Problem A: Hexagonal Packing

With some observation, it's obvious that the hexagonal structure will touch the circumference of the circle to maximize the utilized area. Let, the side length of a hexagon be *a*. Then, we can say, $a^2 + 3(2n-1)^2a^2 = 4r^2$. Solving this equation we get *a*. Now, all we need to do is print the area of a regular hexagon with side length equal to *a*.

You may also use binary search.

Problem B: Game of Rocks

This problem is a variant of the "Maximum sum rectangle in 2D matrix" problem.

Let's convert every rock in the grid into a 0 and every empty cell into a 1. Now, we need to find the maximum perimeter of a rectangle enclosed by 0's.

We pre-calculate the prefix sum of each row and each column.

Now, a possible naive solution is to choose a top-left and a bottom-right corner for a rectangle and check if it satisfies our conditions. It has a $O(N^4)$ complexity, obviously too slow for this problem.

To reduce the complexity, we can do the following:

We choose the top and bottom row of the rectangle. We calculate the cumulative sum of each column. Now, only the columns with a zero cumulative sum qualify as the left/right side of the rectangle. Once we have listed these columns, it only takes a $O(N)$ search to find the best result for the chosen pair of rows. The complexity of this solution is $O(N^3)$.

Please refer to the setter's solution for implementation details.

Code:

```
#include <bits/stdc++.h>
using namespace std;

#define vlong long long
#define pii pair<int,int>
```

```
#define pll pair<LL,LL>
#define ff first
#define ss second
#define MP make_pair
#define pb push_back
#define MOD 100000000711

int n, m, sum[300];
int grd[300][300];
char str[300];

int main() {

    //freopen("input.txt", "r", stdin);

    int t = 1, tc=0;

    scanf("%d", &t);
    while (tc < t) {
        tc++;
        //cerr << "\nCase -> " << tc << " ------------\n" << endl;

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

        for (int i=0; i<n; i++) {
            scanf("%s", str);
            for (int j=0; j<m; j++) {
                if (str[j] == '.') {
                    grd[i][j] = 1;
                }
                else {
                    grd[i][j] = 0;
                }
                //cerr << grd[i][j] << " ";
            } //cerr << endl;
        } //cerr << endl;

        int best = 0, area = 0, flag, st, len, wid, prm, tmpa;

        for (int i=0; i<n; i++) {
            for (int j=0; j<m; j++) {
                sum[j] = grd[i][j];
            }
            for (int j=i+1; j<n; j++) {
                for (int k=0; k<m; k++) {
                    sum[k] += grd[j][k];
                }
                flag = 0;
                for (int k=0; k<m; k++) {
```

```
                    if (grd[i][k] == 1 || grd[j][k] == 1) {
                        flag = 0;
                    }
                    else {
                        if (sum[k] == 0) {
                            if (flag == 1) {
                                len = j-i+1;
                                wid = k-st+1;
                                prm = 2*(len+wid) - 4;
                                tmpa = (len-2) * (wid-2);
                                if (tmpa <= 0) {
                                    continue;
                                }
                                if (prm < best) {
                                    continue;
                                }
                                else if (prm == best) {
                                    area = max(area,tmpa);
                                }
                                else {
                                    best = prm;
                                    area = tmpa;
                                }
                            }
                            else {
                                flag = 1;
                                st = k;
                            }
                        }
                    }
                }
            }
        }

        if (area > 0) {
            printf("%d %d\n", best, area);
        }
        else {
            printf("No solution\n");
        }
    }

    return 0;
}
```

Problem C: Rectangle Division

Let’s think about another problem before solving this. What is the easiest way to divide a rectangle into two equal(area wise) part? An easy way is to draw a diagonal.
Now think about the diagonal. You can just rotate the diagonal around 360 degree(assuming mid-point of diagonal as origin) and get infinite amount of line and every line will divide the rectangle into two equal(area wise) part.

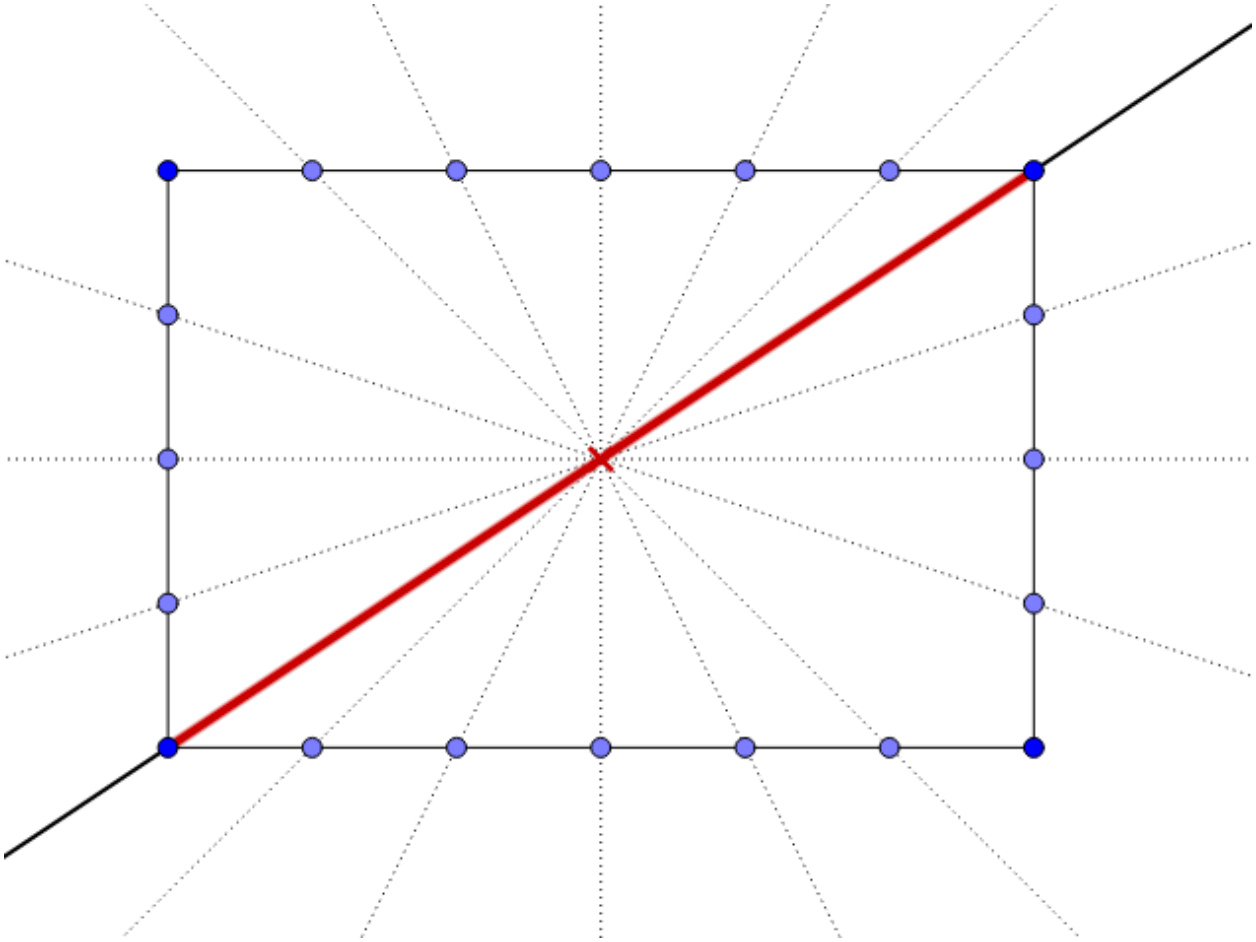


Fig : Here red line is the diagonal. All the dotted lines are some of the instances of the diagonal while rotating around 360 from mid-point of the the diagonal .

Now From this observation, we can solve the problem easily. Create a line(not segment) with given point and the mid point of any diagonal. The points which intersect the sides(as a segment) of the rectangle are the solution. It was told in the problem “there will never be infinite amount of solution” means the given point will never be the midpoint of the diagonal.

Problem D: Chemical Reaction II

For this problem you need to create a bridge with two 3D points. Assume this bridge as a segment. Now to connect any point to the bridge(segment) you need to calculate the shortest length from the point to segment(bridge) in 3D cartesian coordinate system which turns into a problem like “point to segment distance in 3D”.
As N is very small, you can use ternary search to calculate “point to segment distance in 3D” instead of using geometric laws.

The calculation of the cost function is straightforward.

- To create a bridge = Square root distance between 2 points.
- To add a point to the bridge = Point to segment distance(calculated by ternary search or geometric laws)

Now there could be many combination of points to create one or multiple bridges and connect points to the bridges if needed to minimize the result. We can use a bitmask dp(as N is very small) to calculate the answer.
One way to solve with bitmask dp : State will be just an integer number used as a mask where 0’th to (n-1)’th bit of that integer represent chemical compound’s index number(position in the array). If the compound is already used, the position(bit position) of that compound in the mask will be set to one. Now in each state, try all pairs of unused compounds to create a bridge. Say you have 5(0,1,2,3,4) compounds(each number represents a compound). 4(0,1,2,3) of them are unused. There are 6 different pair({0,1},{0,2},{0,3},{1,2},{1,3},{2,3}) of points here. Say for one case you have created a bridge with 0 and 1. You have remaining 2 and 3 which are also unused. So there are 4 (try to add both 2 and 3, try only 2, try only 3, do not add any compound) ways to add 2,3 to the bridge. Try all possible of those. And finally try this process with all possible pair of points. Some small details are not discussed here. While implementing dynamic programming be careful about some points,

- To create a bond, at least two compound needs to create a bridge.
- All the compounds will participate to at least one bond
- No two chemical bonds will interfere with each other while doing the simulation of the chemical reaction.

Problem E: A Multiplayer Action Game

Let's define some symbols first.

i_1 = initial position of first n-D sphere
i_2 = initial position of second n-D sphere
v_1 = velocity of first n-D sphere
v_2 = velocity of second n-D sphere
r_1 = radius of first n-D sphere
r_2 = radius of second n-D sphere

Now, at any time t the position of first sphere will be $i_1 + v_1t$ and second sphere will be at $i_2 + v_2t$. Vector arithmetic rules apply everywhere. Distance between any two n- points can be found by extending the Pythagorean theorem to n^{th} dimension. Now, we need to find the time when the distance between two spheres will be $abs(r_1-r_2)$. As the time vs distance graph will be a parabolic one, so may have at most 2 time stances where distance between 2 sphere is $abs(r_1-r_2)$. One will be before reaching the minimum distance and one after. We need to find the first one. We can use calculus or Ternary search and Binary search to solve the problem. We'll discuss the solution using ternary search and binary search. Now to solve the problem, we first use ternary search to find the minimum distance in the given time range and the time when it happens. If the minimum distance between two sphere is less than $abs(r_1-r_2)$ for $0 \leq t \leq 10^5$ then the solution exists, Otherwise print -1. If solution exists then, we do a binary search between 0 and the time when the distance is minimum to find the first time stance when the distance was $abs(r_1-r_2)$. Worth to mention, if $r_1=r_2$ then no sphere eats another one, as one is not bigger than the other.

Problem F: Gaaner Koli

This problem can be solved using grundy numbers and bitmask DP. Observation: we only need the starting character and the unique characters of any string. We'll keep last character used and mask for unused strings in our state. At any state (ch,mask) we can go to any character of any unused song that starts with ch. Say, that character is x and our selected string is y. Then, our next state will be (x,mask|(1<<y)). Now, the grundy value of (ch,mask) is the mex value for

all possible (x,mask|(1<<y)). We'll try to start with each of the n songs present. If for any song the first player can win then we print YES, otherwise NO.

Problem G: Gretchen and Strange Function

To understand what the **H_function** does, we can simulate the steps for different cases.

But simulation takes too long to answer all the queries. Observing the nature of the function closely, you can understand that, C can only have the value 1 and -1 for this problem. Now, given a starting value A, and another value B,

* If C is negative, the function finds the smallest number $curA > 0$, that is a term of the arithmetic series with A as the first term and $-B$ as the difference between two consecutive terms.

* If C is positive, the function finds the highest number $curA \leq N$, that is a term of the arithmetic series with A as the first term and B as the difference between two consecutive terms.

Then it calls the next instance of itself with $A = curA$, $B = B + 1$ and $C = -C$.

We have to find the value of $curA$ when $B = N$.

Simulation is too slow, because we have to repeat the same calculations over and over. We can try a DP approach, memorizing results for all possible pair (A, B) in complexity $O(N^2)$ and then answering the queries in $O(1)$ [Online method]. But it would take too much space to memorize all the results. (Some of the solutions used "short" data types to reduce the space required. Such solutions passed marginally.)

So, instead we can have a bottom up approach to our DP solution. Since the results of the queries with some particular $B = i$ depends on the results of the queries with $B = i + 1$, we

only need to keep track of the last stage of calculations. We can write an iterative DP solution now.

Although we reduced the space being used, the overall complexity remains $O(N^2)$.

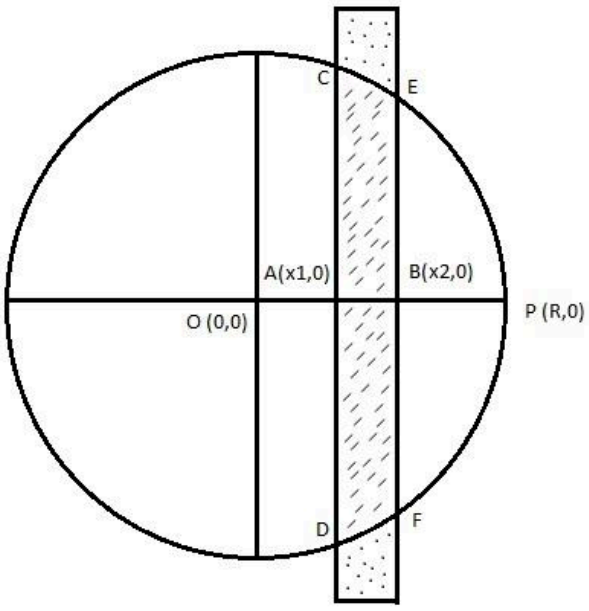
Code:

```
#include <bits/stdc++.h>
using namespace std;

#define MAXN 10004
#define MAXT 1000006

#define pii pair<int,int>
#define ff first
#define ss second

vector< pii > qry[MAXN];
int dp[MAXN][2], tmp[MAXN][2], res[MAXT];

int main() {

    int t, tc=0, N, A, B;

    scanf("%d %d", &t, &N);
    while (tc < t) {
        tc++;

        scanf("%d %d", &A, &B);
        qry[B].push_back ( make_pair(A,tc) );
                        // Collect all queries with the same 'B' in the same place
    }

    for (int i=1; i<=N; i++) {          // Results for A = N+1
        dp[i][0] = dp[i][1] = i;
    }
```

```
    for (int i=N; i>=1; i--) {
                        // Bottom up DP approach to calculate result for B = i
        for (int j=1; j<=N; j++) {
            int jmodi = j%i, nmodi = N%i;

            int curA1 = N - nmodi + jmodi;
            if (curA1 > N) curA1 -= i;
                        // Where would I stop if I started at A=j and went forward

            int curA0 = jmodi;
            if (curA0 == 0) curA0 += i;
                        // Where would I stop if I started at A=j and went backward

            tmp[j][1] = dp[curA1][0];
            tmp[j][0] = dp[curA0][1];
        }

        for (int k=0; k<qry[i].size(); k++) {  // answer queries
            int a = qry[i][k].ff;
            int q = qry[i][k].ss;
            res[q] = tmp[a][1];
        }

        for (int j=1; j<=N; j++) {
            dp[j][1] = tmp[j][1];
            dp[j][0] = tmp[j][0];
        }
    }

    for (int i=1; i<=t; i++) {
        printf("%d\n", res[i]);
    }

    return 0;
}
```

Problem H: Infinite Game

The area covered by Oditi is a circular area. And the area covered by Sam and Ron is actually a rectangle. So we can calculate both of these areas using basic geometry formula. The main part is how to calculate the area of Tom which is the intersected area of the circle and that rectangle.



In this picture let's we want to calculate the lined area of ACP. The radius of the circle is R and the equation of the circle is x^2 + y^2 = R^2. So the area of ACP is $\int_{x1}^{R} \sqrt{(R^2 - x1^2)}$dx. Then the intersection area of CD with the circle is CEPFD = 2*ACP.

Similarly the area of BEP is $\int_{x2}^{R} \sqrt{(R^2 - x2^2)}$dx. and the intersection area of EF with the circle is EPF = 2*BEP.

Finally the intersected area of the circle and the rectangle will be CEFD = CEPFD - EPF. Using the integration we can find a general form of that equation which is R (x2 * sin (acos (x2/R)) - x1 * sin (acos (x1/R))) - R^2 * acos (x2/R) + R^2 * acos (x1/R). Solving this we get the area Tom gets. Then we can easily find the area Oditi, Sam and Ron gets using the area formula of circle & rectangle.

Note : Both or any of the straight line created by Sam and Tom may be outside the circle. So handle these cases carefully.

Solution : http://pastebin.com/v7m59Umc

Problem I : Flyover in Twinland

First of all we can run a DFS to find all the islands and then mark them with distinct colors using a map or array. We will also have to count the number of cities in each of the island. Let C be the number of distinct islands in twinland. As we only need to consider those islands where the number of cities are greater than 100, C will be at most 2500. Complexity of this part is O (N*M).

Then we can build a Graph using a nested loop. For each pair of island we can find both the construction cost and the salary of the engineer's using two pointer. Then we will add that edge in the Graph. Complexity of this part is O (C^2).

Now let's fix the maximum salary of the engineer to K. What will be the total construction cost? Definitely we can't use any edge whose cost is greater than K. So considering all the edges whose cost is not greater than K we can find the minimum spanning tree of the Graph. The total construction cost will be the summation of the cost of the edges of that mst.

But if we can't connect all the islands using those edges then surely we need some more edges whose cost are greater than K to connect the islands. And if we can connect all the islands using those edges then it may also be possible to connect them by decreasing the value of K and then considering the edges not greater than K. So we can use binary search here for the

maximum salary we have to minimize. Then for that maximum salary we can find the total construction cost. Complexity of this part is O (C^2 * log(C^2)).

Solution : http://pastebin.com/bVyVHJ6G

Problem J: Ghosh vs Datta

In order to solve this problem, first imagine that you have just one single barricade in the entire graph. If this is the case, then you have to delete that edge with the barricade, and run the flow on the remaining network from both the sources. After that, the remaining part is trivial. But since we can have more than one barricade, we can solve this problem by sequentially removing one edge at a time, and then run the flow from both the end to find the result for a particular query. For example, if we have 3 barricades on edges a, b and c, first we can run a flow after removing (a), then we can run another flow after removing (a , b) and finally we can run a third flow after removing all of the (a , b , c) edges. This would give us correct result, but would take time. A faster flow algorithm will give us a runtime of O($V^2 * E^2$).

Let's talk about improving our naive approach. Imagine that the first edge to be removed was (u , v) and in in some maximum flow f in the given graph G, f(u , v) amount of flow was augmented from the edge (u , v). If this edge is now deleted, then the new flow f' should follow the inequality f >= f'.

Now, first think that f = f'. In this case, the amount of flow f(u , v) can be passed from source to sink without using (u , v). To do that, we first need to cancel out the flow from source to (u , v) to sink. For that, we need to push f(u , v) amount of flow from u to source and another f (u , v) amount of flow from sink to v. This will be possible because f(u , v) was generated from source and ended up in sink. These two push operations will make sure that no flow has started from source and passed from (u , v) to reach sink. Once the effect of (u , v) has been canceled out, the edge (u , v) can be safely removed from the network, and flow algorithm can again be run from source to sink on the remaining residual graph until f = f'. Similar techniques can be followed when f > f'. If we follow this process for all the queries, the process can propagate at most 4f amount of flow across the whole network and the time is thus reduced. By this way, this problem can be solved online.

An interesting and a little easier solution came from bqi343, who processed the query in reverse order. First he deleted all the barricades from the network and run a flow on the final graph. After that, he added one barricade edge at a time in reverse order in the residual network. This approach will run in O($V^2 * E$).

Jackal_1586's solution: http://pastebin.com/99sUPKA9
Bqi343's solution: http://pastebin.com/ZAEDYxdTac

Problem K: Primary Key

For query 4.

Just use a global variable TOT which tracks each time a primary key has been added or removed from the system. Print the global variable TOT for this query.

For query 1 :

Add a new primary key. You can add it just by incrementing n by 1. Also increment global value TOT by 1.

For query number 3 :
Say the query comes with a value "VAL".
Initially no member is removed from the system. So each time a member is removed from the system and you need to keep track of that primary key. You can use a splay tree, redblack tree,

segment tree or equivalent data structure to ask,

Q(p) = “the number of primary key removed that are strictly smaller than p”.

So, now it’s clear if you need to find a number N for which, N-Q(N+1) = VAL. You can find the number N with a binary search as there is a binary property here(?).

For query 2 :
Say the query comes with a value “VAL”.
First check if the primary key is valid or not. It is invalid if it is greater than n or less than 1 or it is already removed.
If it is a valid and existing primary key, you need to add the primary key VAL to your tree based data structure on which you can ask the question Q().

Problem L: Rio and Inversion

The problem can be solved using sqrt decomposition technique. If you don’t know about it then you can learn it from here : http://e-maxx.ru/algo/sqrt_decomposition . We will separate the elements in some block and for each of the block we will calculate the number of inversions considering the elements in that block.

Besides we will also build a 2-d array where we will pre calculate for each of the element how many larger elements we have found so far. For example let’s that 2-d array be precalc. Then precalc[i][v] will be the number of elements greater than v from index 0 to i.

Now how to handle the first query? If we swap the elements at position u & v then we need to know the number of change of inversion happens. To do that we need the information of number of elements greater than A[u] and A[v] from index 0 to u and 0 to v. Similarly we also need the number of elements smaller than A[u] and A[v] from index u+1 to N-1 and v+1 to N-1. Using the

precalc array we can find that values in O(1) and then we just need to use some inclusion exclusions to get the result.

To handle the second query we have to iterate through all the blocks that lie inside the range from u to v. As we know the number of inversions for all the blocks so for each of the blocks that doesn't lie inside u and v we can iterate through all the values from 1 to 100 and maintain an array where we'll update the number of greater elements we found so far. We just need to add them in the result each time. We can handle this query in O($\sqrt{N}$ * 100).

Total Complexity : O(Q * $\sqrt{N}$ * 100)

Solution : http://pastebin.com/Kszvfvt7