***Matrix Graph***

C371\_Coding\_Febraury2023

**Topic**: Queue

**Difficulty Level:** Hard

**Question / Problem Statement**:

Eleanor has given a table, **a**, with **n** rows and **m** columns. The top-left corner of the table has coordinates(0, 0), and the bottom-right corner has coordinates(**n**-1, **m**-1). The ith cell contains integer **a***ij*.

A path in the table is a sequence of cells (r1, c1), (r2, c2),.....,(rk, ck) such that for each i∈{1,....,k-1}, cell (ri, ci) and cell (ri+1, ci+1) share a side.

The weight of the path (r1, c1),(r2, c2),......(rk, ck) is defined by ∑**a**rici from i=1 to k where **a**rici is the weight of the cell (ri, ci).

Eleanor has to answer **q** queries. In each query, you are given the coordinates of two cells, (r1, c1) and (r2, c2). Eleanor must find and return the minimum possible weight of a path of a path connecting them.

Write a program to return the minimum possible weight of a path between (**r1, c1**) and (**r2, c2**) for each query in a new line.

**Note**

A cell can share sides with at most 4 other cells. A cell with coordinates (r, c) shares sides with (r-1, c), (r+1, c), (r, c-1) and (r, c+1).

**Function Description**

In the provided code snippet, implement the provided **matrixGraph(...)** method using the variables to return the minimum possible weight of a path between (r1, c1) and (r2, c2) for each query in a new line. You can write your code in the space below the phrase **“WRITE YOUR LOGIC HERE”**.   
  
There will be multiple test cases running so the Input and Output should match exactly as provided.  
The base Output variable **result** is set to a default value of **-404** which can be modified. Additionally, you can add or remove these output variables.

**Input Format**

The first line contains 2 space-separated integers**,n** (the number of rows in **a**) and **m** (the number of columns in **a**), respectively.

Each of **n** subsequent lines contains **m** space-separated integers. The jth integer in the ith line denotes the value of **a**ij.

The next line contains a single integer, **q**, denoting the number of queries.

Each of the **q** subsequent lines describes a query in the form of 4 space-separated integers: **r1, c1, r2,** and **c2**, respectively.

**Sample Input**

2 6 –denotes **n** and **m**.

0 0 0 0 0 –denotes **a.**

1 9 9 9 1

0 0 0 0 0

3 –denotes **q**

0 0 2 4 –next **q** lines describe a query in the form of **r1, c1, r2**, and **c2.**

0 3 2 3

1 1 1 3

**Constraints**

1 <= **n** <= 7.

1 <= **m** <= 5\*10^3.

0 <= **aij** <=3\*10^3.

1 <= **q** <= 3\*10^4.

0 <= **r1, r2** < **n**.

0 <= **c1, c2** < **m**.

**Output Format**

Output should return the minimum possible weight of a path between (r1, c1) and (r2, c2) for each query in a new line.

**Sample Output**

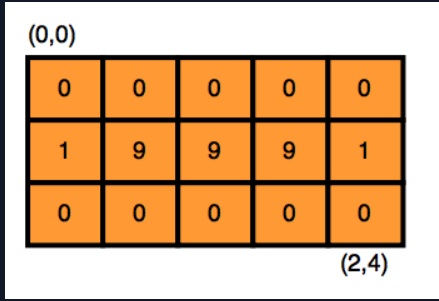
1

1

18

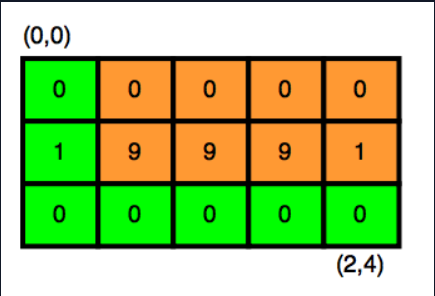
**Explanation**

The input table looks like this:



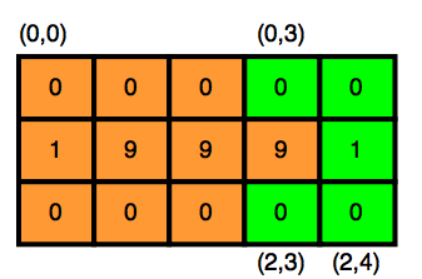
The first two queries are explained below:

1. In the first query, we have to find the minimum possible weight of a path connecting (0,0) and (2,4). Here is one possible path:



The total weight of the path is 0 + 1 +0 +0+0+0+0=1.

2. In the second query, we have to find the minimum possible weight of a path connecting (0,3) and (2,3). Here is one possible path:



The total weight of the path is 0 +0+1+0+0=1.

**Solution Steps**

1. Let's use a divide and conquer approach.

2. Consider column m/2. For all queries such that the first cell is to the left of this column and the second cell is to the right of this column, the path between these cells passes through at least one cell of this table.

3 Let's execute Dijkstra's algorithm n times with starting cells (0, m/2), (1, m/2)...., (n - 1, m/2). We now have n matrices di. i-th matrix contains distances from (i, m/2) to all cells. So minimal-weighted path from x to y (is x from the left from m/2 and y is on the right) is min{ di[x] + di[y] - ai, m/2 : i ∈{0, 1, ……, n-1}}.

4. Then we'll answer queries with cells from the one side of m/2.We must divide our table into two tables (from column 0 to column m/2 and from column m/2 to column m-1 and solve the problem recursively for these parts in the same way. This will be correct only if the path between cells in our queries don't pass through column m/2 but this is easy to deal with. We should only consider all queries in the first step including queries from one side.

5. We have complexity O((n^2\*m \* log(n\*m) + q\*n) \* log m). Because we have log m levels of recursion (we always split tables into two tables with size equal to half of size of original table), every query will be considered only once on each layer.

6. There is another way that turns out to be a little quicker. We should not consider one-side queries, but we should add edges between cells of column m/2 with length equal to the weight of the shortest path between them.

**Running Solution in C++** :

#include <bits/stdc++.h>

using namespace std;

const int maxm=5010;

long long dis[7][75][7][maxm];

bool visited[7][maxm];

const long long INF=1e15;

int weight[7][maxm];

long long temp[7][maxm];

struct cmp{

bool operator()(pair<pair<int,int>,long long > a, pair<pair<int,int>,long long > b)

{

return a.second>b.second;

}

};

bool check(int r,int c,int n,int left,int right,long long d,long long dis[][maxm]){

if(r<0||r>=n)

return false;

if(c<left||c>right)

return false;

if(dis[r][c]<=d+weight[r][c])

return false;

dis[r][c]=d+weight[r][c];

return true;

}

void dijkstra(int r,int c,int n,long long dis[][maxm],int left,int right){

if(left>right)

return;

auto dij=priority\_queue<pair<pair<int,int>,long long >, vector<pair<pair<int,int>,long long >>, cmp>();

for(int i=0;i<n;i++){

fill(visited[i]+left,visited[i]+right+1,false);

fill(dis[i]+left,dis[i]+right+1,INF);

}

dis[r][c]=weight[r][c];

dij.push(make\_pair(make\_pair(r,c), dis[r][c]));

while(dij.size()>0){

pair<pair<int,int>,long long > p=dij.top();

int vr=p.first.first;

int vc=p.first.second;

dij.pop();

if(visited[vr][vc]) continue;

visited[vr][vc]=true;

//left

if(check(vr,vc-1,n,left,right,dis[vr][vc],dis)){

dij.push({{vr,vc-1},dis[vr][vc-1]});

}

//right

if(check(vr,vc+1,n,left,right,dis[vr][vc],dis)){

dij.push({{vr,vc+1},dis[vr][vc+1]});

}

//up

if(check(vr-1,vc,n,left,right,dis[vr][vc],dis)){

dij.push({{vr-1,vc},dis[vr-1][vc]});

}

//down

if(check(vr+1,vc,n,left,right,dis[vr][vc],dis)){

dij.push({{vr+1,vc},dis[vr+1][vc]});

}

}

}

void d1(int r,int c,int n,long long dis[][maxm],int left,int right,int tr,int tc){

if(left>right)

return;

auto dij=priority\_queue<pair<pair<int,int>,long long >, vector<pair<pair<int,int>,long long >>, cmp>();

for(int i=0;i<n;i++){

fill(visited[i]+left,visited[i]+right+1,false);

fill(dis[i]+left,dis[i]+right+1,INF);

}

dis[r][c]=weight[r][c];

dij.push(make\_pair(make\_pair(r,c), dis[r][c]));

while(dij.size()>0){

pair<pair<int,int>,long long> p=dij.top();

int vr=p.first.first;

int vc=p.first.second;

dij.pop();

if(visited[vr][vc])

continue;

visited[vr][vc]=true;

if(vr==tr && vc==tc)

break;

//left

if(check(vr,vc-1,n,left,right,dis[vr][vc],dis)){

dij.push({{vr,vc-1},dis[vr][vc-1]});

}

//right

if(check(vr,vc+1,n,left,right,dis[vr][vc],dis)){

dij.push({{vr,vc+1},dis[vr][vc+1]});

}

//up

if(check(vr-1,vc,n,left,right,dis[vr][vc],dis)){

dij.push({{vr-1,vc},dis[vr-1][vc]});

}

//down

if(check(vr+1,vc,n,left,right,dis[vr][vc],dis)){

dij.push({{vr+1,vc},dis[vr+1][vc]});

}

}

}

int main() {

int n,m;

cin>>n>>m;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

cin>>weight[i][j];

}

}

int segL=200;

int k=(m-1)/segL;

for(int i=0;i<n;i++){

for(int j=0;j<=k;j++){

dijkstra(i,j\*segL,n,dis[i][j],0,m-1);

}

}

int q;

cin>>q;

int r[2];

int c[2];

int g[2];

while(q--){

//cout<<"q="<<q<<endl;

for(int i=0;i<2;i++){

cin>>r[i]>>c[i];

g[i]=c[i]/segL;

//cout<<i<<" "<<g[i]<<endl;

}

if(c[0]%segL==0)

cout<<dis[r[0]][c[0]/segL][r[1]][c[1]]<<endl;

else if(c[1]%segL==0)

cout<<dis[r[1]][c[1]/segL][r[0]][c[0]]<<endl;

else if(g[0]!=g[1]){

int mid=max(g[0],g[1]);

long long ans=INF;

// cout<<"mid="<<mid<<endl;

for(int i=0;i<n;i++){

//cout<<i<<" !!!"<<ans<<endl;

ans=min(ans,dis[i][mid][r[0]][c[0]]+dis[i][mid][r[1]][c[1]]-weight[i][mid\*segL]);

}

cout<<ans<<endl;

}

else{

long long ans=INF;

//cout<<"mid="<<mid<<endl;

for(int i=0;i<n;i++){

//cout<<i<<" !!!"<<ans<<endl;

ans=min(ans,dis[i][g[0]][r[0]][c[0]]+dis[i][g[0]][r[1]][c[1]]-weight[i][g[0]\*segL]);

}

if((g[0]+1)\*segL<m){

for(int i=0;i<n;i++){

// cout<<i<<" !!!"<<ans<<endl;

ans=min(ans,dis[i][g[0]+1][r[0]][c[0]]+dis[i][g[0]+1][r[1]][c[1]]-weight[i][(g[0]+1)\*segL]);

}

}

d1(r[0],c[0],n,temp,g[0]\*segL+1,min(m-1,(g[0]+1)\*segL-1),r[1],c[1]);

ans=min(ans,temp[r[1]][c[1]]);

cout<<ans<<endl;

}

}

return 0;

}

Input:

3 3

0 0 0

1 9 9

1 1 1

3

0 0 1 2

0 2 1 2

1 1 1 2

Output:

9

9

18

**Test Cases [ Qty: 12 ]**

| **Test Case No** | **Input** | **Output** | **Score** |
| --- | --- | --- | --- |
| 1 | 3 5  0 0 0 0 0  1 9 9 9 1  0 0 0 0 0  3  0 0 2 4  0 3 2 3  1 1 1 3 | 1  1  18 | 0 |
| 2 | 3 3  0 0 0  1 9 9  1 1 1  3  0 0 1 2  0 2 1 2  1 1 1 2 | 9  9  18 | 0 |
| 3 | 1 1  0  1  0 0 0 0 | 0 | 1 |
| 4 | 7 5  2408 1225 1201 1511 2227  225 2800 1769 108 1547  1506 2203 1332 1950 2256  2892 2115 816 2302 2849  1927 2587 2424 2995 1752  2966 2565 941 415 1161  2221 2278 823 2409 702  3  0 1 2 4  0 0 2 4  1 2 2 4 | 7848  10256  5680 | 1 |
| 5 | 5 10  1927 2587 2424 2995 1752 1506 2203 1332 1950 2256  2966 2565 941 415 1161 225 2800 1769 108 154  1506 2203 1332 1950 2256 1927 2587 2424 2995 1752  2892 2115 816 2302 2849 2892 2115 816 2302 2849  225 2800 1769 108 1547 2966 2565 941 415 1161  4  0 4 2 5  0 4 3 8  1 0 3 7  0 0 4 9 | 5065  12885  15718  19642 | 1 |
| 6 | 6 10  2 2 2 2 2 2 2 2 2 2  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0  1 2 2 2 2 2 1 1 1 1  4  0 4 4 5  0 4 2 8  1 0 3 7  0 0 5 9 | 3  2  1  4 | 1 |
| 7 | 7 10  2408 1225 1201 1511 2227 2221 2278 823 2409 702  225 2800 1769 108 1547 2966 2565 941 415 1161  1506 2203 1332 1950 2256 1927 2587 2424 2995 1752  2892 2115 816 2302 2849 2892 2115 816 2302 2849  1927 2587 2424 2995 1752 1506 2203 1332 1950 2256  2966 2565 941 415 1161 225 2800 1769 108 154  2221 2278 823 2409 702 2408 1225 1201 1511 2227  4  0 5 2 5  0 4 4 8  1 3 3 7  3 3 6 9 | 7114  15012  11356  14156 | 1 |
| 8 | 1 2  2 5  1  0 0 0 1 | 7 | 1 |
| 9 | 7 5  2408 1225 1201 1511 2227  225 2800 1769 108 1547  1506 2203 1332 1950 2256  2892 2115 816 2302 2849  1927 2587 2424 2995 1752  2966 2565 941 415 1161  2221 2278 823 2409 702  4  0 1 2 4  0 3 4 4  1 2 3 4  3 3 6 4 | 7848  10023  8529  7575 | 1 |
| 10 | 3 5  1927 2587 2424 2995 1752  2966 2565 941 415 1161  2221 2278 823 2409 702  3  0 0 2 4  1 2 2 4  0 1 2 4 | 10157  3219  8230 | 1 |
| 11 | 1 1  5  1  0 0 0 0 | 5 | 1 |
| 12 | 7 100  1927 2587 2424 2995 1752 1672 1365 2563 2827 871 433 1919 302 994 2946 323 1610 36 682 2311 868 1098 2966 2565 941 415 1161 1152 110 822 1639 840 137 1055 1796 1765 877 1148 971 3000 2411 1956 246 2003 2221 2278 823 2409 702 2581 2518 2702 2699 1035 1778 2642 2668 1794 2115 2065 2660 438 1263 2340 2479 2144 612 2517 1923 362 1291 2256 1107 175 1856 1256 98 1433 1074 259 1703 116 1483 0 2432 1133 16 2504 2892 2115 816 2302 2849 1423 20 862 648 578 583 829  1506 2203 1332 1950 2256 1326 2311 1025 48 149 636 2645 2248 1061 1771 850 66 746 2081 624 1200 2174 225 2800 1769 108 1547 1116 623 1147 289 1187 1383 822 118 2586 1777 44 1793 1245 1341 455 2820 2263 2408 1225 1201 1511 2227 2943 347 179 1412 1898 2542 2470 1039 2972 2861 2078 2224 1569 233 918 2842 574 2001 11 182 305 2900 1850 672 1536 2311 877 797 1299 1878 2277 1621 1104 1270 1618 636 1945 768 306 528 2590 105 2051 2914 1716 1246 778 2307 2971 2860 2502  436 2261 780 1518 913 2629 1530 442 2125 2598 187 179 944 276 2829 2048 598 2037 22 438 319 1914 1132 1524 1645 397 2839 1624 1238 556 979 414 1398 134 861 1781 1403 1542 2043 1573 825 2879 2531 1513 1770 20 529 2736 2814 440 25 991 2261 2840 2571 633 1236 901 1956 92 86 402 372 1738 1150 225 2556 750 1353 847 2443 2280 953 2330 1342 1135 2126 866 689 486 2297 2898 2594 2838 1923 1338 1871 1912 2189 658 1171 516 1403 2350 596 2350 62 2990 2946 525  444 1235 2568 2619 1213 1629 1992 639 569 1903 1937 1671 401 2947 2810 960 1004 135 1121 2387 2215 810 2739 1981 1968 990 31 2074 98 2467 1089 447 48 2527 629 2606 490 1986 2110 604 376 177 603 976 2196 1368 824 818 741 616 1055 1103 2120 644 513 1319 1901 2212 917 2327 1699 1081 1669 889 1442 1165 449 1048 2268 145 830 1237 1207 1765 1081 2759 109 1889 1924 2543 1180 879 1160 37 1713 2703 1919 914 285 1011 599 128 1778 1402 681 1214 2916 837 2921 1036  281 797 1320 607 810 500 2532 76 1390 2776 493 1134 1625 1308 1798 1239 73 2288 1484 1803 1447 1054 740 2679 2197 1749 1003 1826 943 1222 211 1627 2692 1281 2899 865 572 1720 2591 380 2665 2747 52 1358 2882 831 2974 931 858 249 682 236 1766 1922 2065 234 2361 2406 285 1449 949 933 411 2093 692 2072 1946 290 161 1605 1678 958 1906 802 1702 1936 2605 632 262 1104 527 1023 2679 830 2036 2416 1275 1586 1098 885 1327 897 2580 190 160 317 1607 1404 2506 2845  1559 1214 132 1528 752 1568 1017 814 1402 1210 2792 598 1244 1625 2838 1083 2550 1429 1863 1883 1705 801 2610 1578 1132 2448 2417 2609 1675 708 122 2251 1531 658 1542 1698 232 2096 1321 606 2500 1645 562 2275 459 1393 642 18 15 959 2671 1548 1773 1941 803 2603 108 410 2709 1474 2404 2870 2472 1041 456 1893 1418 1630 2077 2028 0 2381 240 1831 2093 1528 48 2948 1295 62 2591 1025 2686 1216 1058 2217 2246 1423 1929 2699 1483 695 2681 2739 725 472 1650 1344 624 2792  1167 560 634 1461 2816 80 1812 301 248 1428 2560 1784 2986 2764 1048 2499 608 1210 2892 2341 1146 1983 946 1296 2785 1881 2240 1185 221 1950 1620 195 27 2549 122 2233 2190 2205 2978 1137 810 448 1490 942 1625 2477 1658 657 2006 744 1664 2210 1473 977 2057 1505 1936 66 2626 2042 2627 2708 1715 912 844 541 254 1496 2722 1100 1637 2845 276 2838 1166 1142 1956 247 912 2634 2987 1064 1495 1176 291 1075 101 1635 1261 612 1363 2788 962 2611 2305 897 472 1783 1419 1416  78  2 47 6 88  6 59 6 25  1 49 1 7  6 44 4 20  6 23 2 64  5 16 6 2  0 28 5 49  6 9 5 89  2 77 3 12  5 62 6 61  1 36 6 96  3 16 1 50  6 95 1 44  2 47 2 4  2 2 6 54  4 77 0 83  1 59 0 61  2 82 2 20  1 38 5 58  6 74 6 24  6 93 0 47  0 58 3 24  5 80 0 64  5 7 6 36  0 97 0 62  0 86 2 22  6 0 3 30  6 0 5 83  1 35 5 26  3 50 0 40  6 99 0 14  6 89 0 49  6 6 2 66  6 56 2 93  2 76 0 0  4 98 1 88  1 56 2 74  1 61 6 36  0 21 4 13  0 78 5 93  0 44 5 11  0 22 5 5  2 55 4 60  3 27 5 21  5 31 4 45  1 52 1 1  6 40 2 95  2 7 4 36  3 0 5 60  2 2 0 84  3 84 0 99  0 82 1 3  4 46 5 65  0 91 0 51  1 97 1 70  1 71 5 2  1 23 0 85  1 38 1 48  5 46 6 68  6 90 3 70  1 81 5 72  0 15 4 2  6 34 4 77  6 87 6 40  0 91 1 38  1 94 1 97  0 71 3 58  3 46 6 66  3 41 2 55  0 19 4 77  2 64 1 53  6 83 5 93  2 79 4 46  5 90 1 70  4 25 6 4  3 90 3 58  1 28 5 41  0 3 4 48 | 48069  42130  45886  32740  46930  19019  21860  91037  69908  6895  62737  35748  54879  48422  58445  9831  4665  68035  25322  59156  58289  42896  24216  33816  39729  65898  33205  89236  13868  13927  91531  49177  63935  42901  84009  14104  20724  28600  10623  20139  41433  20455  7552  12315  19198  56253  58101  30522  66652  85081  18495  82680  24092  46900  36159  72086  65753  16107  25082  25523  15159  14375  47262  50709  56829  6325  13920  22218  14553  65955  15476  15112  35775  28059  26937  33747  17634  51694 | 1 |