**Project CSE-200**

Name: Fazle Rabby Sourav

Reg. No: 2010331033

Email to login lightOj : [sourav.sust.cse.10@gmail.com](mailto:sourav.sust.cse.10@gmail.com)

Password to Login lightOJ : akilipip

Format I have used:

I have attached my solution method description and complexity analysis with my .cpp file of solution of particular problem. For convenience I have also given problem statement at the end of the code in .cpp file. And Then I have also created a Doc file.

/\*

**OJ: Light OJ**

**Name Fazle Rabby Sourav**

**Problem Link:** [**http://www.lightoj.com/volume\_showproblem.php?problem=1149**](http://www.lightoj.com/volume_showproblem.php?problem=1149)

**Problem Name: 1149 - Factors and Multiples**

**Rank: 68**

\*/

/\*

**Solution Method:**

It is a very straight forward Bipartite matching problem. In this problem we are given two sets of numbers, A and B. we have to find the minimum number

of elments, which have to remove, in set A and B that No element in set B is a multiple of any element of set A.

we will assume every number of both sets as a node. if an element of B is divisible by an element of set A,

then we will set up an edge between those two nodes. we will repeat this process for every elements of set A and B. after setting up all possible edges,

we can run simple BPM algo [with dfs] to find the maximum possible matching. it is easy to understand that the maximum matching number and the desired minimum

removal numbers are same number. So the maximum matching number is the desired result

\*/

/\*

**Complexity Analysis:**

The Runnig time of DFS is O(E+V) where E is the number of edges and V is is the number of Vertices. in my solution there can be at most

n\*m= N edges. so the complexity will be O(n\*(n\*m)) or O(m\* n^2);

\*/

Code:

#define REP(i,n)for (i=0;i<n;i++)

#define FOR(i,p,k) for (i=p; i<k;i++)

#define FORE(i, p, k) for(i=p; i<=k; i++)

#define FOREACH(it,x) for(\_\_typeof((x).begin()) it=(x.begin()); it!=(x).end(); ++it)

#define FORD(i,n) for(i=n;i>=0;i--)

#define READ(f) freopen(f, "r", stdin)

#define WRITE(f) freopen(f, "w", stdout)

#define REV(s, e) reverse(s, e)

#define ll long long int

#define CLR(p) memset(p, 0, sizeof(p))

#define mset(p, v) memset(p, v, sizeof(p))

#define ALL(c) c.begin(), c.end()

#define SZ(c) (int)c.size()

#define pb(x) push\_back(x)

#define vs vector<string>

#define vi vector<int>

#define vii vector< pair<int, int> >

#define fs first

#define sc second

#define MP(x, y) make\_pair(x, y)

#define pii pair< int, int >

#define psi pair< string, int >

#define pq priority\_queue

#define LOG(x,BASE) (log10(x)/log10(BASE))

#define EQ(a,b) (fabs(a-b)<ERR)

const int INF **=** 0x7f7f7f7f**;**

#define MAX 105

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///

vi A**,** B**;**

vi adj**[**MAX**];**

int par**[**MAX**];**

bool color**[**MAX**];**

bool dfs**(**int from**)**

**{**

int to**,** i**;**

**if(**color**[**from**])** **return** **false;**

color**[**from**]=**1**;**

// deb(from);

**for(**i**=**0**;** i**<**SZ**(**adj**[**from**]);** i**++)**

**{**

to**=** adj**[**from**][**i**];**

**if(** par**[**to**]==-**1 **||** dfs**(**par**[**to**])==**1**)**

**{**

par**[**to**]=**from**;**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

int BPM**()**

**{**

int i**,** j**,** k**,** cnt**=**0**;**

// deb("BPM ok");

mset**(**par**,** **-**1**);**

**for(**i**=**0**;** i**<**SZ**(**A**);** i**++)**

**{**

CLR**(**color**);**

**if(**dfs**(**i**))**

**{**

cnt**++;**

**}**

**}**

**return** cnt**;**

**}**

int main**()**

**{**

//READ("in.txt");

//WRITE("out.txt");

int i**,** j**,** k**,** result**,** t**=**0**,** tcase**,** m**,** n**;**

cin**>>**tcase**;**

**while(**tcase**--)**

**{**

A**.**clear**();** B**.**clear**();**

scanf**(**"%d"**,** **&**m**);**

**for(**i**=**0**;** i**<**m**;** i**++)**

**{**

scanf**(**"%d"**,** **&**k**);**

A**.**pb**(**k**);**

**}**

scanf**(**"%d"**,** **&**n**);**

**for(**i**=**0**;** i**<**n**;** i**++)**

**{**

scanf**(**"%d"**,** **&**k**);**

B**.**pb**(**k**);**

**}**

**for(**i**=**0**;** i**<**SZ**(**A**);** i**++)**

**{**

**for(**k**=**0**;** k**<**SZ**(**B**);** k**++)**

**{**

**if(**B**[**k**]%**A**[**i**]==**0**)**

**{**

adj**[**i**].**pb**(**k**);**

**}**

**}**

**}**

result**=** BPM**();**

csprint

printf**(**"%d\n"**,** result**);**

**for(**i**=**0**;** i**<**MAX**;** i**++)** adj**[**i**].**clear**();**

**}**

**}**

/\*

**OJ: Light OJ**

**Name Fazle Rabby Sourav**

**Rank: 68**

**Problem name: 1201 - A Perfect Murder**

**Problem Link:** [**http://www.lightoj.com/volume\_showproblem.php?problem=1201**](http://www.lightoj.com/volume_showproblem.php?problem=1201)

\*/

/\*

Solution Method:

we assume each mosqueto as an edge and if there is friendship between two mosquitos, then we need to set up an edge between them.

In the statement it is mentioned that there will be no cycle. and there will be no self edge. we need to convert the graph into a bipartite graph.

we will run a DFS to divide all nodes to two sets where there is no edge between the elements of same set.[bipartite]

then from every node we run a DFS to find the maximum matiching. what will the maximum matching in this case stands for??

the maximum node number which are reachable from other set's node is the maximum mathing which is the same number of mosquito those are not possible to kill.

So the desired ans will be- all mosquito minus the maximum matching[ which cant be killed ]

\*/

/\*

Comlexity Analysis:

for converting the graph into bipartite grphe it is needed to O(V\*(E+V)) time

and for BPM its nedded to O(V\*(V+E)) time

SO the complexity of the solution is O(V\*(V+E))

\*/

#define REP(i,n)for (i=0;i<n;i++)

#define FOR(i,p,k) for (i=p; i<k;i++)

#define FORE(i, p, k) for(i=p; i<=k; i++)

#define FOREACH(it,x) for(\_\_typeof((x).begin()) it=(x.begin()); it!=(x).end(); ++it)

#define FORD(i,n) for(i=n;i>=0;i--)

#define READ(f) freopen(f, "r", stdin)

#define WRITE(f) freopen(f, "w", stdout)

#define REV(s, e) reverse(s, e)

//#define ll long long int

#define CLR(p) memset(p, 0, sizeof(p))

#define mset(p, v) memset(p, v, sizeof(p))

#define ALL(c) c.begin(), c.end()

#define SZ(c) (int)c.size()

#define pb(x) push\_back(x)

//#define vs vector<string>

//#define vi vector<int>

//#define vii vector< pair<int, int> >

//#define vvi vector< vi >

#define fs first

#define sc second

#define MP(x, y) make\_pair(x, y)

//#define pii pair< int, int >

//#define psi pair< string, int >

//#define pq priority\_queue

**typedef** long long ll**;**

**typedef** pair**<**int**,**int**>** pii**;**

**typedef** vector**<**int**>** vi**;**

**typedef** vector**<**vi**>**vvi**;**

**typedef** vector **<** pair**<**int**,** int**>** **>** vii**;**

**typedef** vector**<**string**>** vs**;**

**typedef** map**<**string**,**int**>** msi**;**

**typedef** map**<**int**,**int**>**mii**;**

#define LOG(x,BASE) (log10(x)/log10(BASE))

#define EQ(a,b) (fabs(a-b)<ERR)

#define csprint printf("Case %d: ", ++t);

#define PI acos(-1)

#define ERR 10E-5

const int INF **=** 0x7f7f7f7f**;**

#define MAX 1005

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///

int n**,** m**,** col**[**MAX**],** par**[**MAX**];**

vi adj**[**MAX**],** edge**[**MAX**];**

bool matching**(**int from**)**

**{**

int to**,** i**;**

REP**(**i**,** SZ**(**edge**[**from**]))**

**{**

to**=** edge**[**from**][**i**];**

**if(**col**[**to**])** **continue;**

col**[**to**]=**1**;**

**if(**par**[**to**]==-**1 **||** matching**(**par**[**to**]))**

**{**

par**[**to**]=** from**;**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

int BPM**()**

**{**

int i**,** j**,** cnt**=**0**;**

**for(**i**=**1**;** i**<=**n**;** i**++)**

**{**

CLR**(**col**);**

**if(**matching**(**i**))**

cnt**++;**

**}**

**return** cnt**;**

**}**

void dfs**(**int from**)**

**{**

// deb(from, "start");

int i**,** to**;**

REP**(**i**,** SZ**(**adj**[**from**]))**

**{**

to**=** adj**[**from**][**i**];**

**if(**col**[**to**]==-**1**)**

**{**

col**[**to**]=** **(**col**[**from**]^**1**);**

// bug(col[to]);

dfs**(**to**);**

**if(**col**[**from**])**

edge**[**from**].**pb**(**to**);**

**else**

edge**[**to**].**pb**(**from**);**

**}**

**}**

**}**

void create\_edge**()**

**{**

int i**,** j**,** k**;**

**for(**i**=**1**;** i**<=**n**;** i**++)**

**{**

**if(**col**[**i**]==-**1**)**

**{**

col**[**i**]=**1**;**

dfs**(**i**);**

**}**

**}**

**return;**

**}**

int main**()**

**{**

//READ("in.txt");

//WRITE("out.txt");

int i**,** j**,** k**,** result**,** t**=**0**,** tcase**,**u**,** v**,** cnt**;**

cin**>>**tcase**;**

**while(**tcase**--)**

**{**

scanf**(**"%d %d"**,** **&**n**,** **&**m**);**

REP**(**i**,** m**)**

**{**

scanf**(**"%d %d"**,** **&**u**,** **&**v**);**

adj**[**u**].**pb**(**v**);**

adj**[**v**].**pb**(**u**);**

**}**

mset**(**col**,** **-**1**);**

create\_edge**();**

mset**(**par**,** **-**1**);**

result**=**BPM**();**

printf**(**"Case %d: %d\n"**,** **++**t**,** **(**n**-**result**)** **);**

REP**(**i**,** n**+**1**)**

**{**

edge**[**i**].**clear**();**

adj**[**i**].**clear**();**

**}**

**}**

**}**

/\*

**OJ: Light OJ**

**Name Fazle Rabby Sourav**

**Problem Link:** [**http://www.lightoj.com/volume\_showproblem.php?problem=1150**](http://www.lightoj.com/volume_showproblem.php?problem=1150)

**Problem Name: 1150 - Ghosts!**

**Rank: 26**

\*/

/\*

**Solution Method:**

I have used Matching algorithm with Binary search and Breadth-first search to solve this problem.

For each ghost we have to find the minimum distance between ghost and human. it can be done by BFS. then we need to create edge

from the ghost to reachable human. the weight of the edge will be the minimum distance. Then I have used B- search to find the

minimum time to frigten\reach every human. with the help of binary search, for every mid value [time] I have run Matching algo to

check that if it is possible to reach every human in that time or not. if that is possible we decrease upper limit to mid value.

or that's not possible then we change lower limit to mid\_value+1. thus we can find the minimum required time.

\*/

/\*

**Complexity Analysis:**

time complexity of BFS is O(|V|+|E) . BPM with binary search its need O(lgN) and Fo BPM its nedded O(V\*(V+E)).

So the Complexity of the solution will be O( (|V|+|E|) + lgN\*(V\*(V+E))) or O(lgN\*(V\*(V+E)))

\*/

#define REP(i,n)for (i=0;i<n;i++)

#define FOR(i,p,k) for (i=p; i<k;i++)

#define FORE(i, p, k) for(i=p; i<=k; i++)

#define FOREACH(it,x) for(\_\_typeof((x).begin()) it=(x.begin()); it!=(x).end(); ++it)

#define FORD(i,n) for(i=n;i>=0;i--)

#define READ(f) freopen(f, "r", stdin)

#define WRITE(f) freopen(f, "w", stdout)

#define REV(s, e) reverse(s, e)

#define CLR(p) memset(p, 0, sizeof(p))

#define mset(p, v) memset(p, v, sizeof(p))

#define ALL(c) c.begin(), c.end()

#define SZ(c) (int)c.size()

#define pb(x) push\_back(x)

///#define type

#define ll long long int

#define vs vector<string>

#define vi vector<int>

#define vii vector< pair<int, int> >

#define pii pair< int, int >

#define psi pair< string, int >

#define fs first

#define sc second

#define MP(x, y) make\_pair(x, y)

#define pq priority\_queue

#define LOG(x,BASE) (log10(x)/log10(BASE))

#define EQ(a,b) (fabs(a-b)<ERR)

#define csprint printf("Case %d: ", ++t);

#define PI acos(-1)

#define ERR 10E-5

const int INF **=** 0x7f7f7f7f**;**

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///

int n**,** par**[** 150 **],** dist**[**MAX**][**MAX**];**

bool taken**[**MAX**][**MAX**],** col**[**150**];**

char grid**[**30**][**30**];**

vii edge**[**MAX**];**

vii ghost**,** human**;**

void BFS**(**int st\_x**,** int st\_y**)**

**{**

int i**,** to\_x**,** to\_y**,** n\_x**,** n\_y**,** from\_x**,** from\_y**;**

queue **<**pair **<**int **,** int **>** **>** Q**;**

CLR**(**taken**);**

CLR**(**dist**);**

Q**.**push**(**MP**(**st\_x**,** st\_y**));**

dist**[**st\_x**][**st\_y**]=**0**;**

**while(!**Q**.**empty**())**

**{**

from\_x**=** Q**.**front**().**fs**;**

from\_y**=** Q**.**front**().**sc**;**

Q**.**pop**();**

REP**(**i**,** 4**)**

**{**

to\_x**=** from\_x**+**xx**[**i**];**

to\_y**=** from\_y**+**yy**[**i**];**

**if(**to\_x**>=**0 **&&** to\_x**<**n **&&** to\_y**>=**0 **&&** to\_y**<**n **&&** grid**[**to\_x**][**to\_y**]!=**'#' **&&** taken**[**to\_x**][**to\_y**]==false)**

**{**

Q**.**push**(**MP**(**to\_x**,** to\_y**));**

dist**[**to\_x**][**to\_y**]=** dist**[**from\_x**][**from\_y**]+**1**;**

taken**[**to\_x**][**to\_y**]=true;**

**}**

**}**

**}**

**return;**

**}**

bool dfs**(**int from**,** int val**)**

**{**

**if(**col**[**from**])** **return** **false;**

col**[**from**]=**1**;**

int i**,** to**,** w**;**

REP**(**i**,** SZ**(**edge**[**from**]))**

**{**

to**=** edge**[**from**][**i**].**fs**;**

w**=** edge**[**from**][**i**].**sc**;**

**if(**w**<=**val**)**

**{**

**if(**par**[**to**]==-**1 **||** dfs**(**par**[**to**],** val**))**

**{**

par**[**to**]=** from**;**

**return** **true;**

**}**

**}**

**}**

**return** **false;**

**}**

int BPM**(**int val**)**

**{**

int i**,** k**,** cnt**=**0**;**

mset**(**par**,** **-**1**);**

REP**(**i**,** SZ**(**ghost**))**

**{**

CLR**(**col**);**

**if(**dfs**(**i**,** val**))**

cnt**++;**

**}**

**return** cnt**;**

**}**

int main**()**

**{**

//READ("in.txt");

//WRITE("out.txt");

int i**,** j**,** k**,** result**,** t**=**0**,** tcase**,** m**,** cnt**,** max\_t**;**

cin**>>**tcase**;**

**while(**tcase**--)**

**{**

cin**>>**n**;**

REP**(**i**,** n**)**

scanf**(**"%s"**,** grid**[**i**]);**

REP**(**i**,** n**)**

**{**

REP**(**k**,** n**)**

**{**

**if(**grid**[**i**][**k**]==**'G'**)** ghost**.**pb**(**MP**(**i**,** k**));**

**if(**grid**[**i**][**k**]==**'H'**)** human**.**pb**(**MP**(**i**,** k**));**

**}**

**}**

max\_t**=** **-**1**;**

REP**(**i**,** SZ**(**ghost**))**

**{**

BFS**(**ghost**[**i**].**fs**,** ghost**[**i**].**sc**);**

REP**(**k**,** SZ**(**human**))**

**{**

int x**,** y**;**

x**=** human**[**k**].**fs**;**

y**=** human**[**k**].**sc**;**

**if(**dist**[**x**][**y**])**

**{**

edge**[**i**].**pb**(**MP**(**k**+**MAX**,** **(** **(**dist**[**x**][**y**]\***2**)+**2**)** **));**

// deb(ghost[i].fs, ghost[i].sc, x, y, dist[x][y]);

// max\_t= max(max\_t, ( (dist[x][y]\*2)+2));

**}**

**}**

**}**

int upper**=** n**\***n**,** lower**=**0**,** mid**,** x**;**

**while(**lower**<**upper**)**

**{**

mid**=** **(**lower**+**upper**)/**2**;**

// bug(mid);

x**=**BPM**(**mid**);**

// bug(x);

**if(**x**==**SZ**(**human**))** upper**=** mid**;**

**else** lower**=** mid**+**1**;**

**}**

printf**(**"Case %d: "**,** **++**t**);**

**if(**lower**==** n**\***n**)**

printf**(**"Vuter Dol Kupokat\n"**);**

**else**

printf**(**"%d\n"**,** lower**);**

// deb(SZ(human), max\_t, mid);

ghost**.**clear**();**

human**.**clear**();**

// CLR(grid);

REP**(**i**,** MAX**)**

**{**

edge**[**i**].**clear**();**

**}**

**}**

**}**

/\*

**OJ: Light OJ**

**Name Fazle Rabby Sourav**

**Problem Link:** [**http://www.lightoj.com/volume\_showproblem.php?problem=1152**](http://www.lightoj.com/volume_showproblem.php?problem=1152)

**Problem Name: 1152 - Hiding Gold**

**Rank: 43**

\*/

/\*

**Solution Method:**

If we consider every cell on grid as a node then this problem can be easily solved by matching algorithm. From problem statement

at most one dominoes can be placed between two vartical or horizontal adjacent cell.

At First, if there are gold on adjacent two cell, then we will create edge between them[we assum each cell as a node]. then simply

we run bipartite matching matching algorithm. the result is the number of maximum matching. but if

it is noticed carefully, then it will be clear that we have count each matching twice [double counting problem]. as we create edge

between x to y as well as y to x. so the real matching number will be (result)/2 [half of the result].

total cell minus maximum matching number is the desired ans [number of required dominos]

\*/

/\*

**complexity Analysis:**

to create necessary edge we need n\*m iteration. maximum edge can be E= 4\*(m\*n), vertex can be V=(m\*n);

The Runnig time of DFS is O(E+V) where E is the number of edges and V is is the number of Vertices.

so the complexity will be O(V\*(V+E)) or O((m\*n)^2)

\*/

#define REP(i,n)for (i=0;i<n;i++)

#define FOR(i,p,k) for (i=p; i<k;i++)

#define FORE(i, p, k) for(i=p; i<=k; i++)

#define FOREACH(it,x) for(\_\_typeof((x).begin()) it=(x.begin()); it!=(x).end(); ++it)

#define FORD(i,n) for(i=n;i>=0;i--)

#define READ(f) freopen(f, "r", stdin)

#define WRITE(f) freopen(f, "w", stdout)

#define REV(s, e) reverse(s, e)

#define ll long long int

#define CLR(p) memset(p, 0, sizeof(p))

#define mset(p, v) memset(p, v, sizeof(p))

#define ALL(c) c.begin(), c.end()

#define SZ(c) (int)c.size()

#define pb(x) push\_back(x)

#define vs vector<string>

#define vi vector<int>

#define vii vector< pair<int, int> >

#define fs first

#define sc second

#define MP(x, y) make\_pair(x, y)

#define pii pair< int, int >

#define psi pair< string, int >

#define pq priority\_queue

#define csprint printf("Case %d: ", ++t);

#define PI acos(-1)

#define ERR 10E-5

const int INF **=** 0x7f7f7f7f**;**

#define MAX 420

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///

int row**,** colum**,** col**[**MAX**],** par**[**MAX**];**

string str**;**

vs grid**;**

vi edge**[**MAX**],** gold**;**

bool dfs**(**int from**)**

**{**

int i**,** j**,** k**,** to**;**

**for(**i**=**0**;** i**<**SZ**(**edge**[**from**]);** i**++)**

**{**

to**=** edge**[**from**][**i**];**

// bug(to);

**if(**col**[**to**])** **continue;**

col**[**to**]=**1**;**

**if(**par**[**to**]==-**1 **||** **(** col**[**par**[**to**]]==**0 **&&** dfs**(**par**[**to**])** **))**

**{**

par**[**to**]=** from**;**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

int BPM**()**

**{**

int i**,** j**,** k**,** n**,** cnt**=**0**;**

mset**(**par**,** **-**1**);**

n**=** SZ**(**gold**);**

**for(**i**=**0**;** i**<**n**;** i**++)**

**{**

CLR**(**col**);**

**if(**dfs**(**gold**[**i**]))**

cnt**++;**

**}**

**return** cnt**;**

**}**

int main**()**

**{**

int i**,** j**,** k**,** result**,** t**=**0**,** tcase**,** n**,** m**,** cnt**,** from**,** to**,** x**,** y**;**

cin**>>**tcase**;**

**while(**tcase**--)**

**{**

scanf**(**"%d %d"**,** **&**row**,** **&**colum**);**

REP**(**i**,** row**)**

**{**

cin**>>**str**;**

grid**.**pb**(**str**);**

**}**

REP**(**i**,** row**)**

**{**

REP**(**k**,** SZ**(**grid**[**i**]))**

**{**

**if(**grid**[**i**][**k**]** **==**'\*'**)**

**{**

from**=** **(**i**\***colum**)+(**k**+**1**);**

gold**.**pb**(**from**);**

REP**(**j**,** 4**)**

**{**

x**=** i**+**xx**[**j**];**

y**=** k**+**yy**[**j**];**

**if(**x**>=**0 **&&** x**<**row **&&** y**>=**0 **&&** y**<**colum **&&** grid**[**x**][**y**]==**'\*'**)**

**{**

to**=** **(**x**\***colum**)+(**y**+**1**);**

edge**[**from**].**pb**(**to**);**

**}**

**}**

**}**

**}**

**}**

int res**=** BPM**();**

result**=** SZ**(**gold**)-(**res**/**2**);**

printf**(**"Case %d: %d\n"**,** **++**t**,** result**);**

//clear edges

gold**.**clear**();** grid**.**clear**();**

REP**(**i**,** MAX**)**

**{**

edge**[**i**].**clear**();**

**}**

**}**

**}**