
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BÀI TẬP MÔN PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

BÀI TẬP THỰC HÀNH SỐ 03 THIẾT KẾ GIẢI THUẬT

GV hướng dẫn: Huỳnh Thị Thanh Thương

Nhóm thực hiện:

Nguyễn Thị Kim Anh MSSV 20521072

TP.HCM, ngày 23 tháng 1 năm 2022

---

1. BÀI TẬP 1: Số chính phương

1.1. Mô tả bài toán: Bài toán tìm tất cả các số chính phương trong một danh sách các số nguyên cho trước.

a. Viết hàm kiểm tra số chính phương

b. Áp dụng chia để trị để viết thuật toán tìm tất cả các số chính phương

c. Phân tích độ phức tạp của thuật toán tìm tất cả các số chính phương.

1.2. Mô hình hóa bài toán:

Input: số nguyên dương n không vượt quá 1000,. Dãy số nguyên có n phần tử không vượt quá 10^9.

Output: Mảng b chứa Vị trí các số chính phương tìm được (1 tương ứng với số đó là số cp)

1.3. Thiết kế thuật toán: Áp dụng chia để trị

Ý tưởng: Chia mảng số nguyên thành 2 phần. Tìm số chính phương mảng bên trái, tìm số chính phương mảng bên phải=> tổng hợp lại

Cấu trúc dữ liệu: Ngoài biến n, thì mảng a để lưu trữ mảng đầu. Mảng b để lưu trữ mảng vị trí là output của bài.

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng(em có quay video)

1.5. Phân tích độ phức tạp

T(n)=C1 khi n=0

T(n)=2T(n/2) + 2*(n/2)+C2,n>0

=O(nlogn)

Nhận xét: duyệt từng phần tử của mảng chỉ mất O(N) =>Nhanh hơn

```
timSoChinhPhuong(a,l,r){
        if (l==r)
                return;  //kết thúc
```

```
int m=(l+r)/2;   //chia mảng ra làm 2
 timSoChinhPhuong(a,l,m);  //tìm bên trái
 danhdau(a,l,m);
 timSoChinhPhuong(a,m+1,r); //tìm bên phải
 danhdau(a,m+1,r);
```

2. BÀI TẬP 2: Đếm số nghịch thế (Counting Inversions)

1.1. Mô tả bài toán: Cho 1 mảng A có n phần tử không được sắp xếp. Yêu cầu tìm số cặp nghịch thế trong mảng. Với cặp nghịch thế là 2 cặp số có thứ tự lớn bé khác với thứ tự theo chỉ số trong mảng,

1.2. Mô hình hóa bài toán:

Input: số nguyên dương n không vượt quá 1000,. Dãy số nguyên có n phần tử không vượt quá 10^9.

Output: 1 số nguyên nhỏ hơn 1000 là số cặp nghịch thế trong mảng .

1.3. Thiết kế thuật toán:

Ý tưởng: Dựa trên thuật toán merge sort do thuật toán merge sort dựa trên việc đổi chỗ 2 cặp nghịch thế để tạo nên dãy được sắp xếp theo thứ tự.Ở thao tác merge, khi giá trị bên phải nhỏ hơn bên trái thì đổi chỗ và cộng thêm biến đếm 1 giá trị bằng khoảng cách giữa 2 số vừa đổi chỗ.

Cấu trúc dữ liệu:Biến res để lưu kết quả. Dùng mảng L,R để lưu trữ 2 mảng con trái phải để gộp lại cho mảng a và kiểm tra cặp nghịch thế trên 2 mảng, vì dựa trên thao tác mergesort nên mất chi phí n.logn

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng(em có quay video)

1.5. Phân tích độ phức tạp :

T(n)=C1, n=1

T(n)=2.T(n.2)+nC2,n>1 Giống thuật toán merge sort nên DPT là O(nlogn)

MÃ GIẢ:

```
long merge(){
        int n1=m-l+1;          // độ dài nửa bên trái
        int n2=r-m;          // độ dài nửa bên phải
        int L[n1],R[n2];
        for (i->n1)    L[i]=a[l+i];       //  giống như merge sort
        for (i->n2)     R[i]=a[m+1+i];
        res=0; k=l
        while(i<n1&& j<n2){
                if(L[i]>=R[j])
                        res+=n1-i;       // đổi n1-i số
                        a[k++]=R[j++]
                Else   a[k++]=L[j++]
Truyền lại phần dư mảng L và R vào a[k] tiếp thep
      }
long mergeSort(long a[],int l,int r,int res){
         res=0;        //res là số cặp nghịch thế
        if (l<r){
                int m=(l+r)/2;
                res+=mergeSort(a,l,m,res);
                res+=mergeSort(a,m+1,r,res);
                res+=merge(a,l,m,r);                    // khi merge thì mình đếm cặp nghịch thế
        }
```

```
        return res;
}
```

3. Bài 3: Hoán đổi 2 phần trong 1 dãy

1.1. Mô tả bài toán: Cho a là 1 mảng gồm n phần tử. Ta cần chuyển m phần tử đầu tiên của mảng với phần còn lại của mảng (n-m phần tử) mà không dùng một mảng phụ

1.2. Mô hình hóa bài toán:

Input: số nguyên dương n không vượt quá 1000, n số nguyên <10^9 tiếp theo là các phần tử của mảng , số nguyên dương m là số lượng phần tử đầu tiên cần hoán đổi(m<=n)

Output: 1 mảng chứa n số nguyên là mảng số sau khi hoán đổi.

1.3. Thiết kế thuật toán:

Ý tưởng: Xét 2 trường hợp :

TH1 với m=n-m, hoán đổi 2 mảng có độ dài bằng nhau bằng cách hoán đổi từng cặp phần tử theo thứ tự.

TH2: với m>n-m Đổi chỗ n-m số giữa phần đầu và phần cuối. Đổi chỗ (m-(n-m)) số với n-m số cuối theo đệ quy

TH3: với m<n-m Đổi chỗ m số giữa phần đầu và phần cuối. Đổi chỗ ((n-m)-m) số với m số đầu theo đệ quy

Đệ quy th2 và 3 để giống như th1.

Cấu trúc dữ liệu: Dùng biến số nguyên như sum,M,n và dung mảng số nguyên lưu trữ mảng số đề bài, mảng số nguyên lưu trữ phần tử của dãy con và mảng số nguyên đánh dấu phần tử được chọn.

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng

1.5. Phân tích độ phức tạp (nếu làm được)

T(n)=n/2 nếu m=n/2

T(n)=n-m+T(m-n+m) , nếu m>n-m

=m+T(n-m-m), nếu m<n-m

Mã giả :

```
Void doiCho(a,n,m){
                left=m;
                right=n-m;
                if left==right:   doi(a,n-left,m,left);
                  if left!=right:
                        if   left>right:  doi(a,m-i,m,j); left--;
                        else:  right-=left;  doi(a,m-left,m+right,left);
        doi(a,left,right,m)
                for (i in range(m-1)):  swap(a[left +p],a[right+i])
```

4. BÀI TẬP 4:A. Cặp điểm gần nhất (Closest Pair Problem) 1D

1.1. Mô tả bài toán: Cho các tọa điểm nằm trong không gian 1 chiều. Tìm cặp điểm gần nhất.

1.2. Mô hình hóa bài toán:

Input: số nguyên dương n không vượt quá 1000 là số điểm,1 mảng chứa n số thực là tọa độ các điểm có giá trị <10^9

Output: In ra khoảng cách và tọa độ của các điểm gần nhất. là các số thực.

13. Thiết kế thuật toán:

Ý tưởng: Sắp xếp mảng tọa độ các điểm, chia mảng thành 2 phần bằng các lời gọi đệ quy nửa bên trái và nửa bên phải. Tìm cặp điểm gần nhất 2 bên nửa.

Cấu trúc dữ liệu: mảng số thực và biến số nguyên.

1.4 Minh họa(video)

1.5 Độ phức tạp

T(n)= C1 khi n<3

T(n)=2T(n/2) +C2n +C3 khi n>=3

T(n)=O(nlogn) (đã cm trong các bài tập)

Mã giả:

```
float timCapDiem(float a[1000],int l,int r){
        float Min=10e9;
        if((r-l)<=3):
                Duyệt hết phần tử
        int m=(l+r+1)/2;
        float dist_left=timCapDiem(a,l,m);
        float dist_right=timCapDiem(a,m+1,r);
        float dist_min=min(dist_left,dist_right);
        float temp[1000],int index=0;
        for(int i=l;i<r;i++){                    // tìm những điểm thuộc mảng temp
                if (abs(a[m]-a[i])<=dist_min){
                        temp[index]=a[i];
                        index++;
        for (int i=0;i<index-1;i++){          // tìm min trong mảng temp
                if(Min>=(temp[i+1]-temp[i])){
                   Min=temp[i+1]-temp[i];
        return min(dist_min,Min);
```

B. Trong không gian 2D

1.1. Mô tả bài toán: Cho các tọa điểm nằm trong không gian 2 chiều. Tìm cặp điểm gần nhất.

1.2. Mô hình hóa bài toán:

Input: số nguyên dương n không vượt quá 1000 là số điểm,1 mảng chứa n cặp số thực là tọa độ các điểm có giá trị <10^9.

Output: In ra khoảng cách các điểm gần nhất là các số thực.

1.3. Thiết kế thuật toán:

Ý tưởng: Chia mảng thành 2 phần bằng các lời gọi đệ quy nửa bên trái và nửa bên phải. Tìm cặp điểm gần nhất 2 bên nửa rồi so sánh để tìm cặp điểm gần nhất. Trong đoạn từ pmid-d đến pmid+d với d là khoảng cách min vừa tìm được, tìm cặp điểm gần nhất trong đoạn đó rồi so sánh với d để ra kết quả.

Cấu trúc dữ liệu:

- struct để tạo biến điểm

- biến số nguyên n,m, index, số thực x, y là thành phần trong điểm.

1.4 Minh họa(video)

1.5 Độ phức tạp

T(n)=C1

T(n)=2T(n/2)+C2n+C3

ð DPT O(nlogn)

Mã giả

```
float khoangCach(P,n,temp)
        if (n <= 3)
      return distMin(P, n);    //hàm để tìm tất cả các khoảng cách các điểm rồi so sánh để tìm min
        int m = n/2;
        dist_left = khoangCach(P, m,temp);
        dist_right= khoangCach(P + m, n-m,temp);
        dist  = min(dist_left, dist_right);      //tìm min khoảng cách bên trái và phải
```

```
int index = 0;
merge(P, P+m, P+m, P+n, temp,compareY);      //hàm trộn
for (int i = 0; i < n; i++)        // tìm các điểm trong đoạn Pmid-d đến Pmid +d
P[i] = temp[i];
if (abs(temp[i].x - P[m].x) < dist)
        temp[index] = temp[i];
    index++;
return min(dist, distMin(temp, index));      //tìm min 2 khoảng cách
```

6. Bài tập 6: Minimum spanning tree (Prim)

1.1. Mô tả bài toán: Tìm tổng trọng số cây khung nhỏ nhất

1.2. Mô hình hóa bài toán:

Input: số nguyên n<100 là số đỉnh đồ thị, các số nguyên <10^9 là mỗi phần tử 1 ma trận với n dòng, n cột là các cạnh của các đỉnh có tên là các số từ 0,1,2,…n-1.

Giá trị 1 phần tử của ma trận dòng i, cột j chính là cạnh nối I và j.

Output: 1 số nguyên <10^10 là Tổng trọng số cây khung nhỏ nhất, các số nguyên <100 là các cạnh và các đỉnh nối thành cạnh đó

1.3. Thiết kế thuật toán:

Ý tưởng: Thuật toán Prim

- Khởi tạo mảng xét các đỉnh =0 , nghĩa là chưa xét, và cạnh nhỏ nhất tới mỗi đỉnh là MAX
- Chọn đỉnh 0 là đỉnh đầu tiên của cây khung T, tìm đỉnh gần nhất với nó, cập nhật là cạnh nhỏ nhất của mỗi đỉnh không thuộc T với cây khung T
- Xét các đỉnh còn lại, khi xét đỉnh nào thì đánh dấu đỉnh đó, thêm vào T các đỉnh và các cạnh , rồi cập nhật cạnh nhỏ nhất của mỗi đỉnh không thuộc T với các đỉnh trong T
- Dừng khi đủ n-1 cạnh là đủ các đỉnh tạo cây khung nhỏ nhất.

Cấu trúc dữ liệu:

-Dùng struct để khởi tạo biến cạnh gồm có 3 thuộc tính đỉnh đầu, đỉnh cuối và cạnh nối giữa 2 đỉnh

-Dùng số nguyên lưu các biến n,u,v,j,sum, dùng ma trận lưu trữ các cạnh (chứa 3 thuộc tính vd a[i][j].length)

- Dùng các mảng lưu trữ đỉnh đã xét, cạnh nhỏ nhất từ 1 đỉnh, đỉnh gần nhất của 1 đỉnh (daxet[100],canh_min[100],dinh_gn[100])

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng) (video)

1.5 Độ phức tạp: O(N^2) 2 vòng for

Mã giả:

```
for (int i=0;i<n;i++){            // khởi tạo
        daxet[i]=0;
        canh_min[i]=MAX;
        }
        daxet[0]=1;
        for(int i=1;i<n;i++){            //tìm đỉnh gần nhất với đỉnh 0
                if(a[0][i].length<MAX){
                        dinh_gn[i]=0;
                        canh_min[i]=a[0][i].length;
                }
        }
MAX=1 số rất lớn
chondinh(int daxet[],int canh_min[],int n){
        int temp=MAX;
```

```
        int v,v_min;
        for( v=0;v<n;v++){
                if(!daxet[v] && canh_min[v]<=temp){   // đỉnh chưa xét và cạnh gần nhất là
nhỏ nhất
                        temp=canh_min[v];
                        v_min=v;
                }
        return v_min;
thayDoiCanhDinh(edge a[][100],int daxet[],int dinh_gn[],int canh_min[],int v,int n)  //cập
nhật những
        for (int i=1;i<n;i++){                                                  //cạnh nhỏ
nhất từ 1 đỉnh
                if(!daxet[i]&&a[v][i].length<canh_min[i]){
                        canh_min[i]=a[v][i].length;
                 dinh_gn[i]=v;
        for (int j=0;j<n-1;j++){         // tìm n -1 cạnh nhỏ nhất
            v=chondinh(daxet,canh_min,n);
            daxet[v]=1;
            u=dinh_gn[v];
            tree_min[j]=a[u][v];        //cạnh tạo bởi u và v thêm vào mảng khung cây NN
            thayDoiCanhDinh(a,daxet,dinh_gn,canh_min,v,n);
        }
In ra cây khung tree_min
```

- Ý tưởng thuật toán Krusal: Lưu mảng giá trị trọng số các cạnh rồi sắp xếp tăng dần.

Lấy từng giá trị đó bỏ vào mảng tree_min nếu không tạo thành chu trình con, dừng khi đủ số đỉnh của đồ thị.

- Cấu trúc dữ liệu: thêm mảng visited để kiểm tra các đỉnh đã đi qua chưa trong hàm kiểm tra chu trình con, vector adj để lưu đỉnh kề với 1 đỉnh, trong hàm DSF mảng parent lưu đỉnh đi qua trước 1 đỉnh(parent[v]=u).

-Độ phức tạp O(N^2*M) với N là số đỉnh và M là số cạnh

```
for (int k=0;k<m;k++){
        for (int i=0;i<n;i++){
            for (int j=0;j<n;j++){
                memset(visited, false, sizeof(visited));
                 if (a[i][j].length==b[k] && chutrinhcon(tree_min,a,index,i,j,k)==0){
                    tree_min[index]=a[i][j];
                    index++;
```

7. BÀI TẬP 7+8: : Bài toán 8 quân hậu

1.1. Mô tả bài toán: Cho bàn cờ vua kích thước 8x8

Hãy đặt 8 hoàng hậu lên bàn cờ này sao cho không có hoàng hậu nào “ăn” nhau:

▪ Không nằm trên cùng dòng, cùng cột

▪ Không nằm trên cùng đường chéo xuôi, ngược.

1.2. Mô hình hóa bài toán:

Input: 1 ma trận với 8 dòng 8 cột, với mỗi ô là 1 thành phần trong ma trận.

Output: 1 ma trận mà Giá trị của mỗi ô trong ma trận (0 hoặc 1) 1 là vị trí của quân hậu có thể đặt được.

1.3. Thiết kế thuật toán:

Ý tưởng: Khởi tạo tất cả các ô bằng 0 tức là chưa đặt được quân hậu nào. Xét từng ô từ ô từ trái sang phải, kiểm tra xem có đặt được quân hậu hay không?. Nếu có thì đánh dấu và tìm quân hậu tiếp theo sau đó trả về trạng thái ban đầu.

Kiểm tra đặt quân hậu: Kiểm tra đường chéo , hàng và cột xem có quân hậu nào khác không, nếu có thì k đặt được hậu.

Cấu trúc dữ liệu: Dùng ma trận số nguyên gồm 2 số là 0 và 1 có kích thước tùy ý(trong bài này là 8x8)

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng) ---- em làm thành N quân hậu

1.5 Độ phức tạp O(N^2) (xét hết ô vuông trên bàn cờ)

Mã giả:

```
posQueen(int a[][N],int c){
        if c>=N return 1;   //kết thúc đặt hậu
        for (i=0->i<N){
                if (live(a,i,c)==1){      // nếu đặt được hậu
                        a[i][c]=1;
                        if (posQueen(a,c+1))          //tìm hậu tiếp
                                        return 1;
                        a[i][c]=0;  //quay lai
        return 0;                   //ko tìm được
```

9. BÀI TẬP 9: Bài toán các dãy con có tổng cho trước:

1.1. Mô tả bài toán: Cho dãy số nguyên dương (a0 ,a1 , …, an-1) và một số nguyên dương M. Tìm tất cả các dãy con của của dãy a sao cho tổng của các phần tử trong dãy con bằng M

.• TH1: không cần bảo toàn tính thứ tự

• TH2: có ràng buộc bảo toàn tính thứ tự

1.2. Mô hình hóa bài toán:

Input: số nguyên dương n không vượt quá 1000, số nguyên dương M không vượt quá 10^9. Dãy số nguyên dương có n phần tử không vượt quá 10^9.

Output: Nhiều dãy số mà tổng các phần tử trong dãy bằng M.

1.3. Thiết kế thuật toán:

Ý tưởng: với TH1 ta duyệt các phần tử, biến sum lưu trữ tổng các phần tử cần tìm, khởi tạo sum=0 nếu sum+phần tử nào <=M và phần tử đó chưa thêm vào sẽ thêm phần tử đó vào cho đến khi sum=M, sau đó trả lại trạng thái ban đầu cho bài toán.

Cấu trúc dữ liệu: Dùng biến số nguyên như sum,M,n và dung mảng số nguyên lưu trữ mảng số đề bài, mảng số nguyên lưu trữ phần tử của dãy con và mảng số nguyên đánh dấu phần tử được chọn.

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng

1.5. Phân tích độ phức tạp (nếu làm được)

Mã giả:

TH1:

```
void daycon(int i){
        for (int j=1;j<=n;j++){
                if (sum+a[j]<=M && b[j]==0){                //Kiểm tra điều kiện thêm phần tử
                        s[i]=a[j];
                        sum=sum+a[j];
                        b[j]=1;                                 // Đánh dấu phần tử được chọn
                        if (sum==M){                            // Dừng bài toán
                                for (int m=1;m<=i;m++){
                                        cout<<s[m]<<" ";
                                }
```

```
                cout<<endl;
            }
            Else    daycon(i+1);
            b[j]=0;                     //2 bước này trả lại trạng thái ban đầu cho
bài toán
            sum=sum-a[j];
        }
```

TH2:

Độ phức tạp O(N^2)

```
void daycon(int i, int k)
    for (int j=k+1;j<=n;j++)
        if (sum+a[j]<=M){
            s[i]=a[j];
            sum=sum+a[j];
            if (sum==M)
                for (int m=1;m<=i;m++)
                    cout<<s[m]<<" ";
                cout<<endl;
            else
                daycon(i+1,j);
            sum=sum-a[j];
            }
```

10. Bài tập 10: Bài toán nhân chuỗi/dãy ma trận (Matrix Chain Multiplication)

1.1. Mô tả bài toán: Cho 1 chuỗi A1,A2,…An gồm n ma trận với mỗi ma trận Ai có kích thước ai-1xai. Ta cần đóng mở ngoặc để tạo ra thứ tự phép nhân dãy ma trận

1.2. Mô hình hóa bài toán:

Input: Cho số nguyên n <1000 là só lượng ma trận, n +1 số là kích thước của mỗi ma trận

Output: 1 chuỗi là chuỗi con chung dài nhất chiều dài <1000 và 1 số nguyên dương là chiều dài của nó

<1000.

1.3. Thiết kế thuật toán:

Ý tưởng: Quy hoạch động :Tìm lời giải tối ưu

Tách chuỗi ma trận tại ví trí k. => Chi phí tính tích A1*An=chi phí tính A1*Ak+ chi phí tính An*Ak+ nhân chúng lại với nhau.

Gọi m[I,j] là tổng số tối thiểu các phép nhân vô hướng được đòi hỏi để tính Ai*…Aj.

Định nghĩa đệ quy cho chi phí tối thiểu của 1 sự mở đóng ngoặc cho AiAi+1…Aj là:

m[i,j]=0 nếu i=j

=min{ m[i,k]+ m[k+1,j]+ai-1aj} nếu i<j

=>Tạo 1 lời giải tối ưu từ thông tin đã tính toán.

1.4 minh họa

1.5 Độ phức tạp thuật toán

11. BÀI TẬP 11: Bài tập 11: Chuỗi con chung dài nhất (Longest common subsequence)

1.1. Mô tả bài toán: Cho 2 chuỗi s1 và s2.TÌm chuỗi con chung dài nhất và chiều dài của nó

1.2. Mô hình hóa bài toán:

Input: Cho 2 chuỗi string s1,s2 có chiều dài không quá 1000

Output: 1 chuỗi là chuỗi con chung dài nhất chiều dài <1000 và 1 số nguyên dương là chiều dài của nó <1000.

1.3. Thiết kế thuật toán:

Ý tưởng: Quy hoạch động :

- Bước 1: Phương trình đặc trưng. Lời giải bằng chuỗi con chung s1 và s2=> lời giải tối ưu là chuỗi con chung dài nhất. Bài toán con là LCS (chuỗi con). Giảm kích thước bằng cách loại bỏ ký tự.

-Bước 2: Xây dựng phương trình quy hoạch động:

LCS(s1i,s2)=LCS(s1i-1,s2j) hoặc LCS(s1i,s2j-1)

Độ dài = max(LCS(s1i-1,s2j) , LCS(s1i,s2j-1)

Gọi lookup[i][j] là chiều dài chuỗi con chung dài nhất giữa 2 chuỗi s1i,s2

lookup[i][j]=0 khi i=0 ||j=0

=lookup[i-1,j-1] +1 nếu s1i=s2j

=max{ lookup[i-1,j], lookup[i,j-1]} nếu s1i!=s2j

Ví dụ minh họa 2 chuỗi bdcab và abcdab.

s1/s2		0	1	2	3	4	5
			b	d	c	a	b
0		0	0	0	0	0	0
1	a	0	0	0	0	1	<-1
2	b	0	1	<-1	<-1	<-1	2
3	c	0	1^	<-1	2	<-2	<-2
4	d	0	1^	2	<-2	<-2	<-2
5	a	0	1^	2^	<-2	3	<-3
6	b	0	1	2^	<-2	3^	4

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng(em có quay video)

1.5. Phân tích độ phức tạp (nếu làm được)

O(N^2) phải duyệt tất cả các phần tử 2 chuỗi

MÃ GIẢ:

```
int LCSLength()              // hàm tìm chiều dài của chuỗi con chung dài nhất
    for (int i = 1; i <= l1; ++i)
        for (int j = 1; j <= l2; ++j)
            if (s1[i - 1] == s2[j - 1]) lookup[i][j] = lookup[i - 1][j - 1] + 1;
            else lookup[i][j] = max(lookup[i - 1][j], lookup[i][j - 1]); //nó là CT trên
string LCS() {          //hàm tìm chuỗi con chung
    if (l1 == 0 || l2 == 0) return string("");
    if (s1[l1 - 1] == s2[l2 - 1]) return LCS(s1, s2, l1 - 1, l2 - 1) + s1[l1 - 1];
    if (lookup[l1 - 1][l2] > lookup[l1][l2 - 1]) return LCS(s1, s2, l1 - 1, l2);
    return LCS(s1, s2, l1, l2 - 1);
```

Hàm này Ý tưởng cũng giống tìm chiều dài nhưng gọi đệ quy để lần những phần tử chung từ cuối lên đầu

12. BÀI TẬP 12: Tìm đường đi ngắn nhất (Shortest path problem) - Thuật toán Floyd

1.2. Mô hình hóa bài toán: Tìm đường đi ngắn nhất từ 1 đỉnh đến một đỉnh khác cho trước

Input: số nguyên n<100 là số đỉnh đồ thị, số nguyên u, v là đỉnh đầu và đỉnh đích, (u,v<n), các số nguyên <10^9 là mỗi phần tử 1 ma trận với n dòng, n cột là các cạnh của các đỉnh có tên là các số từ 0,1,2,…n-1.

Giá trị 1 phần tử của ma trận dòng i, cột j chính là cạnh nối I và j.

Output: Đường đi ngắn nhất từ 1 đỉnh đến 1 đỉnh khác (1 số nguyên <10e9), phương án đó (ví dụ đường đi từ đỉnh 2->1 là 2->3->4->1

1.3. Thiết kế thuật toán:

Ý tưởng:

- Với mỗi cạnh không xác định, gán giá trị vô cùng lớn

- Tìm đoạn ngắn nhất giữa I và j bằng cách xét các điểm k nằm ngoài điểm I và j, nếu khoảng cách ij> khoảng cách ik+khoảng cách kj thì khoảng cách ij bằng khoảng cách. Cập nhật đỉnh sau i là k chứ không phải j.

- In ra đường đi ngắn nhất từ u->v

Cấu trúc dữ liệu:

- Dùng số nguyên lưu các biến n,u,v,j,sum.

- Dùng ma trận lưu trữ cạnh giữa 2 điểm I ,j a[i][j], ma trận dist_min[i][j] lưu trữ khoảng cách nhỏ nhất giữa 2 đỉnh i và j, dinh_sau[i][j] lưu trữ đỉnh sau đỉnh i trong.

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng) (video)

1.5 Độ phức tạp : O(N^3) (số vòng lặp nhiều nhất là 3 trong ham dung_Floy

T(N)=N*N*N)

Mã giả:

```
void dung_Floy(a,int dist_min,dinh_sau,, n){
  int i, j, k;
  for (i = 1; i <= n; i++){
        for (j = 1; j <= n; j++){
        dist_min[i][j] = a[i][j];   //  gán độ dài max giữa j và i là cạnh nối giữa i và j
        if (dist_min[i][j] == MAX)
              { dinh_sau[i][j] = 0;}      //k tồn tại cạnh nối i và j
```

```
  for (k = 1; k <= n; k++)                    //dùng quy hoạch động để tìm
        for (i = 1; i <= n; i++)
              for (j = 1; j <= n; j++)
                 if (dist_min[i][k] != MAX && dist_min[i][j] > (dist_min[i][k] + dist_min[k][j])){

              dist_min[i][j] = dist_min[i][k] + dist_min[k][j];  // tìm được max khoảng cách i->j
                 dinh_sau[i][j] = dinh_sau[i][k];              // đỉnh sau của i thay đổi
        print đường đi
while (u != v) {
     cout<<"->"<<dinh_sau[u][v];
        u = dinh_sau[u][v];
        }
```

13. BÀI TẬP 13: Tìm đường đi dài nhất (Longest path problem)

1.2. Mô hình hóa bài toán: Tìm đường đi dài nhất từ 1 đỉnh đến một đỉnh khác cho trước

Input: số nguyên n<100 là số đỉnh đồ thị, số nguyên u, v là đỉnh đầu và đỉnh đích, (u,v<n), các số nguyên <10^9 là mỗi phần tử 1 ma trận với n dòng, n cột là các cạnh của các đỉnh có tên là các số từ 0,1,2,…n-1.

Giá trị 1 phần tử của ma trận dòng i, cột j chính là cạnh nối I và j.

Output: Đường đi dài nhất từ 1 đỉnh đến 1 đỉnh khác (1 số nguyên <10e9), phương án đó (ví dụ đường đi từ đỉnh 2->1 là 2->3->4->1

1.3. Thiết kế thuật toán:

Ý tưởng: Giống như bài trên khác là nếu giữ nguyên bài trên thì các đỉnh sẽ được đi qua lại và kéo dài tới vô tận nên ta cần thêm điều kiện, Đó là đồ thì có hướng, không có chu trình, các đỉnh sẽ đi qua 1 lần.

- Với mỗi cạnh không xác định, gán giá trị vô cùng lớn

- Tìm đoạn ngắn nhất giữa I và j bằng cách xét các điểm k nằm ngoài điểm I và j, nếu khoảng cách ij< khoảng cách ik+khoảng cách kj thì khoảng cách ij bằng khoảng cách. Cập nhật đỉnh sau i là k chứ không phải j.

- In ra đường đi dài nhất từ u->v

Cấu trúc dữ liệu:

- Dùng số nguyên lưu các biến n,u,v,j,sum.

- Dùng ma trận lưu trữ cạnh giữa 2 điểm I ,j a[i][j], ma trận dist_min[i][j] lưu trữ khoảng cách nhỏ nhất giữa 2 đỉnh i và j, dinh_sau[i][j] lưu trữ đỉnh sau đỉnh I trong c.

- Lưu mảng flag đánh dấu các đỉnh đã đi qua.

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng) (video)

1.5 Độ phức tạp : O(N^3)

Mã giả:

```
void Floy(a,int dist_ max,flag,dinh_sau, n){
  int i, j, k;
  for (i = 1; i <= n; i++){
        for (j = 1; j <= n; j++){
        dist_max[i][j] = a[i][j];          //gán độ dài max giữa j và j I là cạnh nối giữa i và j
        if (dist_ max [i][j] == MAX)
                { dinh_sau[i][j] = 0;}
  for (k = 1; k <= n; k++)                    //dùng quy hoạch động để tìm
        for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
                if (dist_ max [i][k] != MAX max MAX && flag[k]==0 && dinh_sau[k][v]!=0
&& dist_ max [i][j] < (dist_ max [i][k] + dist_ max [k][j])){          //check đỉnh đó đi qua
chưa và đỉnh sau đỉnh đó có phải đỉnh v?
```

```
                dist_ max [i][j] = dist_ max [i][k] + dist_ max [k][j];

                dinh_sau[i][j] = dinh_sau[i][k];

               flag[k]=1; // đánh dấu

        flag[k]=0;                    // sau vòng for j thì trả lại trạng thái ban đầu bắt đầu 1 đỉnh j mới

        print đường đi

while (u != v) {

    cout<<"->"<<dinh_sau[u][v];

        u = dinh_sau[u][v];

        }
```

14. BÀI TẬP 14: Coin-collecting problem

1.2. Mô hình hóa bài toán: Một số đồng xu được đặt trong các ô của bảng n × m, không nhiều hơn một đồng xu mỗi ô. Một robot, nằm ở ô phía trên bên trái của bảng, cần thu thập như

nhiều tiền nhất có thể và đưa chúng xuống ô dưới cùng bên phải. Trên mỗi bước, rô bốt có thể di chuyển một ô sang phải hoặc một ô xuống dưới từ vị trí hiện tại của nó. Khi rô bốt ghé thăm một ô có đồng xu, nó luôn nhặt đồng xu đó. Thiết kế một thuật toán để tìm số lượng tiền tối đa

rô bốt có thể thu thập và một con đường mà nó cần đi theo để thực hiện việc này.

Input: 2 số nguyên r,c <1000 là kích thước của bảng , r*c số nguyên có giá trị 0 hoặc 1 là giá trị mỗi ô (1 tương ứng với có đồng xu, 0 tương ứng với không có đồng xu).

Output: 1 số nguyên <10^10 là số lượng tiền tối đa rô bốt có thể thu thập và một con đường mà nó cần đi theo để thực hiện việc này.

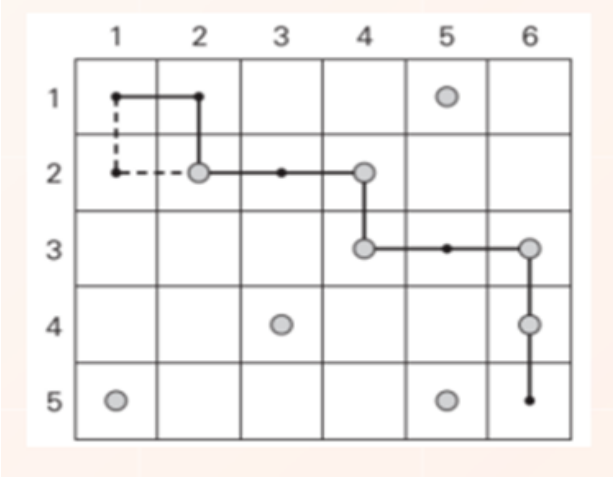
1.3. Thiết kế thuật toán:

Ý tưởng: Vì chỉ được đi xuống hoặc đi sang phải nên tại ô hàng i cột j sẽ bằng những đồng xu góp được ở thời điểm hàng i-1 hoặc j-1 (ở trên hoặc ở bên trái) cộng với đồng xu ở cột j hàng i.

Áp dụng công thức value[i][j]=Max(value[i-1][j],value[i][j-1])+a[i][j];

(a[i][j] co gia tri 0 hoac 1)

Ví dụ ta tạo 1 bảng thể hiện hành trình.



	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1
2	0	0	1	1	2	2	2
3	0	0	1	1	3	3	4
4	0	0	0	2	3	3	5
5	0	1	1	2	3	4	5

Cấu trúc dữ liệu:

- Dùng số nguyên lưu các biến c, r.

- Ma trận a[1000][1000[lưu trữ giá trị mỗi ô trong bảng là có hoặc không có xu. Mảng value[1000][1000] lưu trữ số lượng đồng xu nhiều nhất khi đi tới thời điểm cột j, hàng i.

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng) (video)

1.5 Độ phức tạp : O(N^2) (r*c)

Mã giả:

```
Void coinCollect()

for (i=0 to r)

    for (j=0 to c)

        value[i][j]=Max(value[i-1][j],value[i][j-1])+a[i][j];

        //(a[i][j] co gia tri 0 hoac 1)

cout<<value[r-1][c-1];          // in ra ket qua cuoi cung
```

15. BÀI TẬP 15: Coin-row problem

1.2. Mô hình hóa bài toán: Có 1 dòng gồm n đồng tiền chứa các giá trị dương, không nhất thiết phải khác biệt. Mục tiêu là phải nhặt số lượng tiền nhiều nhất sao cho không được nhặt 2 đồng tiền sát bên nhau.

Input: số nguyên n<1000 là số đồng tiền, n số nguyên dương là giá trị của n đồng tiền (<1000)

Output: 1 số nguyên <10^10 là tổng số lượng tiền nhiều nhất nhặt được thỏa mãn yêu cầu.

1.3. Thiết kế thuật toán:

Ý tưởng: Đưa bài toán về bài toán tìm tổng dãy con lớn nhất trong đó không có 2 phần tử nào liền nhau. Tạo 1 mảng để tìm tổng lớn nhất của dãy con khi xét I phần tử đầu tiên. Áp dụng quy hoạch động và dùng công thức b[i]=max(max(b[i-2]+a[i], b[i-2]), b[i-1]);

Duyệt mảng a và thêm vào từng phần tử cộng dồn vào b[i] ta có: b[i]= max(b[i-2]+a[i], b[i-2])) bởi vì a[i] có thể âm và b[i-2] có thể âm. Theo quy tắc không thêm 2 phần tử liền nhau nên có b[i]=max(max(b[i-2]+a[i], b[i-2]), b[i-1]) .

Cấu trúc dữ liệu:

- Dùng số nguyên lưu các biến n.

- Mảng a để lưu giá trị các số nguyên dương nhập vào, mảng b là mảng chứa tổng lớn nhất khi có i phần tử đầu

Ví dụ mảng có 7 phần tử.

Array a	2	4	5	3	7	9	11
Array b	2	4	7	7	14	16	23

1.4. Ví dụ minh họa: Trình bày các kết quả test của chương trình trên các bộ dữ liệu cụ thể (ít nhất 3 bộ), chụp lại màn hình kết quả để làm minh chứng) (video)

1.5 Độ phức tạp : O(N) (nó duyệt qua n phần tử mảng đầu vào)

Mã giả:

```
        b[0]=a[0];
        b[1]=max(b[0], a[1]);                    //Khởi tạo 2 phần tử đầu
        for(int i=2;i<n;i++)
        {
        b[i]=max(max(b[i-2]+a[i], b[i-1]), b[i-2]);    //so sánh b[i-2], b[i-1] và b[i-2]+a[i], thì
}                                   //b[i] chính là tổng lớn nhất tại thời điểm
                                    //đó( thỏa mãn các pt  kề nhau)
```

In ra max(find_max_elementa(a,n),b[n-1])