BÀI TẬP CHUYÊN ĐỀ: QUY HOẠCH ĐỘNG

Bài 1: HÌNH VUÔNG LỚN NHẤT

Cho một bảng m*n (0 < m,n ≤ 1000) các ô nhỏ chứa các số 0 và 1. Hãy tìm hình vuông chứa toàn số 1 và có diện tích lớn nhất.

- Dữ liệu vào: chứa trong tệp "HINHVUONG.INP".
  - Dòng đầu tiên chứa 2 số m và n.
  - m dòng tiếp theo chứa các số 0 và 1 là biểu diễn của bảng.
- Dữ liệu ra: chứa trong tệp "HINHVUONG.OUT".
  - Một dòng chứa diện tích lớn nhất của hình vuông tìm được.

Ví dụ:

HINHVUONG.INP	HINHVUONG.OUT
4 5 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1	9

<u>Hướng dẫn:</u>

- Gọi B[m,n] là bảng chứa cạnh của hình vuông lớn nhất tìm được. Từ đó, cạnh hình vuông lớn nhất là Max(B[m,n]). Ta có công thức quy hoạch động sau:

$$\begin{cases} B[0,i] = A[0,i] & ,\forall i \le m \\ B[i,0] = A[i,0] & ,\forall i \le n \\ B[i,j] = Min(B[i-1,j-1], B[i,j-1], B[i-1,i]) + 1 & ,\forall i \le n, 2 \le j \le n, A[i,j] = 1. \\ B[i,j] = 0 & ,\forall A[i,j] = 0 \end{cases}$$

- Kết quả là bình phương của Max(B[m,n]).
- Độ phức tạp thuật toán: O(n*m).

<u>Code tham khảo:</u>

```
#include <iostream>
#include <fstream>

using namespace std;

int a[1000][1000],b[1000][1000];
int m,n,i,j,tmp;

int min(int a,int b)
{
        return (a<b) ? a:b;
}

int main()
{
        ifstream fi("HINHVUONG.INP");
```

```
	ofstream fo("HINHVUONG.OUT");
	fi>>m>>n;
	for (i=0;i<m;i++)
		for (j=0;j<n;j++) fi>>a[i][j];
	for (i=0;i<m;i++) b[0][i]=a[0][i];
	for (i=0;i<n;i++) b[i][0]=a[i][0];
	for (i=1;i<n;i++)
		for (j=1;j<m;j++)
			b[i][j]=min(min(b[i-1][j-1],b[i][j-1]),b[i-1][i])+1;
	tmp=b[0][0];
	for (i=0;i<m;i++)
		for (j=0;j<n;j++)
			if (tmp < b[i][j]) tmp=b[i][j];
	fo<<tmp*tmp;
	fi.close();
	fo.close();
	return 0;
}
```

Bài 2: PHÂN TÍCH SỐ

Cho số tự nhiên n ≤ 100. Hãy cho biết có bao nhiêu các phân tích số n thành tổng của dãy các số nguyên dương, các cách phân tích là hoán vị của nhau chỉ tính là một cách.

Ví dụ, với n = 5, ta có 7 cách phân tích:

- 5 = 1 + 1 + 1 + 1 + 1
- 5 = 1 + 1 + 1 + 2
- 5 = 1 + 1 + 1 + 3
- 5 = 1 + 2 + 2
- 5 = 1 + 4
- 5 = 2 + 3
- 5 = 5

- Dữ liệu vào: Chứa trong tệp "PHANTICH.INP".
  - Gồm 1 dòng duy nhất là số n.
- Dữ liệu ra: Chứa trong tệp "PHANTICH.OUT".
  - Gồm 1 dòng duy nhất là số các phân tích số n thành tổng của dãy các số nguyên dương.

Ví dụ:

PHANTICH.INP	PHANTICH.OUT
5	7

Hướng dẫn:

- Gọi F[m,v] là số cách phân tích số v thành tổng của các số nguyên dương ≤ m. Như vậy có chứa ít nhất 1 số m trong phép phân tích. Khi đó nếu trong cách phân tích ta bỏ đi số m thì ta sẽ được các cách phân tích số v – m thành tổng các số nguyên dương ≤ m.
- Ta có công thức quy hoạch động sau:
  - F[0,k]=1 ; ∀k: 1 ≤ k ≤ n.
  - ∀i: 1 ≤ i ≤ n:
    - F[i,k] = F[i,k-1] + F[i-k,k] ; ∀k: 1 ≤ k ≤ i.
    - F[i,k] = F[i,k-1] ; ∀k: i+1 ≤ k ≤ n.

- Kết quả là: F[n,n].
- Độ phức tạp thuật toán: $O(n^2)$.

Code tham khảo:

```
#include <iostream>
#include <fstream>

using namespace std;

int n,f[100][100],i,j,k;

int main()
{
	ifstream fi("PHANTICH.INP");
	ofstream fo("PHANTICH.OUT");
	fi>>n;
	for (k=1;k<=n;k++) f[0][k]=1;
	for (i=1;i<=n;i++)
	{
		for (k=1;k<=i;k++) f[i][k]=f[i][k-1]+f[i-k][k];
		for (k=i+1;k<=n;k++) f[i][k]=f[i][k-1];
	}
	fo<<f[n][n];
	fi.close();
	fo.close();
}
```

Bài 3: DÃY CON ĐƠN ĐIỆU TĂNG DÀI NHẤT.

Cho một dãy số nguyên gồm N phần tử A[1], A[2], ... A[N]. Biết rằng dãy con tăng đơn điệu là 1 dãy $A[i_1],\dots A[i_k]$ thỏa mãn $i_1 < i_2 < \dots < i_k$ và $A[i_1] < A[i_2] < .. < A[i_k]$. Hãy cho biết dãy con tăng đơn điệu dài nhất của dãy này có bao nhiêu phần tử.

- Dữ liệu vào: Chứa trong tệp “DAYCON.INP”.
  - Dòng 1 gồm 1 số nguyên là số N ($1 \le N \le 1000$).
  - Dòng thứ 2 ghi N số nguyên A[1], A[2], .. A[N] ($1 \le A[i] \le 10000$).
- Dữ liệu ra: Chứa trong tệp “DAYCON.OUT”.
  - Dòng 1 là độ dài của dãy con tăng dài nhất.
  - Dòng 2 là các phần tử của dãy con tăng dài nhất, mỗi phần tử cách nhau 1 khoảng trắng.

Ví dụ:

DAYCON.INP	DAYCON.OUT
6 1 2 5 4 6 2	4 1 2 4 6

Hướng dẫn:

- Hàm mục tiêu: f = độ dài dãy con.

- Vì độ dài dãy con chỉ phụ thuộc vào một yếu tố là dãy ban đầu nên bảng phương án là bảng một chiều. Gọi L_i là độ dài dãy con tăng dài nhất, các phần tử lấy trong miền từ A_1 đến A_i và phần tử cuối cùng là A_i.
- Nhận xét với cách làm này ta đã chia 1 bài toán lớn (dãy con của n số) thành các bài toán con cùng kiểu có kích thước nhỏ hơn (dãy con của dãy i số).
- Ta có công thức QHĐ để tính L_i như sau:
  - $L_1 = 1$ (hiển nhiên)
  - $L_i = Max(1, L_j)+1$ với mọi phần tử j: $0<j<i$ và $A_j \le A_i$.
- Tính L_i: phần tử đang được xét là A_i .Ta tìm đến phần tử $A_j < A_i$ có L_j lớn nhất. Khi đó nếu bổ sung A_i vào sau dãy con ...A_j ta sẽ được dãy con tăng dần dài nhất xét từ A_1...A_i
- Truy vết: T[i]=j và xuất kết quả.
- Độ phức tạp thuật toán: $O(n^2)$.

Code tham khảo:

```
#include <iostream>
#include <fstream>
#include <memory.h>

using namespace std;
int n,a[100],l[100],t[100],i,j;
int max(int x[],int n)
{
	int tmp=x[0];
	for (i=1;i<n;i++)
		if (tmp<x[i]) tmp=x[i];
	return tmp;
}

int main()
{
	ifstream fi("DAYCON.INP");
	ofstream fo("DAYCON.OUT");
	fi>>n;
	for (i=0;i<n;i++) fi>>a[i];
	memset(t,-1,sizeof(t));
	for (i=0;i<n;i++)
	{
		l[i]=1;
		for (j=0;j<i;j++)
			if ((a[j]<=a[i]) && (l[i]<l[j]+1))
			{
				l[i]=l[j]+1;
				t[i]=j;
			}
	}
	int res=max(l,n);
```

```
        fo<<a[res];
        res=max(t,n);
        while (res!=-1)
        {
                fo<<a[res]<<" ";
                res=t[res];
        }
        fi.close();
        fo.close();
        return 0;
}
```

Bài 4: TỔ HỢP

Biết công thức khai triển nhị thức Newton: $(a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}$. Cho biết n và k, hãy tính C_n^k.

- Dữ liệu vào: Chứa trong tệp "TOHOP.INP".
  - Gồm 1 dòng chứa 2 số nguyên n và k ($0 < k \le n \le 1000$).
- Dữ liệu ra: Chứa trong tệp "TOHOP.OUT".
  - Gồm 1 dòng duy nhất chứa kết quả C_n^k.

Ví dụ:

TOHOP.INP	TOHOP.OUT
4 2	6

<u>Hướng dẫn:</u>

- Ta có: $C_n^0 = C_n^n = 1$.
- Gọi A[n,k] là giá trị của C_n^k. Ta có công thức QHĐ sau:
  - A[i,0] = 1 ; ∀i: 0 ≤ i ≤ n.
  - A[i,i] = 1 ; ∀i: 0 ≤ i ≤ n.
  - A[i,j] = A[i-1,j-1] + A[i-1,j] ; ∀i,j: 0 ≤ i ≤ n, 1 ≤ j ≤ i-1.
- Độ phức tạp thuật toán: $O(n^2)$.

<u>Code tham khảo:</u>

```
#include <iostream>
#include <fstream>
#include <memory.h>

using namespace std;

int n,k,a[1000][1000],i,j;

int main()
{
        ifstream fi("TOHOP.INP");
        ofstream fo("TOHOP.OUT");
```

```
        fi>>n>>k;
        for (i=0;i<=n;i++)
        {
                a[i][0]=1;
                a[i][i]=1;
                for (j=1;j<=i-1;j++)
                        a[i][j]=a[i-1][j-1]+a[i-1][j];
        }
        fo<<a[n][k];
        fi.close();
        fo.close();
        return 0;
}
```

Bài 5: BÀI TOÁN VỀ CON RÙA

Cho một bảng gồm n dòng, m cột, trong các ô là các số nguyên. Con rùa nằm ở ô bên trái phía dưới của bảng và nó cần phải di chuyển đến ô phái bên phải trên cùng của bảng. Mỗi bước con rùa chỉ có thể di chuyển sang ô kề bên phải hoặc phái trên. Hãy tìm con đường cho rùa để đạt được tổng là lớn nhất.

- Dữ liệu vào: Chứa trong tệp "CONRUA.INP".
  - ➢ Dòng đầu là 2 số n và m ($0 < m, n \le 1000$).
  - ➢ n dòng tiếp theo là biểu diễn của bảng.
- Dữ liệu ra: Chứa trong tệp "CONRUA.OUT".
  - ➢ Dòng đầu là tổng lớn nhất tìm được.
  - ➢ Dòng thứ 2 là các bước di chuyển của con rùa.

Ví dụ:

CONRUA.INP	CONRUA.OUT
4 4 9 8 6 2 10 11 13 11 3 7 12 8 5 9 13 9	65 (4,1) => (4,2) => (4,3) => (3,3) => (2,3) => (2,4) => (1,4)

<u>Hướng dẫn:</u>

- Gọi bảng ban đầu là A, bảng kết quả là B. Ta có công thức quy hoạch động sau:
  - $B[n,1] = A[n,1]$.
  - $B[i,1] = B[i+1,1] + A[i,1]$, $\forall i: n-1 \ge i \ge 1$.
  - $B[n,j] = B[n,j-1] + A[n,j]$, $\forall j: 2 \le j \le m$.
  - $B[i,j] = Max(B[i+1,j],B[i,j-1]) + A[i,j]$, $\forall i,j: n-1 \ge i \ge 1; 2 \le j \le m$.
- Tổng lớn nhất là: $B[1,m]$.
- Tiến hành truy vết để in ra đường đi.
- Độ phức tạp thuật toán: $O(n*m)$.

Code tham khảo:

```
#include <fstream>
#include <memory.h>

using namespace std;

long i,j,x,n,m;
long a[1000][1000],b[1000][1000];
long tracerow[1000],tracecol[1000],num;

long Max(long x, long y)
{
	return (x>y) ? x:y;
}

int main()
{
	ifstream fi("CONRUA.INP");
	ofstream fo("CONRUA.OUT");
	fi>>n>>m;
	for (i=1;i<=n;i++)
		for (j=1;j<=m;j++) fi>>a[i][j];
	memset(b,0,sizeof(b));
	b[n][1]=a[n][1];
	for (i=n-1;i>=1;i--) b[i][1]=b[i+1][1]+a[i][1];
	for (j=2;j<=m;j++) b[n][j]=b[n][j-1]+a[n][j];
	for (i=n-1;i>=1;i--)
		for (j=2;j<=m;j++)
			b[i][j]=Max(b[i+1][j],b[i][j-1])+a[i][j];
	fo<<b[1][m]<<endl;
	num=1;
	tracerow[1]=1;
	tracecol[1]=m;
	i=1; j=m;
	while ((i!=n) || (j!=1))
	{
		if (j==1)
		{
			num++; i++;
			tracerow[num]=i;
			tracecol[num]=j;
		}
		else
		if (i==n)
		{
			num++; j--;
```

```
                    tracerow[num]=i;
                    tracecol[num]=j;
            }
            else
            {
                    if (a[i][j]+b[i+1][j]==b[i][j])
                    {
                            num++; i++;
                            tracerow[num]=i;
                            tracecol[num]=j;
                    }
                    else
                    {
                            num++; j--;
                            tracerow[num]=i;
                            tracecol[num]=j;
                    }
            }
    }
    for (i=num;i>1;i--) fo<<"("<<tracerow[i]<<","<<tracecol[i]<<") => ";
    fo<<"("<<tracerow[1]<<","<<tracecol[1]<<")";
    fi.close();
    fo.close();
    return 0;
}
```