1, j - a[i]] = true) OR (L[i-1, j + a[i]] = true)

Kết quả của bài toán là: Hỏi xem L[n, S] = true

*Bài toán con cơ bản:*

L[1, a[1]]: Gọi L[i, j] = true nếu có thể điền các dấu + hoặc - vào các dấu ?

để biểu thức a1 ? a2 ? a3 ? .... ? ai-1 ? a1 = a1

Vậy: L[1, a[1]] = true

S tối đa là: a1 + a2 + a3 + .... + an = P

S tối thiểu: a1 - a2 - a3 - .... - an > - a1 - a2 - a3 - .... - an = -P

Vậy: -P <= S <= P

***Cài đặt***

#include <iostream>

#include <fstream>

#include <string.h>

#include <stdlib.h>

**using namespace** std;

#define fi "diendau.inp"

#define fo "diendau.out"

**const** **int** MAXN = 1000;

**const** **int** MAXS = 1000;

fstream f;

**int** n, S, P;

**int** a[MAXN];

bool L[MAXN][2\*MAXS];

**void** Nhap() {

f.open(fi, ios::in);

f >> n >> S;

P = 0;

**int** i;

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

f >> a[i];

P = P + a[i];

}

f.close();

}

**void** QHD() {

/// Bai toan con co ban:

**int** i, j;

///cout << P << endl;

L[1][a[1] + P] = true;

/// Cong thuc Quy hoach dong

**for** (i = 2; i <= n; i++) {

**for** (j = -P; j <= P; j++) {

L[i][j + P] = false;

**if** ((-P <= j + a[i]) && (j + a[i] <= P))

L[i][j + P] = L[i][j + P] || (L[i-1][j + a[i] + P] == true);

**if** ((-P <= j - a[i]) && (j - a[i] <= P))

L[i][j + P] = L[i][j + P] || (L[i-1][j - a[i] + P] == true);

}

}

}

**void** TruyVet(**int** i, **int** j) {

**if** (i==1) {

f << a[1];

**return**;

}

/// i > 0 va j != 0

**if** ((-P <= j + a[i]) && (j + a[i] <= P))

**if** (L[i - 1][j + a[i] + P] == true) {

TruyVet(i-1, j + a[i]);

f << " - " << a[i];

**return**;

}

**if** ((-P <= j - a[i]) && (j - a[i] <= P))

**if** (L[i - 1][j - a[i] + P] == true) {

TruyVet(i-1, j - a[i]);

f << " + " << a[i];

**return**;

}

}

**void** InKQ() {

f.open(fo, ios::out);

**if** (L[n][S + P] == false) f << "Khong co cach chon";

**else**

{

f << S << " = ";

TruyVet(n, S);

}

f.close();

}

**int** main() {

Nhap();

QHD();

InKQ();

}

# III. Biến đổi xâu

## 1. Mô hình

Cho 2 xâu X, Y. Xâu nguồn có n kí tự X1X2…Xn, xâu đích có m kí tự Y1Y2...Ym . Có 3 phép biến đổi:

* Chèn 1 kí tự vào sau kí tự thứ i: I i C
* Thay thế kí tự ở vị trí thứ i bằng kí tự C: R i C
* Xóa kí tự ở vị trí thứ i: D i

Hãy tìm số ít nhất các phép biến đổi để biến xâu X thành xâu Y

## 2. Công thức QHĐ

Gọi F[i, j] là số phép biến đổi ít nhất khi biến

xâu X1X2X3...Xi-1Xi thành xâu Y1Y2Y3Y4...Yj-1Yj

Như vậy, yêu cầu của bài toán: F[n, m]

Công thức Quy hoạch động:

X1X2X3...Xi-1Xi

Y1Y2Y3Y4...Yj-1Yj

Có 2 trường hợp xảy ra:

\* TH1: Xi = Yj

F[i, j] = F[i-1, j-1]

\* TH2: Xi # Yj

thì ta có thể lựa chọn 1 trong 3 cách biến đổi

+ Cách 1: Chèn thêm kí tự Yj vào cuối xâu X1X2X3...Xi-1Xi

X1X2X3...Xi-1XiYj

Y1Y2Y3Y4...Yj-1Yj

F[i, j] = 1 + F[i, j-1]

+ Cách 2: Thay thế kí tự Xi thành kí tự Yj

X1X2X3...Xi-1Yj

Y1Y2Y3Y4...Yj-1Yj

F[i, j] = 1 + F[i-1, j-1]

+ Cách 3: Xóa kí tự Xi

X1X2X3...Xi-1

Y1Y2Y3Y4...Yj-1Yj

F[i, j] = 1 + F[i-1, j]

F[i, j] = min(1 + F[i, j-1], 1 + F[i-1, j-1], 1 + F[i-1, j])

*Bài toán con cơ bản:*

F[i, 0]: là số phép biến đổi ít nhất khi biến xâu X1X2X3...Xi-1Xi thành xâu rỗng

Vậy F[i, 0] = i

F[0, j]: là số phép biến đổi ít nhất khi biến xâu rỗng thành xâu Y1Y2Y3Y4...Yj-1Yj

Vậy F[0, j] = j

## 3. Cài đặt

#include <bits/stdc++.h>

**using namespace** std;

#define fi "BIENDOIXAU.INP"

#define fo "BIENDOIXAU.OUT"

**const** **int** maxn = 1001;

string X, Y;

**int** n, m;

**int** f[maxn][maxn];

**void** enter()

{

freopen(fi,"r",stdin);

freopen(fo,"w",stdout);

cin >> X;

cin >> Y;

n = X.length();

m = Y.length();

X = "." + X;

Y = "." + Y;

}

**int** min3(**int** x, **int** y, **int** z)

{

**return** min(x, min(y, z));

}

**void** QHD()

{

**for** (**int** i = 0; i <= n; i++) f[i][0] = i;

**for** (**int** j = 0; j <= m; j++) f[0][j] = j;

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

**for** (**int** j = 1; j <= m; j++)

{

**if** (X[i] == Y[j]) f[i][j] = f[i-1][j-1];

**else** f[i][j] = min3(f[i][j-1], f[i-1][j-1], f[i-1][j]) + 1;

}

}

**void** TruyVet(**int** i, **int** j) {

**if** (i == 0)

{

**for** (**int** k = 1; k <= j; k++)

{

cout << "Chen ki tu " << Y[k] <<

" vao sau ki tu tai vi tri thu " <<

k-1 << " cua xau X ban dau" << endl;

}

**return**;

}

**if** (j == 0)

{

**for** (**int** k = i; k >= 1; k--)

{

cout << "Xoa ki tu " << X[k] << " tai vi tri thu " << k

<< " cua xau X ban dau" << endl;

}

**return**;

}

**if** (X[i] == Y[j]) TruyVet(i-1, j-1);

**else** {

**if** (f[i][j] == 1 + f[i-1][j]) {

cout << "Xoa ki tu " << X[i] << " tai vi tri thu " << i << " cua xau X ban dau" << endl;

TruyVet(i - 1, j);

}

**else** **if** (f[i][j] == 1 + f[i][j-1])

{

cout << "Chen ki tu " << Y[j] <<

" vao sau ki tu " << X[i] << " tai vi tri thu " <<

i << " cua xau X ban dau" << endl;

TruyVet(i, j - 1);

}

**else** {

cout << "Thay the ki tu " << X[i] << " tai vi tri thu " << i

<< " cua xau X ban dau thanh ki tu " << Y[j] << endl;

TruyVet(i-1, j-1);

}

}

}

**void** InKQ()

{

cout << f[n][m] << endl;

TruyVet(n, m);

}

**int** main()

{

enter();

QHD();

InKQ();

**return** 0;

}

## 4. Một số bài toán khác

### a) Xâu con chung dài nhất

***Đề bài***

Cho 2 xâu X, Y. Hãy tìm xâu con của X và của Y có độ dài lớn nhất

***Công thức QHĐ***

Gọi F[i][j] là độ dài xâu chung dài nhất của X1X2...Xi và Y1Y2...Yj

Kết quả bài toán là F[n][m]

TH1 : X[i] == Y[j]

F[i][j] = 1 + F[i-1][j-1]

TH2 : X[i] # Y[j]

F[i][j] = max(F[i][j-1], F[i-1][j])

*Bài toán con cơ bản*

F[0][j] là độ dài xâu chung dài nhất của xâu rỗng và Y1Y2...Yj

Vậy F[0][j] = 0

F[i][0] là độ dài xâu chung dài nhất của X1X2...Xi và xâu rỗng

Vậy F[i][0] = 0

***Cài đặt***

#include <bits/stdc++.h>

**using namespace** std;

#define fi "XAUCONCHUNG.INP"

#define fo "XAUCONCHUNG.OUT"

**const** **int** maxn = 1001;

string X, Y;

**int** n, m;

**int** F[maxn][maxn];

**void** enter()

{

freopen(fi,"r",stdin);

freopen(fo,"w",stdout);

cin >> X;

cin >> Y;

n = X.length();

m = Y.length();

X = "." + X;

Y = "." + Y;

}

**void** QHD()

{

**for** (**int** i = 0; i <= n; i++) F[i][0] = 0;

**for** (**int** j = 0; j <= m; j++) F[0][j] = 0;

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

**for** (**int** j = 1; j <= m; j++)

{

**if** (X[i] == Y[j]) F[i][j] = 1 + F[i-1][j-1];

**else** F[i][j] = max(F[i][j-1], F[i-1][j]);

}

}

**void** TruyVet(**int** i, **int** j)

{

**if** (i == 0) **return**;

**if** (j == 0) **return**;

**if** (X[i] == Y[j])

{

TruyVet(i-1, j-1);

cout << X[i];

}

**else**

{

**if** (F[i][j] == F[i-1][j])

TruyVet(i-1, j);

**else**

TruyVet(i, j-1);

}

}

**void** InKQ()

{

cout << F[n][m] << endl;

TruyVet(n, m);

}

**int** main()

{

enter();

QHD();

InKQ();

**return** 0;

}

### b) Bắc cầu

***Đề bài***

Hai nước Anpha và Beta nằm ở hai bên bờ sông Omega, Anpha nằm ở bờ bắc và có M thành phố được đánh số từ 1 đến m, Beta nằm ở bờ nam và có N thành phố được đánh số từ 1 đến n (theo vị trí từ đông sang tây). Mỗi thành phố của nước này thường có quan hệ kết nghĩa với một số thành phố của nước kia. Để tăng cường tình hữu nghị, hai nước muốn xây các cây cầu bắc qua sông, mỗi cây cầu sẽ là nhịp cầu nối 2 thành phố kết nghĩa. Với yêu cầu là các cây cầu không được cắt nhau và mỗi thành phố chỉ là đầu cầu cho nhiều nhất 1 cây cầu, hãy chỉ ra cách bắc cầu được nhiều cầu nhất.

***Công thức QHĐ***

Gọi F[i, j] là số cây cầu nhiều nhất xây được khi xét các thành phố A[1..i] và B[1..j]

Có 2 trường hợp xảy ra:

\* TH1: Ai có quan hệ kết nghĩa với Bj

F[i, j] = 1 + F[i-1, j-1]

\* TH2: Ai không có quan hệ kết nghĩa với Bj

F[i, j] = max(F[i, j-1], F[i-1, j])

*Bài toán con cơ bản:*

F[0, j]: là số cây cầu nhiều nhất xây được khi xét các thành phố A[1..0] và B[1..j]

Vậy F[0, j] = 0

F[i, 0]: là số cây cầu nhiều nhất xây được khi xét các thành phố A[1..i] và B[1..0]

Vậy F[i, 0] = 0

***Cài đặt***

#include <bits/stdc++.h>

**using namespace** std;

#define fi "baccau.inp"

#define fo "baccau.out"

**const** **int** maxn = 1000;

**int** n, m, k;

bool connected[maxn][maxn];

**int** F[maxn][maxn];

**void** enter()

{

freopen(fi,"r",stdin);

freopen(fo,"w",stdout);

cin >> n >> m >> k;

memset(connected, false, **sizeof**(connected));

**for** (**int** i = 1; i <= k; i++)

{

**int** u, v;

cin >> u >> v;

connected[u][v] = true;

}

}

**void** solve()

{

**for** (**int** i = 0; i <= n; i++) F[i][0] = 0;

**for** (**int** j = 0; j <= m; j++) F[0][j] = 0;

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

**for** (**int** j = 1; j <= m; j++)

{

**if** (connected[i][j] == true)

F[i][j] = 1 + F[i-1][j-1];

**else**

F[i][j] = max(F[i-1][j], F[i][j-1]);

}

}

**void** TruyVet(**int** i, **int** j)

{

**if** (i == 0) **return**;

**if** (j == 0) **return**;

**if** (connected[i][j] == true)

{

TruyVet(i-1, j-1);

cout << "Xay dung cau giau thanh pho A" << i << " va thanh pho B" << j << endl;

}

**else**

{

**if** (F[i][j] == F[i][j-1])

TruyVet(i, j-1);

**else**

TruyVet(i-1, j);

}

}

**void** InKQ()

{

cout << F[n][m] << endl;

TruyVet(n, m);

}

**int** main()

{

enter();

solve();

InKQ();

**return** 0;

}

### c) Palindrom

***Đề bài***

Một xâu được gọi là xâu đối xứng (palindrom) nếu xâu đó đọc từ trái sang phải hay đọc từ phải sang trái đều như nhau. Cho một xâu S, hãy tìm số kí tự ít nhất cần thêm vào S để S trở thành xâu đối xứng

***Công thức QHĐ***

Gọi F[i, j] là số lượng kí tự ít nhất cần thêm vào để xâu S[i..j] là đối xứng.

SjSiSi+1Si+2...Sj-1Sj

Như vậy, có 2 trường hợp xảy ra:

\* TH1: S[i] = S[j]

F[i, j] = F[i+1, j-1]

\* TH2: S[i] <> S[j]

F[i, j] = min(1 + F[i, j-1], 1 + F[i+1, j])

*Bài toán con cơ bản*:

F[i, 0]: là số lượng kí tự ít nhất cần thêm vào để xâu S[i..0] là đối xứng: vô lý

F[0, j]: là số lượng kí tự ít nhất cần thêm vào để xâu S[0..j] là đối xứng: không tính được

***Cài đặt***

#include <bits/stdc++.h>

**using namespace** std;

#define fi "palindrome.inp"

#define fo "palindrome.out"

string s;

**const** **int** maxn = 1000;

**int** n, F[maxn][maxn];

**void** enter()

{

freopen(fi, "r", stdin);

freopen(fo, "w", stdout);

cin >> s;

n = s.length();

s = '.' + s;

}

**void** QHD()

{

**for** (**int** i = 0; i <= n; i++) F[i][i] = 0;

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

**if** (s[i] == s[i+1])

F[i][i+1] = 0;

**else** F[i][i+1] = 1;

**for** (**int** k = 2; k <= n - 1; k++)

**for** (**int** i = 1; i <= n - k; i++)

{

**int** j = i + k;

**if** (s[i] == s[j])

F[i][j] = F[i+1][j-1];

**else**

F[i][j] = min(1 + F[i+1][j], 1 + F[i][j-1]);

}

}

**void** TruyVet(**int** i, **int** j)

{

**if** (i >= j) **return**;

**if** (s[i] == s[j])

TruyVet(i+1, j-1);

**else**

{

**if** (F[i][j] == 1 + F[i+1][j])

{

cout << "Dien ki tu " << s[i] << " vao sau ki tu " << s[j] << " tai vi tri thu " << j

<< " cua xau X ban dau" << endl;

TruyVet(i+1, j);

}

**else**

{

cout << "Dien ki tu " << s[j] << " vao truoc ki tu " << s[i] << " tai vi tri thu " <<

i << " cua xau X ban dau" << endl;

TruyVet(i, j-1);

}

}

}

**void** InKQ()

{

cout << F[1][n] << endl;

TruyVet(1, n);

}

**int** main()

{

enter();

QHD();

InKQ();

**return** 0;

}

# IV. Vali (A)

## 1. Mô hình

Chọn n vật, vật i nặng ai và có giá trị bi. Hãy chọn ra một số vật để cho vào balo sao cho tổng khối lượng không vượt quá W và tổng giá trị là lớn nhất. Chú ý rằng mỗi vật có thể được chọn nhiều lần

## 2. Công thức QHĐ

Gọi L[i, j] là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..i sao cho tổng trọng lượng không vượt quá j

Vậy: yêu cầu của bài toán là: L[n, W]

Có 2 khả năng xảy ra:

\* Chọn đồ vật thứ i: W[i] <= j

L[i, j] = V[i] + L[i, j - W[i]]

\* Không chọn đồ vật thứ i

L[i, j] = L[i-1, j]

L[i, j] = max(V[i] + L[i, j - W[i]], L[i-1, j])

*Bài toán con cơ bản:*

Gọi L[0, j] là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..0 sao cho tổng trọng lượng không vượt quá j

Vậy L[0, j] = 0

Gọi L[i, 0] là tổng giá trị lớn nhất khi chọn các đồ vật từ 1..i sao cho tổng trọng lượng không vượt quá 0

Vậy L[i,0] = 0

## 3. Cài đặt

#include <fstream>

#include <string.h>

**using namespace** std;

#define fi "VALI.INP"

#define fo "VALI.OUT"

**const** **int** MAXN = 1000;

**const** **int** MAXW = 1000;

fstream f;

**int** n, W;

**int** w[MAXN];

**float** v[MAXN];

**float** L[MAXN][MAXW];

**void** Nhap() {

f.open(fi, ios::in);

f >> n >> W;

**int** i;

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

f >> w[i] >> v[i];

f.close();

}

**void** QHD() {

memset(L, 0, **sizeof**(L)); /// Bai toan con co ban

**int** i, j;

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

**for** (j = 1; j <= W; j++)

**if** (w[i] > j)

L[i][j] = L[i-1][j];

**else** /// w[i] <= v[i]

L[i][j] = max(L[i-1][j], v[i] + L[i][j - w[i]]);

}

**void** TruyVet(**int** i, **int** j) {

**if** (i==0) **return**;

**if** (j==0) **return**;

/// i > 0 va j > 0

**if** (w[i] > j) TruyVet(i-1, j);

**else** { /// w[i] <= j

**if** (L[i][j] == L[i-1][j]) TruyVet(i-1, j);

**else** {/// L[i][j] == v[i] + L[i][j]

TruyVet(i, j - w[i]);

f << "Chon do vat thu " << i << " voi trong luong " << w[i]

<< ", va gia tri la: " << v[i] << endl;

}

}

}

**void** InKQ() {

f.open(fo, ios::out);

f << L[n][W] << endl;

TruyVet(n, W);

f.close();

}

**int** main() {

Nhap();

QHD();

InKQ();

}

## 4. Một số bài toán khác

### a) Farmer

***Đề bài***

Một người có N mảnh đất và M dải đất. Các mảnh đất có thể coi là một tứ giác và các dải đất thì coi như một đường thẳng. Dọc theo các dải đất ông ta trồng các cây bách, dải đất thứ i có ai cây bách. Ông ta cũng trồng các cây bách trên viền của các mảnh đất, mảnh đất thứ j có bj cây bách. Cả ở trên các mảnh đất và dải đất, xen giữa hai cây bách ông ta trồng một cây oliu. Ông ta cho con trai được chọn các mảnh đất và dải đất tùy ý với điều kiện tổng số cây bách không vượt quá Q. Người con trai phải chọn thế nào để có nhiều cây oliu (loài cây mà anh ta thích) nhất.

***Cài đặt***

#include <fstream>

#include <string.h>

**using namespace** std;

#define fi "FARMER.INP"

#define fo "FARMER.OUT"

**const** **int** MAXN = 1000;

**const** **int** MAXW = 1000;

fstream f;

**int** n, m, W;

**int** w[MAXN];

**int** v[MAXN];

**int** L[MAXN][MAXW];

**void** Nhap() {

f.open(fi, ios::in);

f >> n >> m >> W;

**int** i;

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

{

f >> w[i];

v[i] = w[i] - 1;

}

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

{

f >> w[i];

v[i] = w[i];

}

f.close();

}

**void** QHD() {

memset(L, 0, **sizeof**(L));

**int** i, j;

**for** (i = 1; i <= n + m; i++)

**for** (j = 1; j <= W; j++)

**if** (w[i] > j)

L[i][j] = L[i-1][j];

**else**

L[i][j] = max(L[i-1][j], v[i] + L[i-1][j - w[i]]);

}

**void** TruyVet(**int** i, **int** j) {

**if** (i==0) **return**;

**if** (j==0) **return**;

/// i > 0 va j > 0

**if** (w[i] > j) TruyVet(i-1, j);

**else** { /// w[i] <= j

**if** (L[i][j] == L[i-1][j]) TruyVet(i-1, j);

**else** {/// L[i][j] == v[i] + L[i-1][j]

TruyVet(i-1, j - w[i]);

**if** (i <= n)

{

f << "Chon dai dat thu " << i << " voi so cay bach " << w[i]

<< ", va so cay oliu la : " << v[i] << endl;

}

**else**

{

f << "Chon manh dat thu " << i - n << " voi so cay bach " << w[i]

<< ", va so cay oliu la : " << v[i] << endl;

}

}

}

}

**void** InKQ() {

f.open(fo, ios::out);

f << L[n+m][W] << endl;

TruyVet(n+m, W);

f.close();

}

**int** main() {

Nhap();

QHD();

InKQ();

}

### b) Đổi tiền

***Đề bài***

Ở đất nước Omega người ta chỉ tiêu tiền xu. Có N loại tiền xu, loại thứ i có mệnh giá là ai đồng. Một người khách du lịch đến Omega du lịch với số tiền M đồng. Ông ta muốn đổi số tiền đó ra tiền xu Omega để tiện tiêu dùng. Ông ta cũng muốn số tiền đổi được là ít nhất (cho túi tiền đỡ nặng khi đi đây đi đó). Bạn hãy giúp ông ta đổi tiền.

***Công thức QHĐ***

Gọi L[i, j] là số lượng đồng tiền ít nhất khi đổi j đồng với cách chọn các mệnh giá từ 1..i

\* Chọn mệnh giá thứ i: W[i] <= j

L[i, j] = 1 + L[i, j - W[i]]

\* Không chọn mệnh giá thứ i

L[i, j] = L[i-1, j]

Bài toán con cơ bản

Gọi L[0, j] là số lượng đồng tiền ít nhất khi đổi j đồng với cách chọn các mệnh giá từ 1..0

Vậy L[0, j]: Không đổi được, suy ra gán = +oo với j > 0

L[0, 0] = 0

Gọi L[i, 0] là số lượng đồng tiền ít nhất khi đổi 0 đồng với cách chọn các mệnh giá từ 1..i

Vậy L[i, 0] = 0

***Cài đặt***

#include <fstream>

#include <string.h>

**using namespace** std;

#define fi "OMEGA.INP"

#define fo "OMEGA.OUT"

**const** **int** MAXN = 10;

**const** **int** MAXW = 2000000;

fstream f;

**int** n, W;

**int** w[MAXN];

**int** L[MAXN][MAXW];

**const** **int** oo = 2000000;

**void** Nhap() {

f.open(fi, ios::in);

f >> n >> W;

**int** i;

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

f >> w[i];

f.close();

}

**void** QHD() {

/// Bai toan con co ban

**for** (**int** j = 1; j <= W; j++) L[0][j] = +oo;

**for** (**int** i = 0; i <= n; i++) L[i][0] = 0;

**int** i, j;

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

**for** (j = 1; j <= W; j++)

**if** (w[i] > j)

L[i][j] = L[i-1][j];

**else** /// w[i] <= v[i]

L[i][j] = min(L[i-1][j], 1 + L[i][j - w[i]]);

}

**void** TruyVet(**int** i, **int** j) {

**if** (i==0) **return**;

**if** (j==0) **return**;

/// i > 0 va j > 0

**if** (w[i] > j) TruyVet(i-1, j);

**else** { /// w[i] <= j

**if** (L[i][j] == L[i-1][j]) TruyVet(i-1, j);

**else** {/// L[i][j] == 1 + L[i][j]

TruyVet(i, j - w[i]);

f << "Chon do dong tien thu " << i << " voi menh gia " << w[i] << endl;

}

}

}

**void** InKQ() {

f.open(fo, ios::out);

f << L[n][W] << endl;

TruyVet(n, W);

f.close();

}

**int** main() {

Nhap();

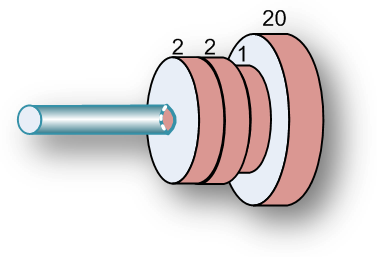
QHD();

InKQ();

}

# V. Một số bài tập nâng cao

## 1. Bài toán Cử tạ

Rèn luyện thể lực bằng cách tập nâng tạ thu hút được sự chú ý của rất nhiều bạn trẻ. Tạ là một thanh trục có gắn ở hai đầu các đĩa tạ. Bộ đĩa tạ trong phòng tập bao gồm các loại 1kg, 2kg, 5kg, 10kg, 15kg và 20kg với số lượng mỗi loại là đủ nhiều. Các đĩa tạ ở hai đầu thanh được gắn đối xứng để đảm bảo thanh tạ được cân. Mỗi người, tùy theo thể lực của mình, lắp các đĩa tạ để có trọng lượng phù hợp. Để điều chỉnh trọng lượng, người ta tháo các đĩa ngoài cùng, lắp các đĩa mới vào. Do tính đối xứng của thanh tạ, ta chỉ xét các thao tác điều chỉnh ở một đầu.

Hiện tại ở một đầu đang có ***n*** đĩa tạ gắn vào trục (1 ≤ ***n*** ≤ 10), tính từ trong ra ngoài đĩa thứ ***i*** có trọng lượng ***pi***. Bạn cần có thanh tạ với trọng lượng một đầu là ***w*** (0 ≤ ***w*** ≤ 100).

Ví dụ, hiện tại ***n*** = 4 và các đĩa tạ là (2, 2, 1, 20), bạn cần điều chỉ trọng lượng thành 14kg. Bạn sẽ phải thực hiện 3 thao tác tháo lắp: tháo đĩa 20kg, tháo đĩa 1kg và lắp đĩa 10kg.

**Yêu cầu:** Cho ***n***, ***pi***, ***i*** = 1 ÷ ***n***, ***w***. Hãy xác định số thao tác ít nhất cần thực hiện.

**Dữ liệu vào:** Cho trong file văn bản DUMBBELL.INP có cấu trúc như sau

*- Dòng 1*: Ghi số nguyên ***n***.

*- Dòng 2*: Ghi n số nguyên ***p1***, ***p2***, . . ., ***pn***, các số được ghi cách nhau ít nhất một dấu cách.

*- Dòng 3*: Ghi số nguyên *w*.

**Dữ liệu ra:** Ghi ra file văn bản DUMBBELL.OUT theo cấu trúc như sau:

*- Dòng 1:* Ghi số thao tác cần thực hiện.

**Ví dụ:**

|  |  |
| --- | --- |
| DUMBBELL.INP | DUMBBELL.OUT |
| 4  2 2 1 20  14 | 3 |

**Thuật toán:**

Ta tạo ra bảng phương án B, B*i* xác định số lần lắp đĩa tối thiểu để làm tăng trọng lượng lên *i* kg.

int b[100]

Việc xác định Bi (i = 0 ÷ 100) khá đơn giản:

**t := i / 20; j := i % 20;**

**t := t + j /15; j := j % 15;**

**t := t+j / 10; j := j % 10;**

**t:= t+j / 5; j := j % 5;**

**t:= t+j / 2; j := j % 2;**

**B[i]:= t+j ;**

Nếu sử dụng mảng hằng C với C*i* là số lần lắp đĩa tối thiểu để làm tăng trọng lượng lên i kg (i = 0 ÷ 19).

**int C[20]= (0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 1, 2, 2, 3, 3, 1, 2, 2, 3, 3);**

Việc tính Bi (i:= 0 ÷ 100) lúc này chỉ cần sử dụng vòng lặp:

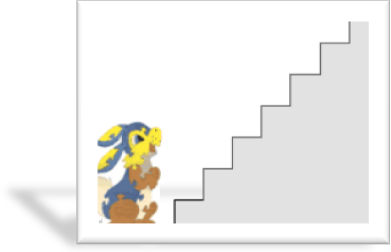
**For (int i=0;i<=9;i++) do B[i]:=C[i];**

**For (int i= 20;i<=100;i++) B[i]:= i / 20 + B[i % 20];**

Lời giải của bài toán có thể nhận được bằng cách duyệt tất cả các cách tháo lần lượt đĩa tạ n, n-1, n-2, . . ., 2, 1.

Bảng phương án còn là công cụ sắc bén với các loại bài toán liên quan tới phát hiện, nhận dạng chu trình. Nó giúp ta đạt được hiệu quả O(n) và trong nhiều trường hợp – O(1)!

## 2. Bài toán Thỏ nhặt cà rốt

Các con thú nuôi trong chuồng ở vườn bách thú thường ít có điều kiện vận động. Điều này vừa có hại cho sức khỏe của thú nuôi, vừa làm làm giảm hứng thú của khách tham quan. Để khắc phục điều đó, Ban giám đốc cho đặt một cái thang có ***n*** bậc trong chuồng thỏ. Đến giờ cho ăn người ta đặt cà rốt - thứ khoái khẩu nhất của thỏ, lên bậc trên cùng của thang. Thỏ phải nhảy theo các bậc thang để lấy cà rốt. Mỗi bước nhảy thỏ có thể vượt được ***k*** bậc (1 ≤ ***k*** ≤ ***n*** ≤ 300). Có thể có nhiều cách nhảy để lấy cà rốt. Hai cách nhảy gọi là khác nhau nếu tồn tại một bậc thỏ tới được ở một cách nhảy và bị bỏ qua ở cách kia. Ví dụ, với ***n*** = 4 và ***k*** = 3 có tất cả 7 cách lấy cà rốt khác nhau: 1+1+1+1, 1+1+2, 1+2+1, 2+1+1, 2+2, 1+3, 3+1.

***Yêu cầu***: Cho ***k*** và ***n***. Hãy xác định số cách khác nhau thỏ có thể thực hiện để lấy cà rốt.

***Dữ liệu vào***: Ghi trong file văn bản CARROT.INP có cấu trúc như sau:

*- Dòng 1:* Ghi số nguyên dương *t* là số lượng cặp ***k*** và ***n*** (1 ≤ ***t*** ≤ 50).

*- Dòng thứ i trong t dòng tiếp theo:* Mỗi dòng ghi 2 số nguyên ***k*** và ***n***.

***Dữ liệu ra:***: Ghi ra file văn bản CARROT.OUT, kết quả mỗi test đưa ra trên một dòng dưới dạng số nguyên.

***Ví dụ***:

|  |  |  |
| --- | --- | --- |
| CARROT.INP |  | CARROT.OUT |
| **3**  **1 3**  **2 7**  **3 10** |  | **1**  **21**  **274** |

***Thuật toán:***

Đây là bài toán áp dụng sơ đồ tính lặp để tích lũy kết quả. Loại sơ đồ này có bản chất rất gần với giải thuật quy hoạch động nên nhiều khi người ta cũng gộp nó vào bài toán có thuật giải quy hoạch động.

Ta có thuật toán như sau:

Gọi *fi* là số cách mà thỏ có thể nhảy tới bậc thứ i của thang,

Công thức lặp (cách tính *fi*): *fi* = 

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

{

For (int j= i - k ; j<=i-1; j++)

{

F[i] = F[i] + F[j];

}

}

Giá trị đầu: Cần có k giá trị ban đầu để triển khai công thức lặp. Có thể chọn một trong hai cách:

Tính riêng *fi* (i = 1 ÷ k) theo công thức *f0*=1, *fi* = 

F[0] = 1;

For (int i=1; i<=k; i++)

{

For (int j=1; j<=i-1 ; j++)

{

F[i] = F[i] + F[j];

}

}

Cho *f0* = 1, *fi* = 0, i = -k+1 ÷ -1,

# C. LỜI KẾT

Quy hoạch động là một thuật toán nhìn chung là có tư duy rõ ràng và áp dụng cho tương đối nhiều bài toán. Chính vì vậy việc rèn tư duy cho học sinh khối chuyên có thể áp dụng được trong khi lập trình cho các bài toán đặc trưng là rất quan trọng vầ cần thiết. Trên đây là một số vấn đề cần dạy các tư duy và tiếp cận cho học sinh chuyên khi bắt đầu học về quy hoạch động. Chắc chắn chưa đầy đủ, rất mong các Thày cô góp ý cho chuyên đề được hoàn thiện tốt hơn.