Một số bài toán QHĐ cơ bản

Bài 1. Tam giác số (Triangle)

Cho tam giác số N dòng dạng như hình vẽ. Tại mỗi ô chỉ có thể đi xuống 1 trong 2 ô phía dưới nó. Hãy tìm một đường đi từ đỉnh xuống đáy tam giác sao tổng các số trên đường đi là lớn nhất.

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

Dữ liệu vào : File TRIANGLE.INP

- Dòng đầu là số n – Số dòng của tam giác ($n \leq 1000$)
- Dòng thứ i trong số n dòng tiếp theo mỗi dòng chứa i số nguyên (có giá trị tuyệt đối không quá 10^6) là các số trên dòng thứ i của tam giác.

Kết quả ra : File TRIANGLE.OUT

- Dòng đầu ghi một nguyên là tổng các số trên đường đi tìm được.
- Dòng thứ hai ghi n số t_i – là chỉ số của số trên đường đi tại dòng thứ i ($1 \leq i \leq n$).

Ví dụ:

TRIANGLE.INP	TRIANGLE.OUT
5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5	30 1 1 1 2 2 *Giải thích:* 7—3 —8—7—5

Thuật toán: Xét tọa độ hàng cột, biểu diễn như trong file input. Gọi $A[i][j]$ chính là giá trị ô dòng thứ i, cột thứ j của tam giác số.

Ta thấy để đi tới được ô (i, j), ô trước đó phải tới là một trong hai ô hàng trên $(i - 1, j)$ và $(i - 1, j - 1)$.

Gọi $F[i][j]$ là giá trị đường đi lớn nhất ta có thể xây dựng khi đi từ ô (1,1) tới ô (i, j).

Dễ thấy, nếu đường đi này qua ô $(i - 1, j)$ thì đường đi từ ô $(1, 1)$ tới ô $(i - 1, j)$ cũng phải là đường có tổng các số là lớn nhất. Với trường hợp này:

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

Tương tự, nếu đường đi đi qua ô $(i - 1, j - 1)$ thì: $F[i][j] = F[i-1][j-1] + a[i][j]$

Vì vậy:

$$F[i][j] = a[i][j] + \boldsymbol{MAX}(F[i-1][j], F[i-1][j-1])$$

7
10 15
18 16 15
20 25 20 19
24 30 27 26 24

Vì ta cần tìm đường đi lớn nhất từ đỉnh và kết thúc tại đáy của tam giác nên kết quả bài toán sẽ là giá trị lớn nhất trong các $F[n][1], F[n][2], \dots, F[n][n]$, *hay:*

$$\boldsymbol{MAX}\ (F[n][i]) \text{ với } i = 1, 2, \dots, N$$

Bảng giá trị F của dữ liệu đề bài nhận giá trị như bảng bên.

Chương trình được thực hiện như sau:

```
#define maxc 1000000001
void chuanbi()
{   // Cơ sở QHĐ
  for (int i = 0; i <= n; i++)   d[i][0] = d[i][j+1] = -maxc;
  d[1][1] = a[1][1];
}
void xuli()
{
    // QHD
    for (int i = 2; i <= n; i++)
        for (int j = 1; j <= i; j++)
                d[i][j] = a[i][j] + max(d[i-1][j-1],d[i-1][j]);
    // Tim diem ket thuc tren duong di lon nhat
    int vtmax = 1;
    for (int j = 2; j <= n; i++)
        if (d[n][vtmax] < d[n][j]) vtmax = j;
    kq = d[n][vtmax];
    //truy vet
    for (int i = n; i >=1 ; i--)
    {
        luu[i] = vtmax;
        if (d[i-1][vtmax-1] > d[i-1][vtmax]) vtmax--;
       // chon 1 trong 2 vi tri o dong tren
    }
}

void ghikq()
{
    cout << kq << endl;
    for (int i = 1; i <= n; i++)
                cout << luu[i] << " ";
}
```

Bài 2. Di chuyển từ Tây sang Đông (WTOE)

Cho ma trận $M * N$ mỗi ô chứa một số nguyên ta cần di chuyển từ một ô bất kì thuộc cột bên trái sang một ô bất kì thuộc cột bên phải. Mỗi bước di chuyển từ một ô (i, j) ta có thể đi sang ô $(i-1, j+1)$ hoặc $(i, j+1)$ hoặc $(i+1, j+1)$. Chi phí cho một đường đi là tổng của các số nguyên trên con đường đó. Yêu cầu hãy tìm ra một con đường đi với chi phí thấp nhất.

Dữ liệu vào : File TAYDONG.INP

- Dòng đầu là số m, n $(m, n \leq 100)$
- Dòng thứ *i* trong số *m* dòng tiếp theo ghi n số $A_{i1}, A_{i2}, \dots, A_{in}$ $(|A_{ij}| \leq 10^5)$

Kết quả ra : File TAYDONG.OUT

- Dòng đầu ghi một nguyên là chi phí ***min*** tìm được.
- Dòng thứ hai ghi *N* chỉ số của hàng lần lượt di chuyển từ Tây sang Đông.

TAYDONG.INP	TAYDONG.OUT
4 4 2 4 3 1 8 5 2 9 1 7 4 6 1 7 8 9	9 1 1 2 1 *Giải thích:* 2 – 4 - 2 - 1

Hướng dẫn:

Gọi F[i,j] là giá trị tổng các số trên các ô đi qua theo con đường tốt nhất từ ô (i,j) tới một ô thuộc cột n.

Ta có: $\boldsymbol{F[i][n] = A[i][n]} \quad \forall i\ 1, 2, \dots, n$

Các ô còn lại, lần lượt từ cột n-1 về cột 1 được xây dựng theo công thức truy hồi sau:

$$\boldsymbol{F[i][j] = MIN\,(F[i-1][j+1],\ F[i][j+1],\ F[i+1][j+1]) + A[i][j]}$$

Ngoài ra, dùng mảng hai chiều $Trace[1..M, 1..N]$ để ghi lại chỉ số dòng của ô thuộc cột $j+1$ mà ô (i, j) đi tới để đạt giá trị tốt nhất.

```
void chuanbi()
{
  // Rào hai hàng trên cùng và dưới cùng
   for (int i = 0; i <= n; i++)
        d[0][i] = d[m+1][i] = maxc;
  // Nếu ở cột cuối thì giá trị chính là a[i][n]
   for (int i = 1; i <= m; i++) d[i][n] = a[i][n];
}

void QHĐ()
{
   // đi từng cột từ phải sang trái ( Đông về Tây)
   for (int j = n-1; j >= 1; j--)
       for (int i = 1; i <= m; i++)// tính d[i][j]
       {
           int vtmin = i;
           if (d[vtmin][j+1] > d[i-1][j+1]) vtmin = i-1;
           if (d[vtmin][j+1] > d[i+1][j+1]) vtmin = i+1;
           d[i][j] = d[vtmin][j+1] + a[i][j];
           trace[i][j] = vtmin; // lưu vết
       }
}
```

```
void ghikq()
{
   int vtmin = 1;
   for (int i = 1; i <= m; i++)
      if (d[vtmin][1] > d[i][1]) vtmin = i;
      cout << d[vtmin][1] << endl;
   for (int j = 1; j <= n; j++)
   {
       cout << vtmin << " ";
       vtmin = trace[vtmin][j];
   }
}
```

Bài 3. Dãy con tăng dài nhất (LIS - Longest Increasing Subsequence)

Cho dãy số nguyên n phần tử. Hãy tìm dãy con tăng chặt dài nhất của dãy đã cho (các phần tử của dãy con có thể không liên tiếp nhau).

Dữ liệu: Vào từ file **DCT.INP**

- Dòng đầu tiên ghi số n $(1 \leq n \leq 10^5)$
- Dòng thứ 2 ghi n số $A_1, A_2, \dots, A_n$ $(|A_i| \leq 10^5)$

Kết quả: Ghi ra file **DCT.OUT**

DCT.INP	DCT.OUT
6 1 5 2 8 3 7	4 1 2 3 7

- Dòng đầu tiên ghi m là số lượng phần tử của dãy con tăng dài nhất tìm được.
- Dòng thứ hai ghi m phần tử của dãy con tăng tìm được.

Thuật toán:
Gọi $L[i]$ là độ dài dãy con tăng dài nhất với phần tử cuối cùng là A_i (A_i là đuôi của dãy con).
Dãy này được thành lập bằng việc lấy A_i ghép một trong số các dãy con đơn điệu tăng dài nhất kết thúc tại vị trí A_j đứng trước A_i. Khi đó:

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

Ta sẽ chọn dãy nào để ghép A_i vào đuôi? Tất nhiên ta sẽ chỉ ghép được A_i vào đuôi những dãy kết thúc tại A_j nào đó nhỏ hơn A_i (để đảm bảo tính tăng). Và chắc chắn ta sẽ chọn dãy dài nhất như vậy để đảm bảo tính dài nhất.
Vậy $L[i]$ được tính như sau:
Xét tất cả các chỉ số j trong khoảng từ 1 tới $i-1$ mà $A_j < A_i$, chọn ra chỉ số $jmax$ có L[j_{max}] lớn nhất.
$L[i] = L[jmax] + 1$. Trong trường hợp đặc biệt, không tồn tại giá trị j nào thỏa mãn, $L[i] = 1$. Ở trường hợp này, ta đặt $jmax = 0, L[0] = 0$.
Ta nên bổ sung phần tử $A_0 = -\infty$, $A_{n+1} = +\infty$, nối vào đầu và đuôi dãy con ta tìm được.
Kết quả bài toán là : $\boldsymbol{L[n+1] - 1}$ (ta bỏ đi phần tử cuối cùng A_{n+1}, phần tử A_0 không phải trừ đi vì $L[0] = 0$).
Ta dùng thêm một mảng $Trace$[] với $Trace[i] = jmax$.

```
void xuli_xuoi(){
a[0] = -maxc;  // -oo
a[n+1] = maxc; // +oo
l[0] = 0; // cơ sở QHĐ
FOR(i, 1, n+1){
   int jmax = 0;
   FOR(j, 1, i-1)
      if (a[j] < a[i] && l[j] > l[jmax]) jmax = j;
      l[i] = l[jmax] + 1;
      trace[i] = jmax;
}
lmax = l[n+1] - 1;
```

```
void truy_vet_xuoi()
{
  // gt[] là mảng lưu các phần tử của dãy tìm được,
   cout << lmax << endl;
   int u = trace[n+1];
    int dem = 0;
    while (u) {
           gt[++dem] = a[u];
           u = trace[u];        }
    FORD(i,dem,1) cout << gt[i] << " ";
  // dem = lmax
}
```

Chú ý: Ta có thể quy hoạch động với L[i] là độ dài dãy tăng dài nhất bắt đầu tại A_i.Với cách này, khi truy vết ta có thể đưa ra luôn không cần sử dụng mảng *gt*[]. Yêu cầu thực hiện code thêm theo cách này.
Độ phức tạp: O(N^2).
Có một số thuật toán với độ phức tạp O(*NlogN*) sẽ được trình bày sau.

Bài 4. Dãy con chung dài nhất (LCS – Longest Common Subsequence)

Cho 2 dãy số $A_1, A_2, \ldots, A_m$ và $B_1, B_2, \ldots, B_n$. Tìm dãy con chung dài nhất của 2 dãy.

Dữ liệu: Vào từ file **LCS.INP**

- Dòng đầu tiên ghi 2 số m, n ($1 \le m,n \le 100$)
- Dòng thứ 2 ghi m số $A1, A2, \ldots, A_m$ ($|A_i| \le 100$)
- Dòng thứ 3 ghi n số $B1, B2, \ldots, B_n$ ($|B_i| \le 100$)

Kết quả: Ghi ra file **LCS.OUT**

- Dòng đầu tiên ghi l - số lượng phần tử của dãy con chung dài nhất tìm được.
- Dòng thứ hai ghi l phần tử của dãy con chung tăng tìm được.
- Dòng thứ ba ghi l số là chỉ số tương ứng của dãy con chung trong dãy A
- Dòng thứ tư ghi l số là chỉ số tương ứng của dãy con chung trong dãy B

LCS.INP	LCS.OUT
6 7 1 5 2 8 3 7 4 2 5 6 1 2 3	3 5 2 3 2 3 5 3 6 7

Thuật toán: Gọi L[i][j] là độ dài dãy con chung dài nhất của 2 dãy $A_1, A_2, \ldots, A_i$ và $B_1, B_2, \ldots, B_j$. Ta dễ thấy:
Nếu $A_i = B_j$, ta thêm phần tử A_i(hoặc B_j) vào đuôi dãy con dài nhất của $A_1, A_2, \ldots, A_{i-1}$ và $B_1, B_2, \ldots, B_{j-1}$.
L[i][j] = L[i-1][j-1] + 1
Nếu $A_i \neq B_j$, L[i][j] = ***max*** (L[i-1][j], L[i][j-1]).
Độ dài lớn nhất cần tìm là L[m][n].

Quy Hoạch Động
`FOR(i,1,m)`
`   FOR(j,1,n)`
`      if (a[i] == b[j]) L[i][j] = L[i-1][j-1] + 1;`
`         else L[i][j] = max(L[i-1][j], L[i][j-1]);`
`kq = L[m][n];`
 	
Ghi Kết Quả
Gt: là mảng lưu giá trị dãy con chung
L1: Là mảng lưu các vị trí tương ứng trong dãy A
L2: Là mảng lưu các vị trí tương ứng trong dãy B
	`void Tim_Vet()` **Truy vết**
	`{`
	`    int i = m;`
	`    int j = n;`
	`    int vt = kq; // L[m][n]`
	`    while (i && j)    {`
	`        if (a[i] == b[j])`
	`        {`
	`            gt[vt] = a[i]; // đưa vào dãy`
	`            l1[vt] = i--;`
	`            l2[vt--] = j--;// viết tắt câu lệnh`
	`        } else`
	`            if (L[i][j] == L[i-1][j]) i--;`
	`                else j --;`
	`    }`
	`}`

Bài tập áp dụng tương ứng: Xâu con chung dài nhất của 3 xâu. (Xâu chung)

Bài 5. Sinh tổng

Cho dãy số $A_1, A_2, \ldots, A_n$. Liệt kê tất cả các tổng có thể sinh ra từ dãy số đã cho.

Ví dụ: Dãy 1,3,4. Các tổng có thể sinh ra: 1, 3, 4, 5, 7, 8.

SUM.INP	SUM.OUT
3 1 3 4	1 3 4 5 7 8

Dữ liệu: Vào từ file **SUM.INP**

- Dòng đầu tiên ghi số n ($1 \le n \le 100$)
- Dòng thứ 2 ghi n số nguyên dương $A1, A2, \ldots, A_n$ ($A_i \le 100$)

Kết quả: Ghi ra file **SUM.OUT** m số có thể tạo thành từ các số đã cho theo thứ tự tăng dần.

Thuật toán: Sử dụng mảng dd[i][j] đánh dấu.

dd[i][j] = 1 nếu có thể tạo thành tổng j bởi i số đầu tiên. dd[i][j] = 0 trong trường hợp ngược lại.

Cơ sở QHĐ: dd[0][0] = 1. dd[0][i] = 0 với $i \neq 0$

Tính dd[i][j] = dd[i-1][j] || dd[i-1][j- A_i]

QHĐ tính dd[i][j]
`for (int i = 1; i <= n; i++)`
`{`
`   for (int j = 0; j <= sum; j ++)`
`      dd[i][j] = dd[i-1][j];`
`   for (int j = a[i]; j <= sum; j ++)`
`      if (dd[i-1][j-a[i]]) dd[i][j] = 1;`
`}`
 	
Ghi Kết Quả
dd[n][x] == 1 → có thể sinh ra tổng **x**
	`dd[0] = 1;`
	`for (int i = 1; i <= n; i++)`
	`    for (int j = sum; j >= a[i]; j--)`
	`        if (dd[j-a[i]]) dd[j] = 1;`

Ta cũng có thể chỉ sử dụng mảng 1 chiều dd[i] với ý nghĩa có thể sinh ra tổng i hay không (code bên phải).

Question:

- *Tại sao vòng lặp thứ 2 phía phải, j lại chạy giảm dần?*
- *Nếu A_i có thể âm cần thực hiện thế nào?*

Bài 6. Chia kẹo

Có N gói kẹo. Gói thứ i có A_i chiếc. Cần chia N gói kẹo thành 2 phần (không được bóc gói kẹo ra) sao cho tổng số kẹo chênh lệch của hai phần là ít nhất.

CHIAKEO.INP	CHIAKEO.OUT
5 1 6 3 5 8	1 1 2 4 3 5

Dữ liệu: Vào từ file **CHIAKEO.INP**

- Dòng đầu tiên ghi số n ($1 \le n \le 100$)
- Dòng thứ 2 ghi n số nguyên dương $A1, A2, \ldots, A_n$ ($A_i \le 100$)

Kết quả: Ghi ra file **CHIAKEO.OUT**

- Dòng đầu ghi chênh lệch tổng số kẹo trong 2 phần sau khi chia.
- Dòng thứ hai ghi chỉ số của gói kẹo trong phần chia thứ nhất.
- Dòng thứ ba ghi chỉ số các gói kẹo trong phần chia thứ hai.

Thuật toán: Để được độ lệch hai phần chia là nhỏ nhất thì phần nhỏ phải gần giá trị sum/2 nhất. Với sum là tổng các A_i(i=1,2…n). Do đó, để tìm phần nhỏ, ta cần xét giá trị nào gần nhất với sum/2 và có thể tạo thành từ N số đã cho. Gọi phần nhỏ là X. Phần còn lại, chắc chắn là N-X.
Để xét xem có thể tạo thành tổng X hay không, ta thực hiện giống bài 5 phía trên.
Ở bài toán này, ta cần biết những số nào tạo thành tổng X, nên cần làm khác một chút.
Gọi tr[j] là chỉ số gói kẹo nhỏ nhất tạo thành tổng j.

```
FOR(i,1,n)
    FORD(j,sum,0)
        if (dd[j]==1 && dd[j+a[i]]==0){
           dd[j + a[i]] = 1;
           tr[j + a[i]] = i;
        }
s1 = sum/2;
while (dd[s1]==0) s1 --; // tìm phần nhỏ
kq = sum - 2*s1;//chênh lệch nhỏ nhất 2 phần
int x = s1;
while (x!=0){  // trừ dần tới 0
     int goi = tr[x]; // gói tạo ra tổng x
     luu[goi] = 1;//đánh dấu gói thuộc phần 1
     x -= a[goi];
}
Luu[i] = 1/0 nếu gói i thuộc phần nhỏ/to
```

```
Code không sử dụng mảng dd[]
Tr[0] = -1;// chọn 1 số bất kì khác 0.
FOR(i,1,n)
   FORD(j,sum,0)
       if (tr[j]!=0 && tr[j+a[i]] == 0)
          tr[j + a[i]] = i;
Phần truy vết không khác gì với việc sử dụng dd[]
s1 = sum/2;
while (tr[s1]==0) s1 --;
kq = sum - 2*s1;

int x = s1;
while (x)
{
       int goi = tr[x];
       luu[goi] = 1;
       x -= a[goi];
}
```

Ta cũng có thể không cần sử dụng mảng dd[], vì **dd[x] = 0 ↔ tr[x] != 0** (ban đầu đặt tr[0] = 1 số khác 0).

Bài 7. Đếm cách sinh tổng

Một người đi lấy tiền ở một ngân hàng. Anh ta cần rút đúng P đồng còn ngân hàng còn đúng N đồng tiền với giá trị $A_1, A_2, \dots, A_n$. Hỏi ngân hàng có bao nhiêu cách trả tiền?

Ví dụ: Mệnh giá các đồng tiền 1,3,4,5,8.

P = 8: 8 = 1 + 3 + 4 = 3 + 5 (*3 cách*)

P = 9: 1 + 8 = 1 + 3 + 5 = 4 + 5. (*3 cách*)

COUNTSUM .INP	COUNTSUM .OUT
5 9 1 3 4 5 8	3

Dữ liệu: Vào từ file **COUNTSUM.INP**

- Dòng đầu tiên ghi 2 số nguyên dương n và P ($n \le 100$, $P \le 10000$)
- Dòng thứ 2 ghi n số nguyên dương $A1, A2, \dots, A_n$ ($A_i \le 100$)

Kết quả: Ghi ra file **COUNTSUM.OUT** ghi số cách trả tiền của ngân hàng.

Gọi d[j] là số cách tạo tổng j.

```
d[0] = 1;
for (int i = 1; i <= n; i++)
      for (int j = sum; j >= a[i]; j--) d[j] += d[j-a[i]];
cout << d[p];
```

Bài 8. Bài toán cái túi

Trong siêu thị có N gói hàng. Gói thứ i có trọng lượng W_i và có giá trị V_i. Một tên trộm đột nhập vào siêu thị và sức của tên trộm không thể mang quá trọng lượng M. Hỏi tên trộm phải mang đi những gói hàng nào để lấy được tổng giá trị lớn nhất.

Dữ liệu: Vào từ file **PACK.INP**

- Dòng đầu tiên ghi 2 số n, m ($1 \le n \le 100$, $m \le 10000$)
- Dòng thứ 2 ghi n số $W_1, W_2, \dots, W_n$ ($W_i \le 100$)
- Dòng thứ 3 ghi n số $V_1, V_2, \dots, V_n$ ($V_i \le 10^4$)

PACK.INP	PACK.OUT
4 10 9 6 2 8 10 2 5 6	11 2 3 4

Kết quả: Ghi ra file **PACK.OUT**

- Dòng đầu tiên ghi tổng giá trị lớn nhất có thể được.
- Dòng thứ hai ghi l – số gói hàng tên trộm sẽ lấy đi.
- Dòng cuối cùng ghi l số là chỉ số xác định gói hàng tên trộm mang theo.

Thuật toán:

Gọi F[i][j] là tổng giá trị lớn nhất có thể bằng cách chọn trong các gói 1, 2, …, i để được trọng lượng đúng bằng j.
Cơ sở QHĐ: F[0][j] = 0 với mọi j.
Tính : Khi xét tới gói thứ i, nếu không chọn gói i: F[i][j] = F[i-1][j].
Trong trường hợp chon gói i, ta có F[i][j] = F[i-1][j-W_i] + V_i.
Do đó: F[i][j] = ***max*** (F[i-1][j], F[i-1][j-W_i] + V_i).

```
memset(F, 0, sizeof(F));
FOR(i,1,n)
   FOR(j, 1, m)
      if (j < w[i]) F[i][j] = F[i-1][j];
      else
  F[i][j] = max(F[i-1][j], F[i-1][j-w[i]] + v[i]);

int x = 0;//Tổng trọng lượng khi giá trị max
FOR(j, 1, m)
      if (F[n][j] > F[n][x]) x = j;
cout << F[n][x] << endl; //Tổng giá trị max
```

```
int dem = 0;
FORD(i,n,1)
 If (F[i][x] != F[i-1][x])
// nếu chọn gói i
     {
            luu[++dem] = i;
            x -= w[i];
     }
cout << dem << endl;
FORD(i, dem, 1) cout << luu[i] << " ";
```

Bài 9. Rút bài

Có N lá bài ($3 \le N \le 100$), trên mỗi lá bài ghi một số nguyên dương không vượt quá 1000. Các quân bài được xếp thành một chồng. Người ta lần lượt rút các lá bài *bên trong* chồng bài (tức là trừ lá bài trên cùng và dưới cùng), mỗi lần rút một quân cho đến khi chỉ còn lại lá trên cùng và dưới cùng. Chi phí rút một lá bài là tích ba số ghi ở lá bài trên lá được rút, lá bài dưới lá được rút và số ghi trên lá bài được rút. Khi rút hết N-2 lá bài, ta có tổng chi phí rút bài. Mỗi trình tự rút bài sẽ có một tổng chi phí.

Ví dụ, N = 5 và số ghi trên các lá bài là 10 1 50 20 5. Nếu lần lượt rút các lá bài có các số 1, 20 và 50, thì tổng chi phí là:

$$10*1*50 + 50*20*5 + 10*50*5 = 500+5000+2500 = 8000$$

Còn nếu rút các lá với số 50, 20 và 1 thì tổng chi phí sẽ là:

$$1*50*20 + 1*20*5 + 10*1*5 = 1000+100+50 = 1150$$

Yêu cầu: Hãy tính tổng chi phí nhỏ nhất.

Dữ liệu: Vào từ file **CARD.INP**
- Dòng đầu tiên ghi số n
- Dòng thứ 2 ghi n số $A1, A2,…,A_n$ được ghi trên n lá bài.

Kết quả: Ghi ra file **CARD.OUT** tổng chi phí nhỏ nhất tìm được.

CARD.INP	CARD .OUT
5 10 1 50 20 5	1150

Thuật toán:

Gọi F[i][j] là tổng chi phí nhỏ nhất để rút hết các lá bài từ vị trí i+1 tới vị trí j-1.
Cơ sở QHĐ: F[i][i+1] = 0 với mọi I = 1, 2, …, n-1
Tính F[i][j]: Gọi k là lá bài cuối cùng ta rút trong khoảng (i, j).
F[i][j] = F[i][k] + F[k][j] + A_i * A_k * A_j.
Ta thử tất cả các k = i+1,…,j-1 để tìm giá trị tối ưu.
Kết quả bài toán là F[1][n].
Chú ý: Ta cần tính các cặp (i,j) "nở" dần ra, tức chênh lệch i và j tăng dần từ 2 tới n-1.

```
memset(F, 0, sizeof(F));
FOR(kc, 2, n-1)
   FOR(i, 1, n-kc){
      int j = i + kc;
      F[i][j] = maxc;
      FOR(k,i+1,j-1) F[i][j] = min(F[i][j],
F[i][k] + F[k][j] + a[i] * a[k] * a[j]);
   }
cout << F[1][n] << endl; //Tổng giá trị nhỏ nhất
```

```
Ta cũng có thể duyệt các cặp (i,j) như sau:
FORD(i,n-1,1)
  FOR(j,i+2,n){
      F[i][j] = maxc;
      FOR(k,i+1,j-1)
      F[i][j] = min(F[i][j],
F[i][k] + F[k][j] + a[i] * a[k] * a[j]);
  }
```

Bài 10. Biến đổi xâu

Với một xâu ký tự *S* cho trước. Ta có thể thực hiện các phép biến đổi sau:

- D *i*: Xóa ký tự thứ *i* trong xâu *S*.
- I *t c*: chèn trước vị trí *t* của xâu *S* một ký tự *c* nào đó.
- R *t c*: thay ký tự thứ *t* của *S* bởi ký tự *c* nào đó.

Giả sử X và Y là hai xâu ký tự. Chỉ số ký tự trong X, Y đánh số từ 1.

Yêu cầu: Hãy tìm một dãy gồm ít nhất các phép biến đổi xâu X thành xâu Y. (Số phép biến đổi ít nhất gọi là khoảng cách giữa hai xâu).

CHANGE.INP	CHANGE.OUT
trtgui tyuhi	4 D 1 D 1 I h 4 R 2 y

Dữ liệu: Vào từ file **CHANGE.INP** gồm hai dòng:

- Dòng thứ nhất là xâu X.
- Dòng thứ hai là xâu Y.

Kết quả: Ghi ra file **CHANGE.OUT**

- Dòng thứ nhất là K, đó là khoảng cách hai xâu.
- K dòng tiếp theo mỗi dòng ghi ký hiệu một phép biến đổi theo trình tự thực hiện để biến X thành Y.

<u>Thuật toán:</u>

Gọi L[*i*][*j*] là số bước ít nhất để biến đổi xâu $X_1X_2\ldots X_i$ thành xâu $Y_1Y_2\ldots Y_j$.

Cơ sở QHĐ:

- L[0][j] = j : Để biến đổi xâu rỗng thành xâu $Y_1Y_2\ldots Y_j$ ta thực hiện thao tác chèn *j* lần.
- L[i][0] = j : Để biến đổi xâu $X_1X_2\ldots X_i$ thành xâu rỗng ta thực hiện thao tác xóa *i* lần.

Tính L[i][j]:

- Nếu $X_i = Y_j$, hiển nhiên L[*i*][*j*] = L[*i*-1][*j*-1], ta không cần biến đổi gì.
- Nếu $X_i \neq Y_j$, ta có 1 trong 3 phép biến đổi cuối cùng.
  - Chèn ký tự Y_j vào đuôi xâu đã biến đổi thành $Y_1\ldots Y_{j-1}$, số bước là L[*i*][*j*- 1] + 1.
  - Xóa ký tự X_i rồi biến đổi $X_1\ldots X_{i-1}$ thành $Y_1\ldots Y_j$, số bước thực hiện là L[*i*-1][*j*] + 1.
  - Thay ký tự cuối cùng X_i bằng Y_j, số bước cần thực hiện L[*i*-1][*j*-1] + 1.

  Do đó: L[*i*][*j*] = **MIN (**L[*i*][*j*- 1], L[*i*-1][*j*], L[*i*-1][*j*-1]) + 1.

QHĐ tính L[i][j]

```
m = X.length() - 1;
n = Y.length() - 1;
for (int i = 0; i <= 100; i++) L[i][0] =
L[0][i] = i;
for (int i = 1; i <= m; i++)
  for (int j = 1; j <= n; j++)
     if (X[i] == Y[j]) L[i][j] = L[i-1][j-1];
        else L[i][j] = min (min (L[i][j-1],
L[i-1][j]), L[i-1][j-1]) + 1;
cout << L[m][n] << endl;
```

notes: Thủ tục truy vết dùng đệ quy để mô phỏng lại các bước biến đổi. Có nhiều cách truy vết khác hoàn toàn có thể sử dụng.

```
void Truyvet(int i, int j){
    if (i==0 && j == 0) return;
    if (X[i] == Y[j]) Truyvet(i-1, j-1);
    else if (i && L[i][j] == L[i-1][j] + 1){
            Truyvet(i-1,j);
            printf("D %d\n",j+1);
        }
   else if (j && L[i][j] == L[i][j-1] + 1){
            Truyvet(i,j-1);
            printf("I %d %c\n", j, Y[j]);
        }else{
            Truyvet(i-1,j-1);
            printf("R %d %c\n", j, Y[j]);
        }
}
```

Bài 11. Palindrome - Xâu đối xứng

Palindrome là một xâu đối xứng, tức là một xâu mà đọc từ trái sang phải cũng giống như đọc từ phải sang trái. Bạn cần viết một chương trình với một xâu cho trước, xác định số ít nhất các ký tự cần chèn vào xâu để nhận được một Palindrome. Ví dụ, bằng cách chèn hai ký tự vào xâu "Ab3bd" ta nhận được một Palindrome ("dAb3bAd" hoặc "Adb3bdA"). Tuy nhiên, nếu chèn ít hơn 2 ký tự thì không thể tạo được Palindrome.

Dữ liệu: Vào từ file **PALIN.INP** gồm hai dòng:

- Dòng thứ nhất gồm một số nguyên là độ dài N của xâu, $3 \leq N \leq 5000$.
- Dòng thứ hai gồm một xâu có độ dài N. Xâu gồm các ký tự là các chữ cái hoa A..Z, các chữ cái thường a..z và các chữ số thập phân 0..9, các chữ cái hoa và thường xem như là khác nhau.

PALIN.INP	PALIN.OUT
5 Ab3bd	2

Kết quả: Ghi ra file **PALIN.OUT** một số nguyên là số lượng ký tự tối thiểu cần chèn vào.

Thuật toán:

Gọi L[i][j] là số ký tự ít nhất để chèn vào để xâu $S_iS_{i+1}....S_j$ thành đối xứng.

Cơ sở QHĐ:

- L[i][i] = 0 :Xâu chỉ gồm 1 ký tự S_i vốn đã đối xứng nên không cần chèn thêm gì.
- L[i][i+1] = 0 nếu $S_i = S_{i+1}$ và L[i][i+1] = 1 nếu $S_i \neq S_{i+1}$. Trường hợp này ta chỉ cần chèn 1 ký tự giống S_i vào cuối hoặc 1 ký tự giống S_{i+1} vào đầu.

Tính L[i][j]tổng quát:

- Nếu $S_i = S_j$: L[i][j] = L[i+1][j-1], ta không cần chèn thêm gì cả, chỉ cần biến đối $S_{i+1}..S_{j-1}$ thành đối xứng.
- Nếu $S_i \neq S_j$, ta có 2 cách biến đổi cuối cùng.
  - Biến đổi $S_iS_{i+1}....S_{j-1}$ thành đối xứng rồi chèn thêm S_j vào đầu. Số bước là L[i][j- 1] + 1.
  - Biến đổi $S_{i+1}....S_{j-1}S_j$ thành đối xứng rồi chèn thêm S_i vào cuối. Số bước là L[i+1][j] + 1.

  Do đó: L[i][j] = **MIN (**L[i][j- 1], L[i+1][j]) + 1.

Bài tiếp:

Cho hình chữ nhật. Tìm đường đi có tổng lớn nhất từ đỉnh trái trên tới đỉnh phải dưới.