Mario and Princess Peach

Limits: 2s, 512 MB

The world of Mario can be imagined as 2D grid of row **N** and column **M**containing a total of **N×M** cell. Mario starts from top-leftmost cell and Princess Peach is located at bottom-rightmost cell.

In each step, Mario can jump *at most* **P** cell right or *at most* **P** cell down, where P is the power of the cell Mario is located. Obviously Mario cannot go beyond the grid. Each cell has **V** points located in it. When Mario steps on a cell he gets the points located in that cell (if **V** is negative **|V|** points is deducted). As you are a programmer, you want to find out what can be the maximum collected points in a given grid if player plays in an optimal way.

Input

The first line contains the number of test cases, **T** (T ≤ 20). Then **T**cases follow. Each of the test case contains **N, M** (1 ≤ N, M ≤ 1000) in the first line. Then **N×M** numbers follow showing the power of each cell (|V| <= 100). After that another **N×M** numbers follow showing the points of that cell.

Limits:

T ≤ 20; 1 ≤ N, M ≤ 1000; 1 ≤ P ≤ 1000; |V| ≤ 100

Output

For each test case print a line “Case x: y” where **x** is the case number and **y** is the maximum points Mario can get while rescuing Princess Peach

Samples

Input	Output
2 3 3 3 3 2 2 3 2 1 1 2 3 -10 6 -1 2 -8 4 3 2 3 3 3 3 2 2 3 2 1 1 2 3 -5 7 -1 2 -3 2 4 1	Case 1: 12 Case 2: 11

Huge input file, avoid slower input methods.

```cpp
int a[1001][1001],v[1001][1001],p[1001][1001];
int rowtree[1001][4001],coltree[1001][4001];
void rowUpdate(int id,int nd,int b,int e,int x,int val){
    if(b==x&&e==x){ rowtree[id][nd] = val; return; }
    int lf=2*nd, rg = 2*nd+1, md = (b+e)/2;
    if(x<=md)rowUpdate(id,lf,b,md,x,val);
    else rowUpdate(id,rg,md+1,e,x,val);
    rowtree[id][nd] = max(rowtree[id][lf],rowtree[id][rg]);
}
void colUpdate(int id,int nd,int b,int e,int x,int val){
    if(b==x&&e==x){ coltree[id][nd] = val; return; }
    int lf=2*nd, rg = 2*nd+1, md = (b+e)/2;
    if(x<=md)colUpdate(id,lf,b,md,x,val);
    else colUpdate(id,rg,md+1,e,x,val);
    coltree[id][nd] = max(coltree[id][lf],coltree[id][rg]);
}
int rowQuery(int id,int nd,int b,int e,int x,int y){
    if(b>y||e<x)return -1000000000;
    if(b>=x&&e<=y)return rowtree[id][nd];
    int lf=2*nd, rg = 2*nd+1, md = (b+e)/2;
    int m1 = rowQuery(id,lf,b,md,x,y);
    int m2 = rowQuery(id,rg,md+1,e,x,y);
    return max(m1,m2);
}
int colQuery(int id,int nd,int b,int e,int x,int y){
    if(b>y||e<x)return -1000000000;
    if(b>=x&&e<=y)return coltree[id][nd];
    int lf=2*nd, rg = 2*nd+1, md = (b+e)/2;
    int m1 = colQuery(id,lf,b,md,x,y);
    int m2 = colQuery(id,rg,md+1,e,x,y);
    return max(m1,m2);
}
int main(){
    int tt; scanf("%d",&tt);
    for(int ks=1; ks<=tt; ks++){
        int n,m; scanf("%d%d",&n,&m);

        for(int i=1; i<=n; i++)
            for(int j=1; j<=m; j++)
                scanf("%d",&p[i][j]);

        for(int i=1; i<=n; i++)
            for(int j=1; j<=m; j++)
                scanf("%d",&v[i][j]);

        for(int i=1; i<=1000; i++)
            for(int j=1; j<=4000; j++)
                rowtree[i][j] = coltree[i][j] = -1000000000;

        rowUpdate(n,1,1,m,m,v[n][m]);
        colUpdate(m,1,1,n,n,v[n][m]);
        a[n][m] = v[n][m];

        for(int j=m; j>=1; j--){
            for(int i=n; i>=1; i--){
                if(i==n&&j==m)continue;
                int rowcnt=-1000000000; int colcnt=-1000000000;
                if(j<m){
                    int x = j+1; int y = min(j+p[i][j],m);
                    rowcnt = rowQuery(i,1,1,m,x,y);
                }
                if(i<n){
                    int x = i+1; int y = min(i+p[i][j],n);
                    colcnt = colQuery(j,1,1,n,x,y);
                }
                int maxx = max(rowcnt,colcnt);
                a[i][j] = maxx+v[i][j];
                rowUpdate(i,1,1,m,j,maxx+v[i][j]);
                colUpdate(j,1,1,n,i,maxx+v[i][j]);
            }
        }
        //int ans = rowQuery(1,1,1,m,1,1);
        int ans = a[1][1];
        printf("Case %d: %d\n",ks,ans);
    }
}
```