Competitive Programming

From Problem 2 Solution in O(1)

Coordinate Compression

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Max people in room problem

- Given N pairs
  - each pair represents time someone enter/exit a room
- Given queries: query interval(start, end)
  - What is max # of people in room in the given interval?
- Input
  - (100, 150) (200, 300), (120, 450), (100, 300)
  - E.g. 1st user entered at t=100, leave at t=150
  - Assume the times are big (e.g. 10^12), but N < 100
- Query: 120 to 160

Max people in room problem

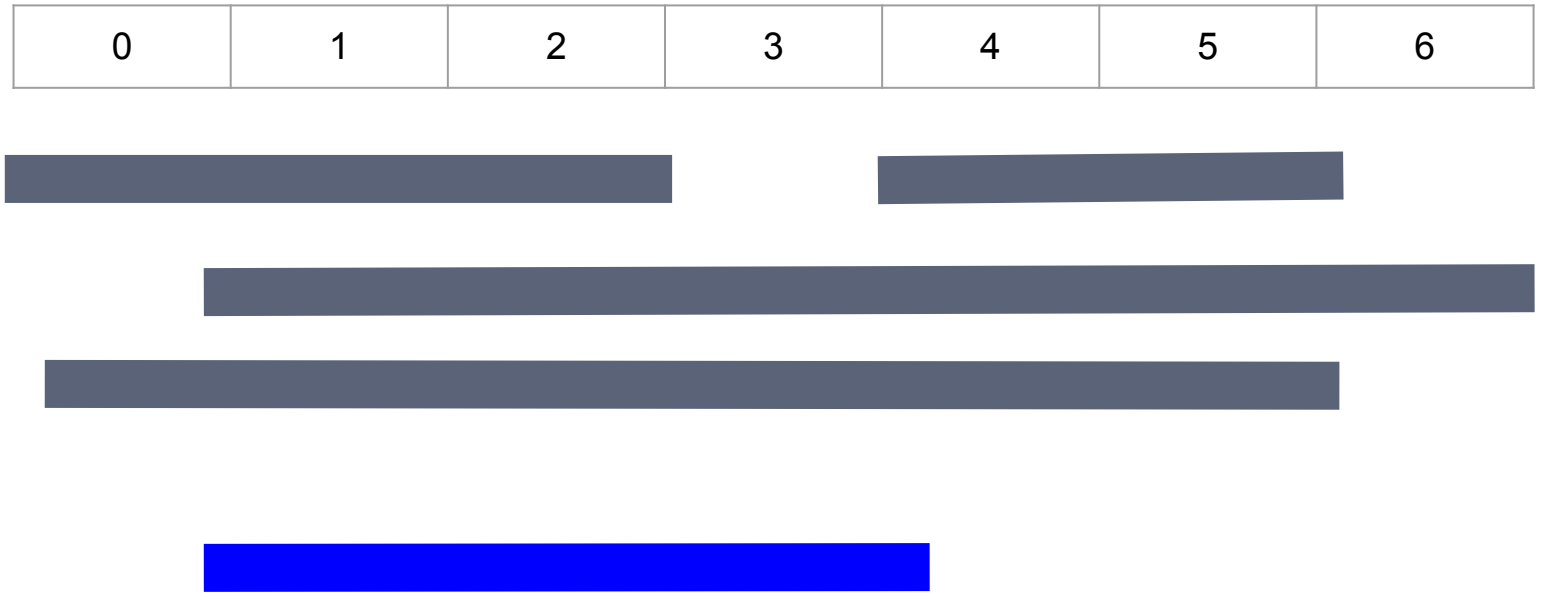
100	120	150	160	200	300	450

Max # of people in the room from time=120 to time 160 = 3 persons

Problem to solve

- Many algorithms can can solve the problem well, if t is small (e.g. 10^5)
- Coordinate Compression is to **simply** compress the values if we care about relative values NOT exact ones
- E.g. Sort all given distinct input values
  - 100, 120, 150, **160**, 200, 300, 450
- Map old values to the sorted order positions
  - 100=0, 120=1, 150 =2, 160=3, 200=4, 300=5, 450=6
- Recompute the input and solve problem

Max people in room problem



Max # of people in the room from time=1 to time 3 = 3 persons

Now, you can just count ++ for the intervals in an array and compute answer

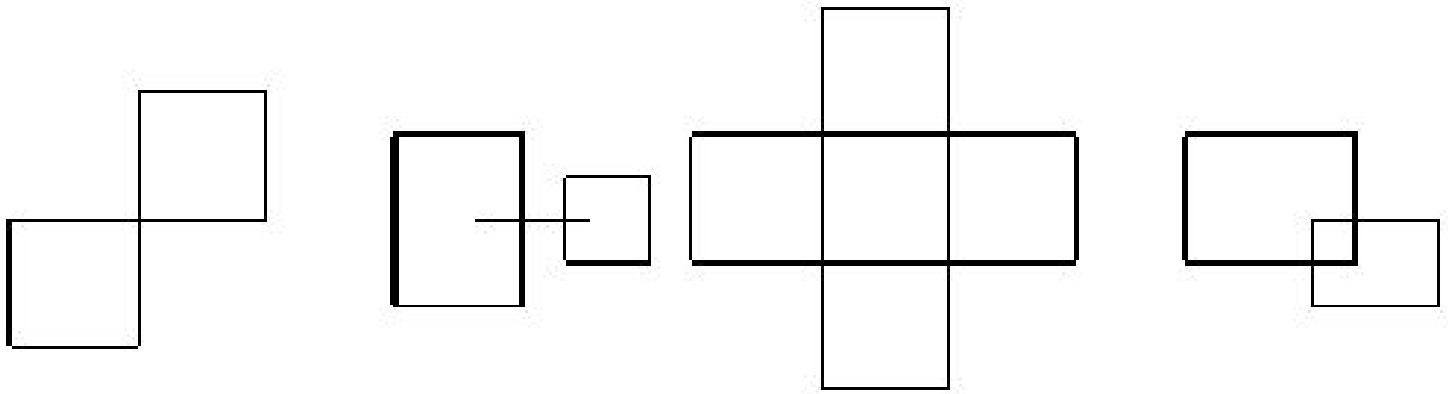
Coordinate Compression

- A simple technique we apply by re-assigning **smaller** values to given input
- Typically happens when actual values are not needed (or not fully needed)
- We just compress it, **preserving** orders
  - E.g. if p1 < p2 $\Rightarrow$ remains so
  - E.g. if line 1 above line 2 $\Rightarrow$ remains so
- Read all input/queries...sort them...reassign

Count Holes Problem

Given tin plates (rectangular e.g. 10000 x 10000)
Given serious of horizontal or vertical segment cuts in it

The cuts will cause **holes** in rectangular grid...How many?



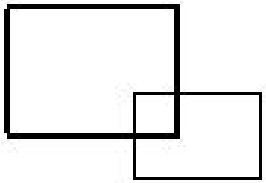
2 holes

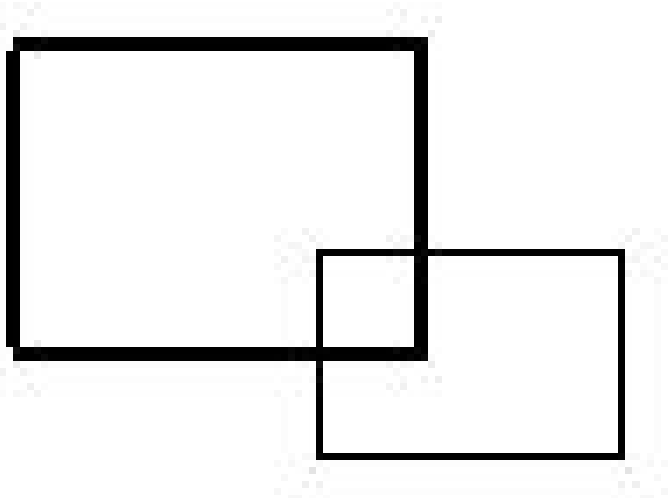
2 holes

1 hole

1 hole

Count Holes Problem





Assume we are given above 2 cases

- one of them has small 2 boxes...and one with 2 very big boxes (e.g. covering 10^18 area)
- The answer is **SAME**!
- If we can mark the given lines in 2D array...we can do BFS to compute holes

Key: **Compress coordinates** first

- Get all X's … compress them
- Get all Y's … compress them
- Reindex given input lines
- Mark lines in small 2D matrix...BFS to compute answer (avoid DFS, space large)

Coordinate Compression

```cpp
/*
 * {300, 10000, -5, -100, 300}
 *      start = 0, step = 1   ==>   2 3 1 0 2
 *      start = 2, step = 2   ==>   6 8 4 2 6
 */
vector<int> coordinate_compress(vector<int> x, int start = 0, int step = 1)
{
    set<int> unique(x.begin(), x.end());
    map<int, int> valPosMap;

    int idx = 0;
    for(auto val : unique)
    {
        valPosMap[val] = start + idx * step;
        ++idx;
    }
    for(auto &x_pos : x)
        x_pos = valPosMap[x_pos];
    return x;
}
```

- In 1D, also known as “Interval Compression”
- Call it once for X..once for Y
- Sometimes we need to return **map** itself (e.g. map query indices)
- Start and step is based on problem
- Generally, start with 2 and step with 2 is cool

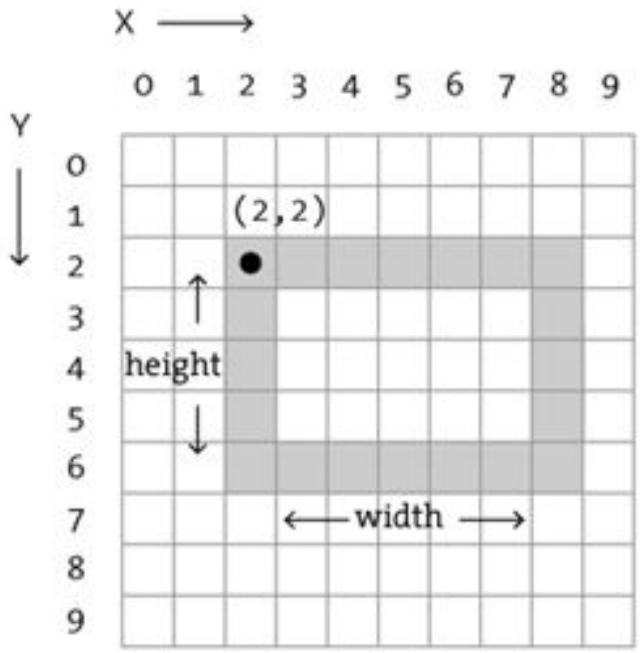
Coordinate Compression

```
void coordinate_compress(vector<int> &axis, vector<int> &iToV,
        map<int, int> &vToI, int start = 0, int step = 1) {
    for (auto &x_pos : axis)
        vToI[x_pos] = 0;

    iToV.resize(start + step * vToI.size());
    int idx = 0;

    for (auto &entry : vToI)
    {
        entry.second = start + step * idx;
        iToV[entry.second] = entry.first;
        ++idx;
    }
    for (auto &x_pos : axis)
        x_pos = vToI[x_pos];
}
```

Rectangle Vs Array Grid




```
rect(x,y,width,height)
```

Example:

```
rect(2,2,7,5)
```

Notice:
Lattice grid of **2x2**
corresponds to **3x3 array**

If we interested in area..and marked in 2d array, we will have larger area

solution: mark from start to end-1

```
// Mark rectangle (x1, y1) (x2, y2) in 2d array
// start inclusive and end exclusive
// line 0-2 is 2 units in array -> cell 0 and 1
for (int i = x1; i < x2; ++i)
        for (int j = y1; j < y2; ++j)
                grid[i][j] = 1;
```

Src: http://py.processing.org/tutorials/drawing/imgs/1.7.jpg

Overlapping Rectangles Area

- Given set of 2D axis aligned rectangles, find their area
  - Notice, we need original positions!
- Compress coordinates
  - Mark in 2D array
  - iterate on grid
  - If cell is covered, get its corresponding rectangle

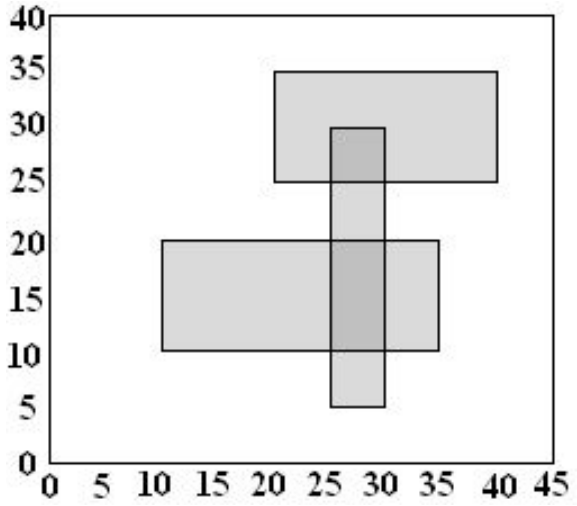
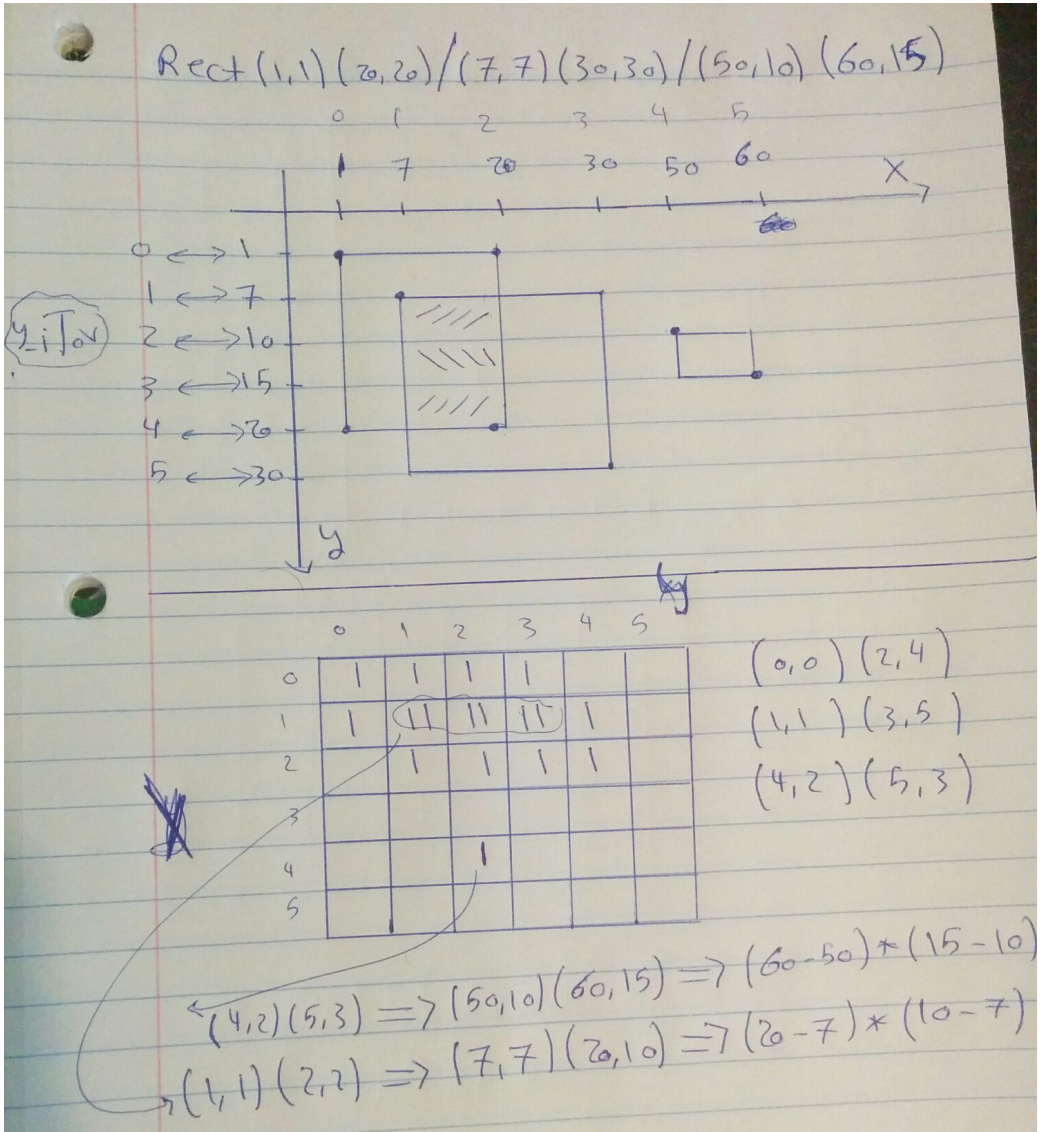


Photo Src: https://icpcarchive.ecs.baylor.edu/external/41/p4171.png

Overlapping Rectangles Area



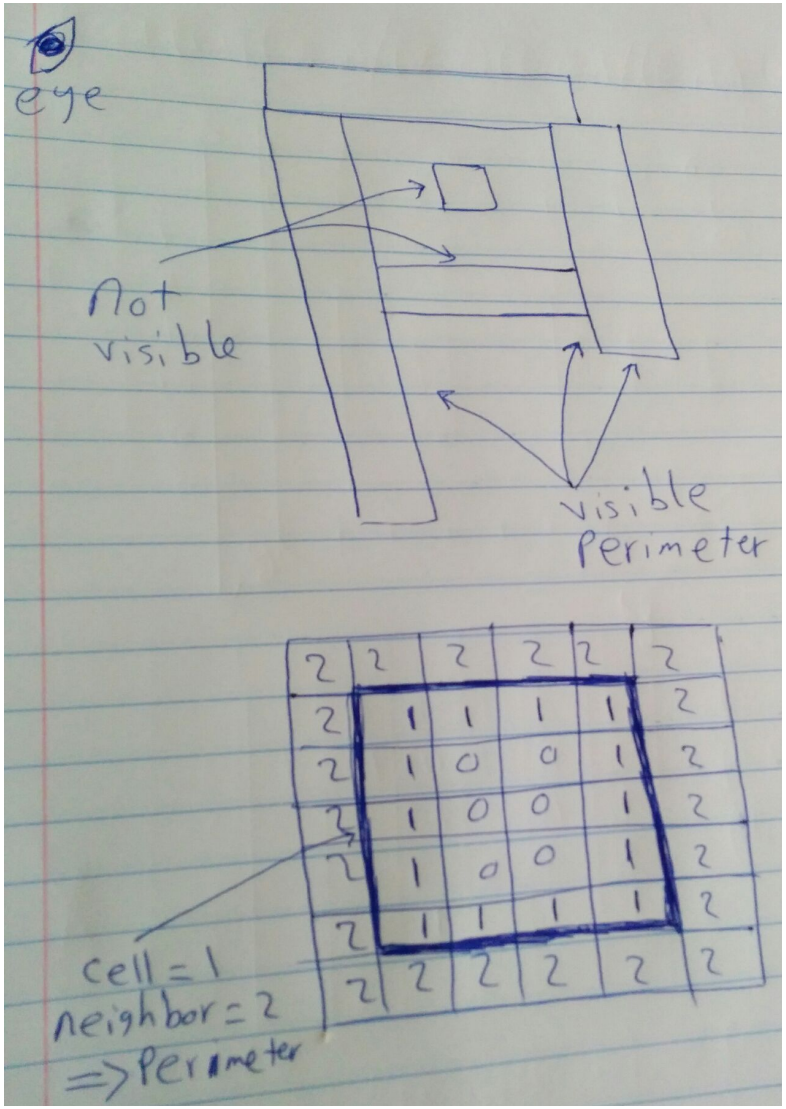
Overlapping Rectangles Area

```
// To calc area of a shape, iterate on compressed grid,
// i, +1 and j, j+1 forms 1x1 compressed rectangle
// x_iToV is the mapping of i to it original coordinate
int calArea(vector<int> & x_iToV, vector<int> & y_iToV) {
        int area = 0;
        rep(i, grid) rep(j, grid[0])
            if (grid[i][j])
                area += (x_iToV[i+1] - x_iToV[i]) *
                        (y_iToV[j+1] - y_iToV[j]);
        return area;
}
```

Overlapping Rectangles Perimeter

- Simply find positions where (x, y) is marked and then for each 4 directions find (xs, ys) that is not marked.
  - Then we have side (in the direction) to cover
- What about **visible** perimeter?
  - If looking to shapes from outside, enclosed perimeter not considered
  - Let 1 be marked, 0 inclosed, 2 visible (but not marked)
  - Run BFS from any external point, mark them 2
  - If p = 1 and neighbour = 2 => consider perimeter
- What about volumes? same logic

Overlapping Rectangles Perimeter

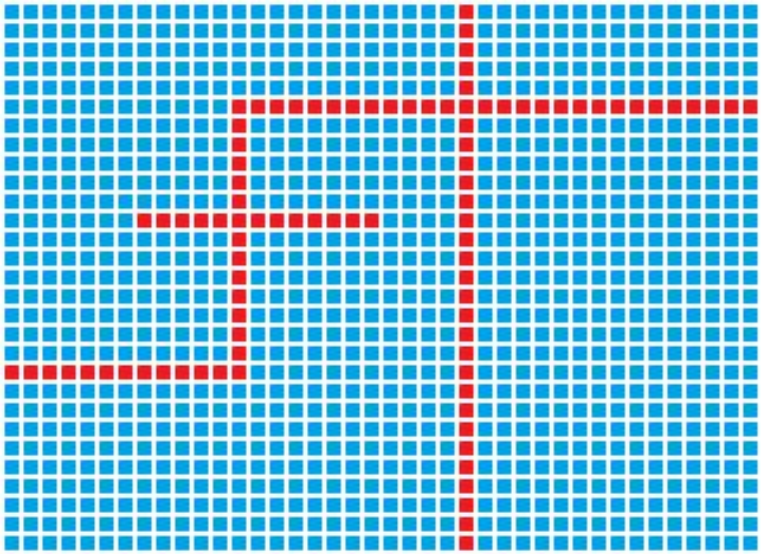


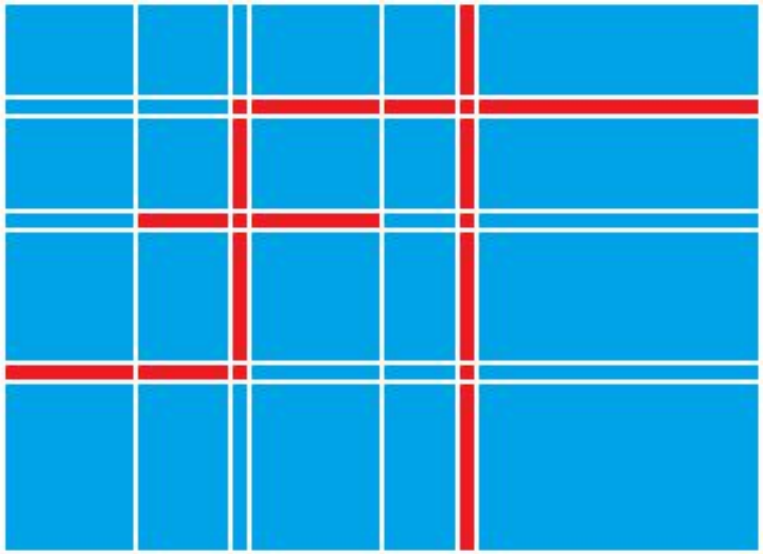
3D cubes volume surface

```
lp(i, MX)  lp(j, MY) lp(k, MZ) {
    ll x = ix[i + 1] - ix[i], y = iy[j + 1] - iy[j], z = iz[k+ 1] - iz[k];
    // (1 -> occupied), (0 -> space enclosed), (2, visible, not marked)
    if (grid[i][j][k] != 2)
    {
        volume += x * y * z;
        continue;
    }

    lp(d, 6) {
        int nx = i + dx[d], ny = j + dy[d], nz = k + dz[d];
        if (nx >= 0 && ny >= 0 && nz >= 0 && nx < MX &&
            ny < MY && nz < MZ) {
            if (grid[nx][ny][nz] != 2) //one face outside and another inside
                surfacesArea += dx[d] ? y * z : dy[d] ? x * z : x * y;
        }
    }
}
```

Summary





Src: http://stackoverflow.com/questions/29528934/coordinate-compression

Summary

- Very large 1D/2D/3D inputs
- We only need relative positions
- Or relative positions are enough and map back to original values
- Start/Step between points based on problem
- Code may vary from problem to another

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً

Problems

- SRM(277-D1-1000), Live Archive (4787)
- UVA (308, 10864)
- 3289 Robots inside the Labyrinth
- 1019 Tin Cutter
- 870 Intersecting Rectangles
- 4171 Bulletin Board
- 4291 Sculpture
- https://www.google.com.au/?client=firefox-a&hs=gi#hl=en&safe=off&q=coordinate+compression+site%3Atopcoder.com&meta=&gws_rd=ssl