IOI2016 - Alien

Jeremy Chow

30th June 2018

Motivation

- Practice on a past IOI problem
- Review on common DP optimizations
  - Advanced session – Dynamic Programming (III)
- Introduction on a new DP optimization trick
  - “Alien Trick” or “wqs 二分”

Problem Statement

- Given N points on a M * M grid
- A photo on a grid is a square
  - Such that two opposite corners of the square both lie on the main diagonal of the grid
  - Each cell of the grid if either completely inside of completely outside the photographed area
- You need to take at most K photo to cover all N points
- Find the minimum area of cell photographed at least once to achieve such goal

Subtasks

For all subtasks, $1 \leq k \leq n$

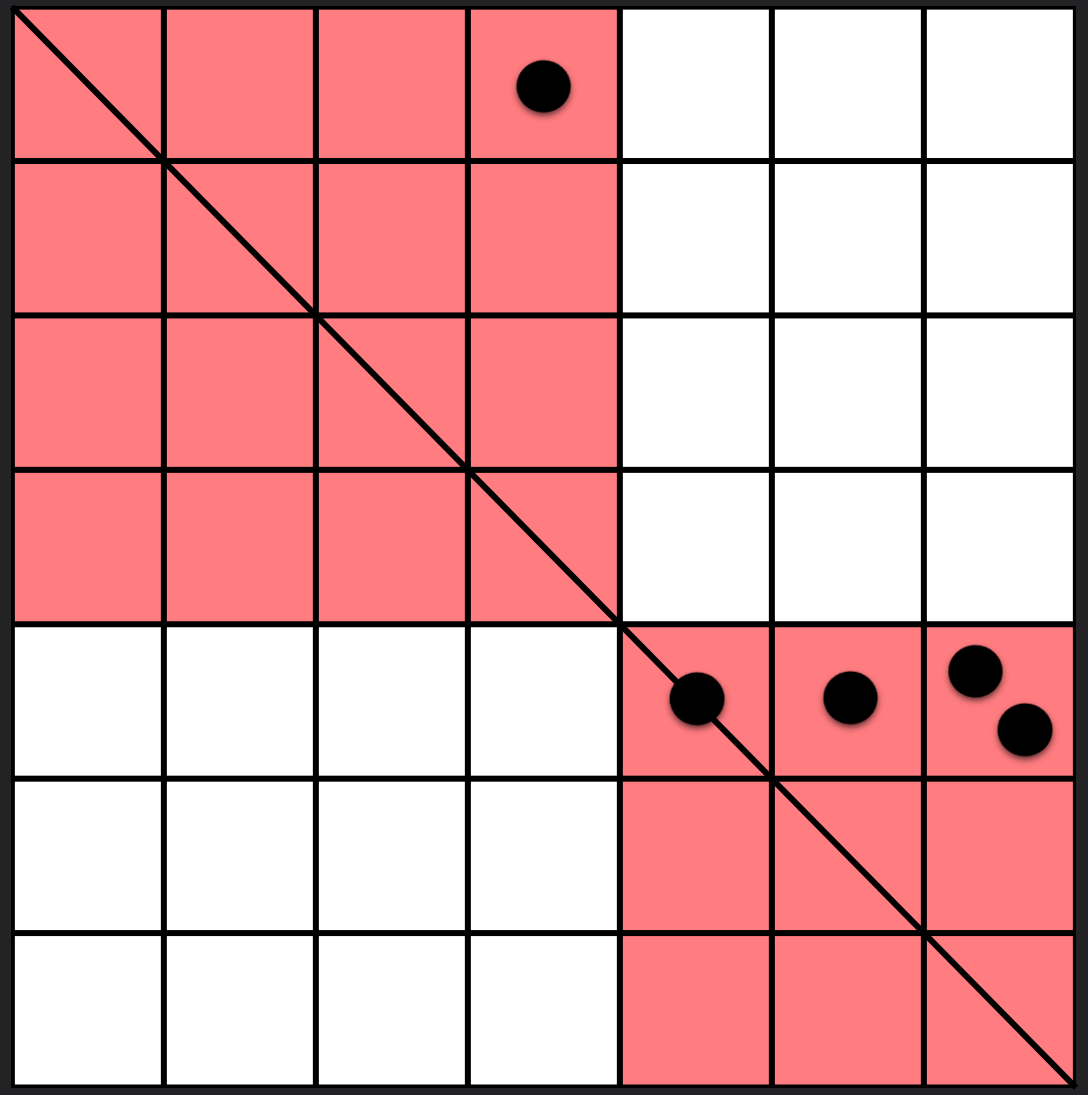
1. 4% $1 \leq n \leq 50, 1 \leq m \leq 100, k = n$
2. 12% $1 \leq n \leq 500, 1 \leq m \leq 1000$

   $for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
3. 9% $1 \leq n \leq 500, 1 \leq m \leq 1000$
4. 16% $1 \leq n \leq 4000, 1 \leq m \leq 10^6$
5. 19% $1 \leq n \leq 50000, 1 \leq k \leq 100, 1 \leq m \leq 10^6$
6. 40% $1 \leq n \leq 10^5, 1 \leq m \leq 10^6$

Sample Input 1

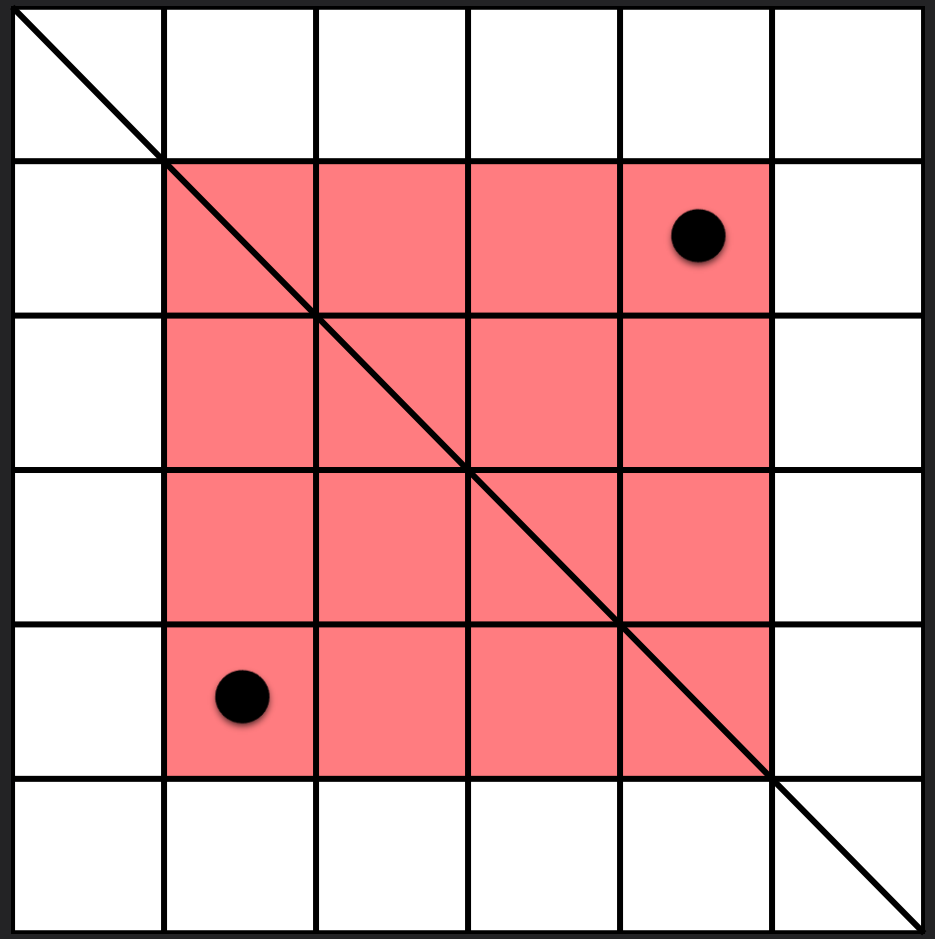
- 5 7 2
- 0 3
- 4 4
- 4 6
- 4 5
- 4 6

- ANS = 16 + 9 = 25



Sample Input 2

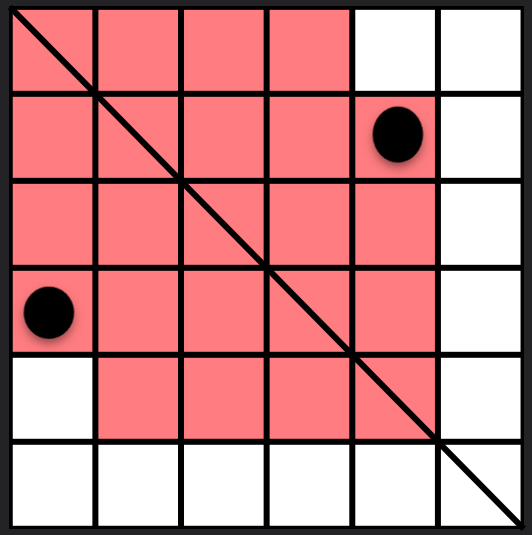
- 2 6 2
- 1 4
- 4 1



- ANS = 16

Subtask 1

- $1 \leq n \leq 50, 1 \leq m \leq 100, k = n$
- As k = n, we can cover each point by 1 photo
- For each point, we use the smallest photo to cover it
  - Square with (r[i], r[i]) and (c[i], c[i]) as corners



Subtask 1

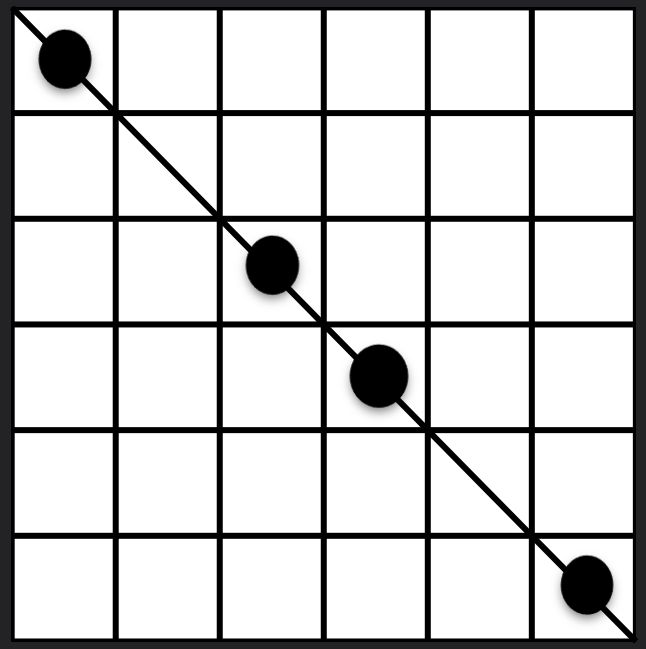
- $1 \leq n \leq 50, 1 \leq m \leq 100, k = n$
- Constraint is small
- We can an 2D array to store the whole grid state
  - Photographed or not
- Ans = number of cells that are photographed
- Time Complexity = $O(NM^2)$
- Space Complexity = $O(M^2 + N)$

Subtask 2

- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- Remove the constraint “k always = n”
- Algorithm in subtask 1 does not work anymore
- Need some new methods to solve the following subtasks

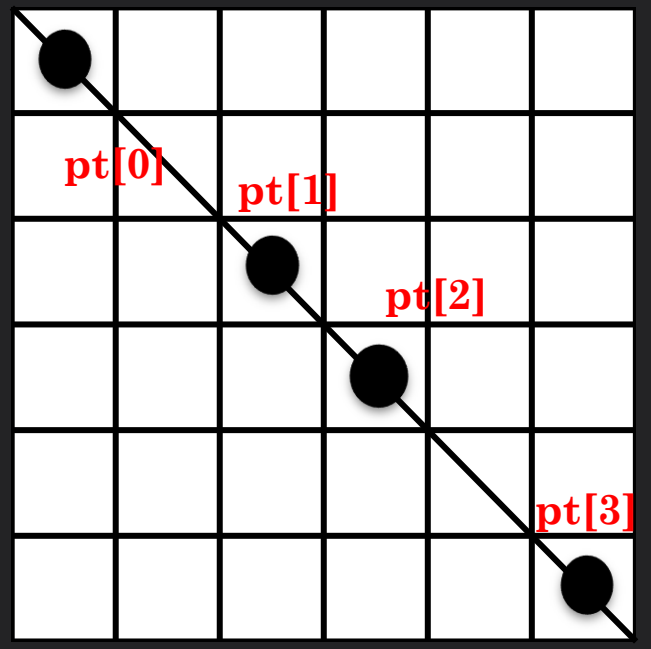
Subtask 2

- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- Notice that every point lies on the main diagonal line of the grid
  - $r_i = c_i$



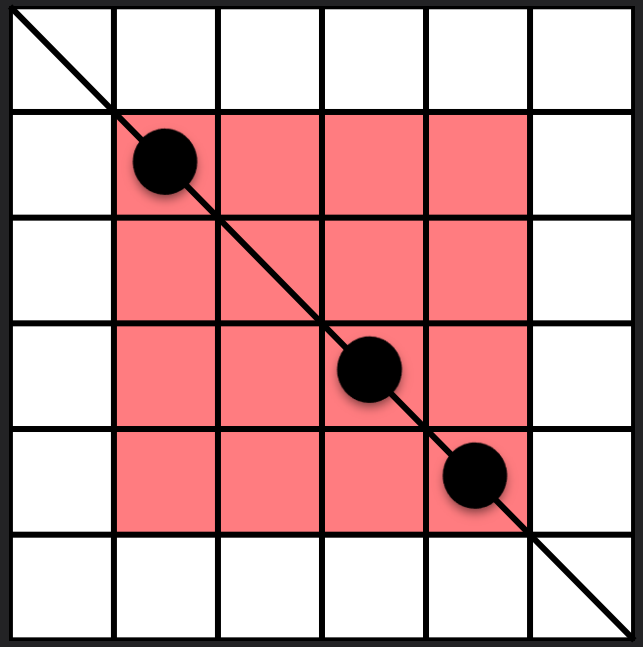
Subtask 2

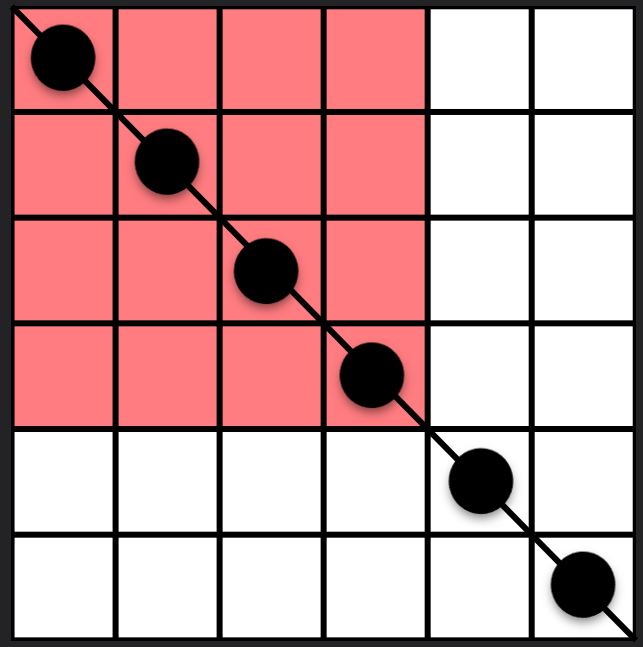
- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- Sort the points by r_i (or c_i)
- If a photo can cover pt[i] and pt[j] ($0 \leq i \leq j \leq n-1$)
- It also cover pt[i+1],p[i+2],…,p[j – 1]



Subtask 2

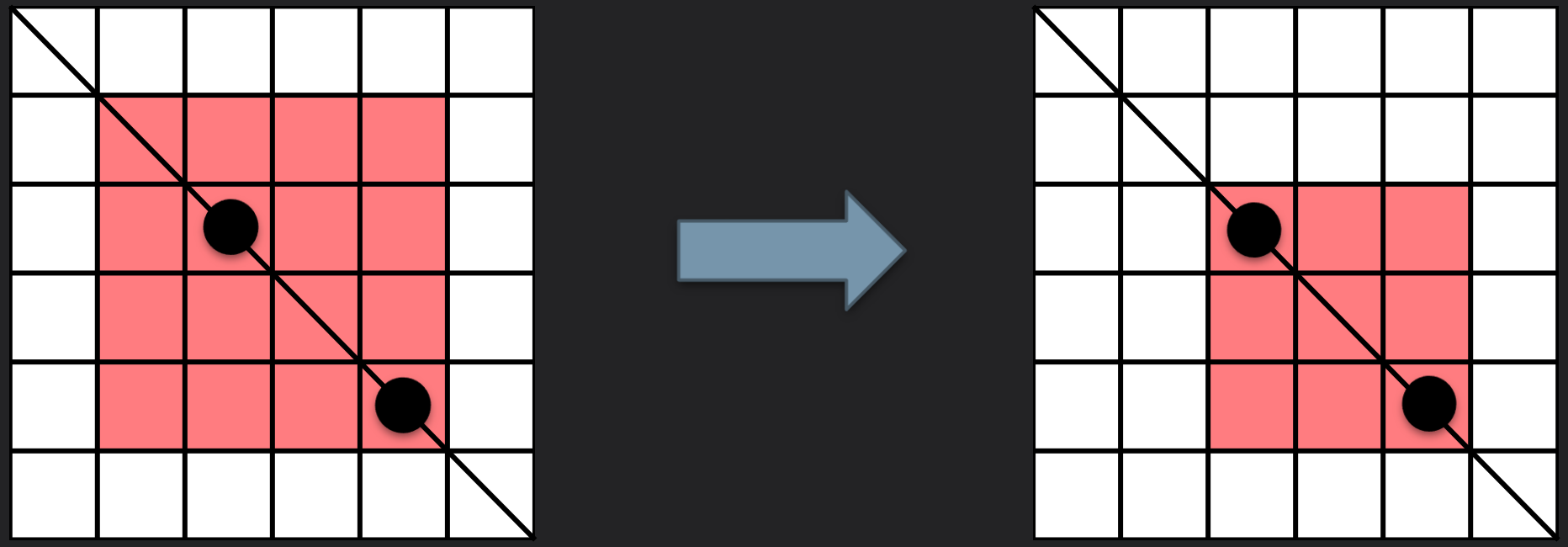
- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$





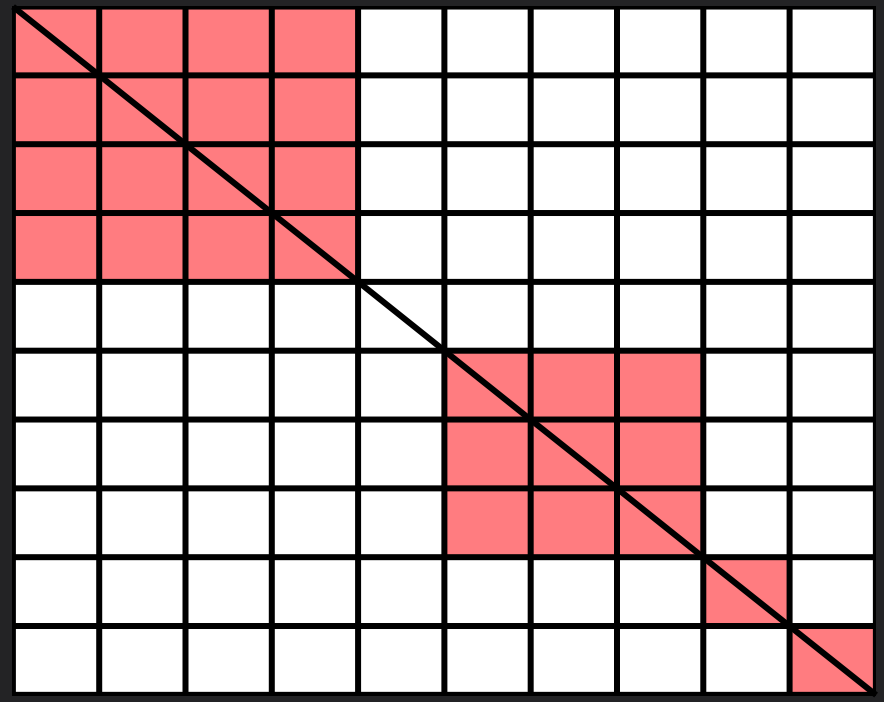
Subtask 2

- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- Also, it is optimal to take photo such that its corners on main diagonal is occupied by points



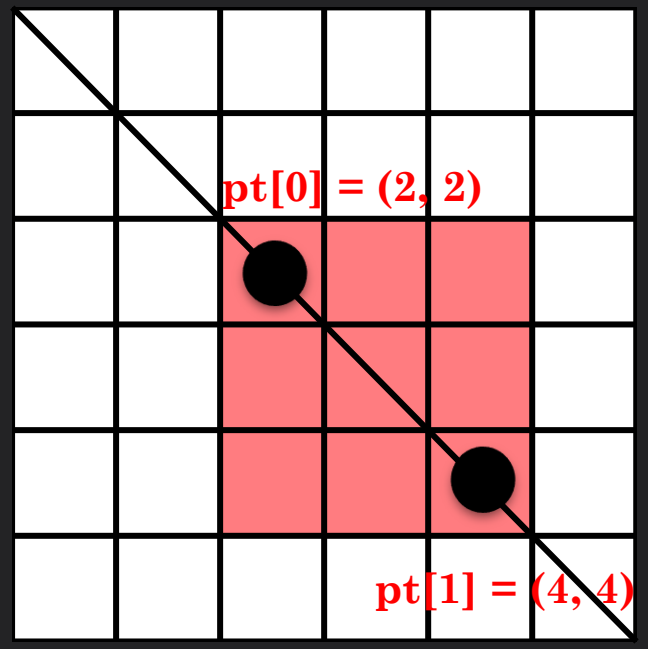
Subtask 2

- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- So the optimal solution will look like something like this
- Photos are non-overlapping
- => Each photo is independent



Subtask 2

- $1 \leq n \leq 500, 1 \leq m \leq 1000,$ $for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- Let cost(i, j) = minimum area of photo covering pt[i..j]
- $cost(i,j) = (r_j - r_i + 1)^2$
- E.g. cost(0, 1) = $(4 - 2 + 1)^2 = 9$



Subtask 2

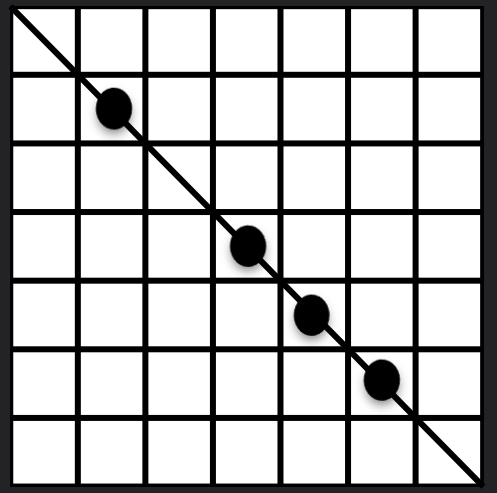
- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- We can come up with a dynamic programming formula
- Let dp[i][j] = minimum number of photographed cells to cover first j^{th} points by using at most i photos
- $$dp[i][j] = \begin{cases} cost(0, j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \leq t < j} dp[i-1][t] + cost(t+1, j)\right), i > 1 \end{cases}$$
- $Ans = dp[k][n-1]$

Subtask 2

- $1 \le n \le 500, 1 \le m \le 1000,\ for\ all\ i\ such\ that\ 0 \le i \le n-1, r_i = c_i$
- $dp[i][j] = \begin{cases} cost(0,j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \le t < j} dp[i-1][t] + cost(t+1,j)\right), i > 1 \end{cases}$

i\j	0	1	2	3
1	INF	INF	INF	INF
2	INF	INF	INF	INF
3	INF	INF	INF	INF

N = 4, K = 3

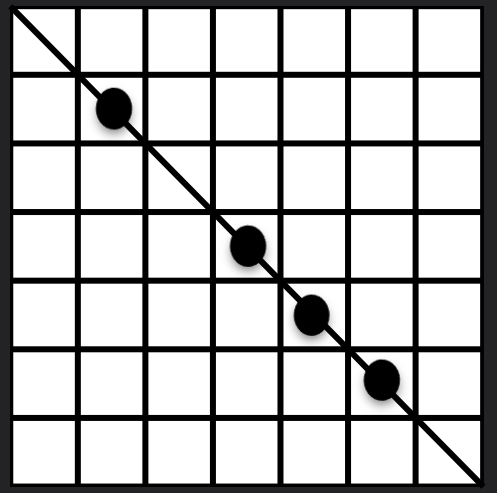


Subtask 2

- $1 \le n \le 500, 1 \le m \le 1000,\ for\ all\ i\ such\ that\ 0 \le i \le n-1, r_i = c_i$
- $dp[i][j] = \begin{cases} cost(0,j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \le t < j} dp[i-1][t] + cost(t+1,j)\right), i > 1 \end{cases}$

i\j	0	1	2	3
1	1	9	16	25
2	INF	INF	INF	INF
3	INF	INF	INF	INF

N = 4, K = 3

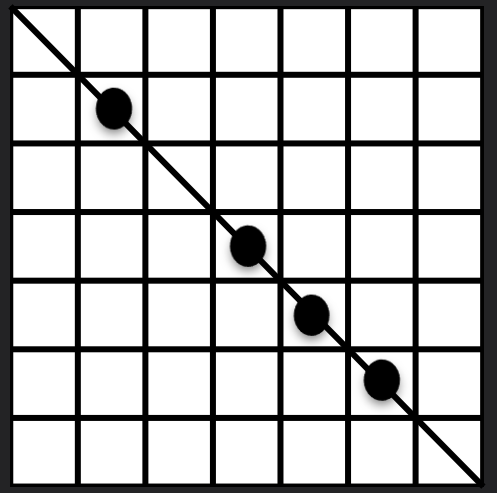


Subtask 2

- $1 \le n \le 500, 1 \le m \le 1000,\ for\ all\ i\ such\ that\ 0 \le i \le n-1, r_i = c_i$
- $$dp[i][j] = \begin{cases} cost(0,j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \le t < j} dp[i-1][t] + cost(t+1,j)\right), i > 1 \end{cases}$$

i\j	0	1	2	3
1	1	9	16	25
2	INF	2	5	10
3	INF	INF	INF	INF

N = 4, K = 3

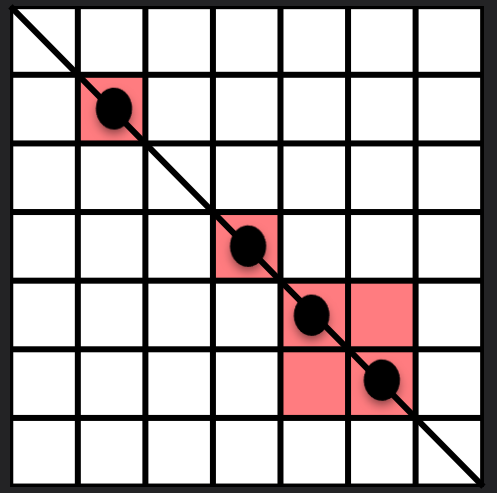


Subtask 2

- $1 \le n \le 500, 1 \le m \le 1000,\ for\ all\ i\ such\ that\ 0 \le i \le n-1, r_i = c_i$
- $$dp[i][j] = \begin{cases} cost(0,j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \le t < j} dp[i-1][t] + cost(t+1,j)\right), i > 1 \end{cases}$$

i\j	0	1	2	3
1	1	9	16	25
2	INF	2	5	10
3	INF	INF	3	6

N = 4, K = 3

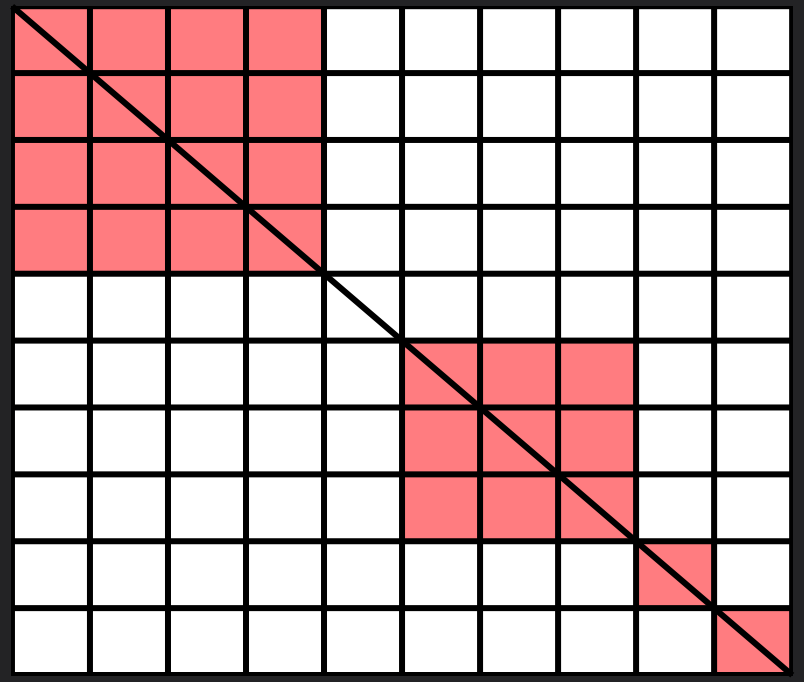


Subtask 2

- $1 \leq n \leq 500, 1 \leq m \leq 1000,\ for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$
- $dp[i][j] = \begin{cases} cost(0,j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \leq t < j} dp[i-1][t] + cost(t+1,j)\right), i > 1 \end{cases}$
- Time Complexity = $O(N^2K)$
- Space Complexity = $O(NK)$

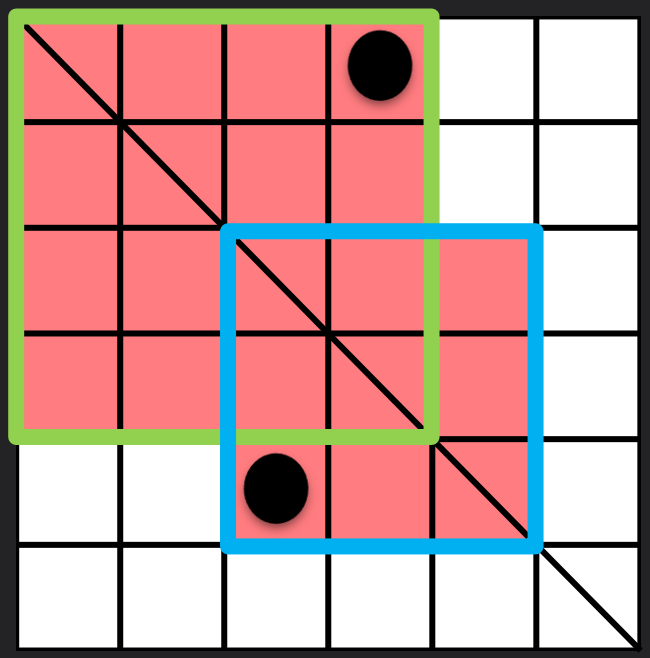
Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- ~~$for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$~~
- Optimal Solution no longer look like this
- Photos may overlap



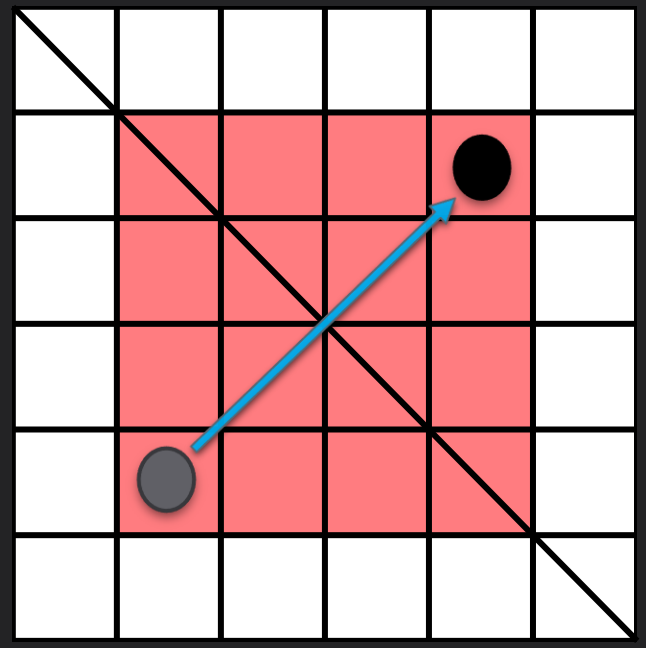
Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- ~~$for\ all\ i\ such\ that\ 0 \leq i \leq n-1, r_i = c_i$~~
- Optimal Solution no longer look like this
- Photos may overlap
- Need modify the algorithms a bit
- DP still work



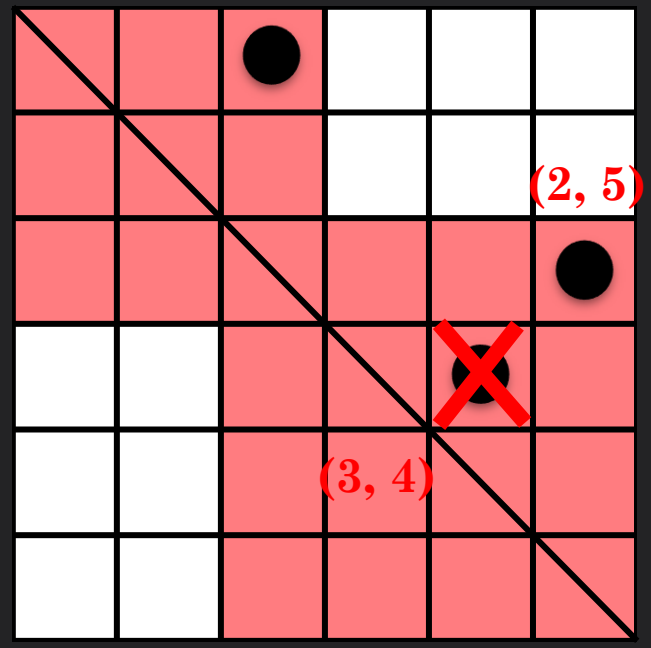
Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- Notice that point (r_i, c_i) is same as (c_i, r_i)
- If a photo cover (r_i, c_i), it must also cover (c_i, r_i)
- So if $r_i \geq c_i$, we can swap two value
  - = reflect along main diagonal



Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- Also, notice that some points are redundant
- If there exist j such that
- $r_i \geq r_j \; and \; c_i \leq c_j$
- (lies on square with corners(r_i, r_i) and (c_i, c_i))
- Point i is redundant

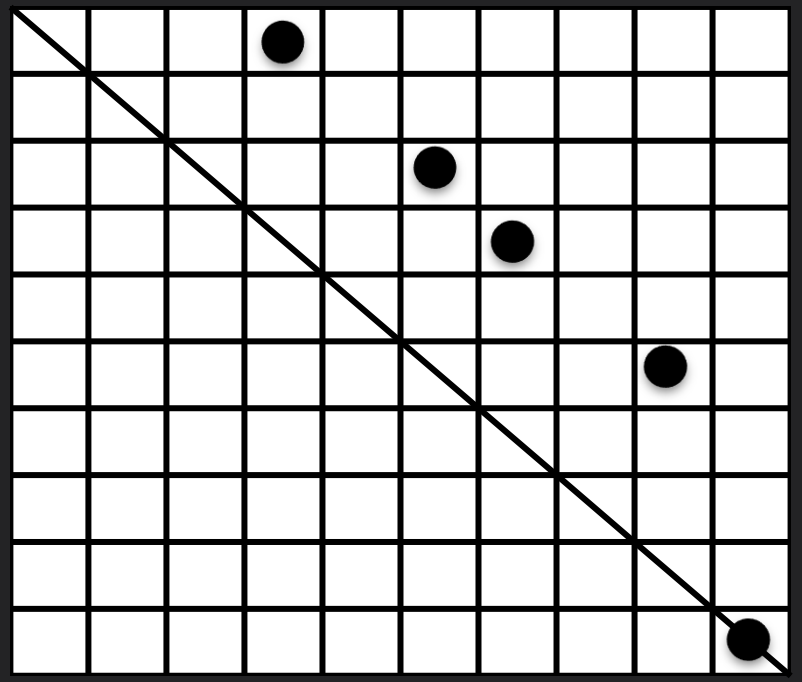


Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- How to remove those redundant points?
- Sort all points by r_i
- For each i, check if point i will make the last non-redundant points redundant
- If so, remove it
- You may use stack to implement it

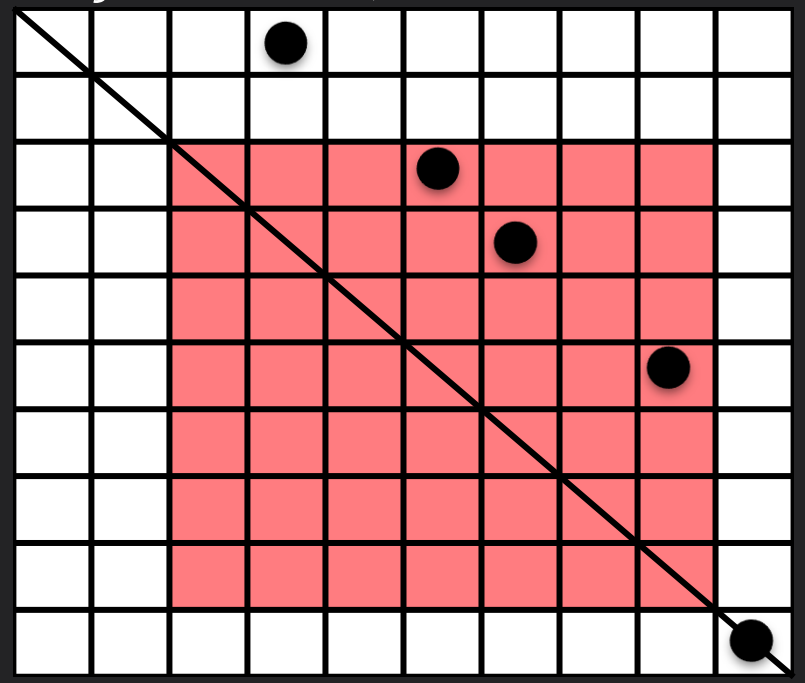
Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- After moving all points above main diagonal and removing all redundant points
- Points will have following property
- $r_i < r_{i+1}, c_i < c_{i+1}, \quad for\ 0 < i < n - 1$
- i.e. $r_i\ and\ c_i$ is strictly increasing



Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- Just like what we have mentioned in subtask 2
- If a photo can cover pt[i] and pt[j] $(0 \leq i \leq j \leq n - 1)$
- It also cover pt[i+1],p[i+2],…,p[j – 1]



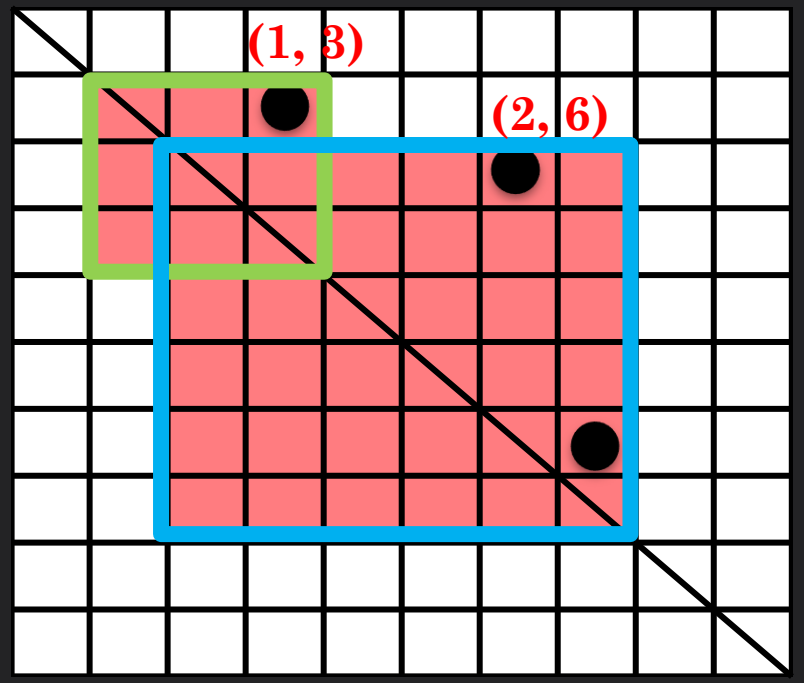
- Area of photo
- $= (c_j - r_i + 1)^2$

Subtask 3

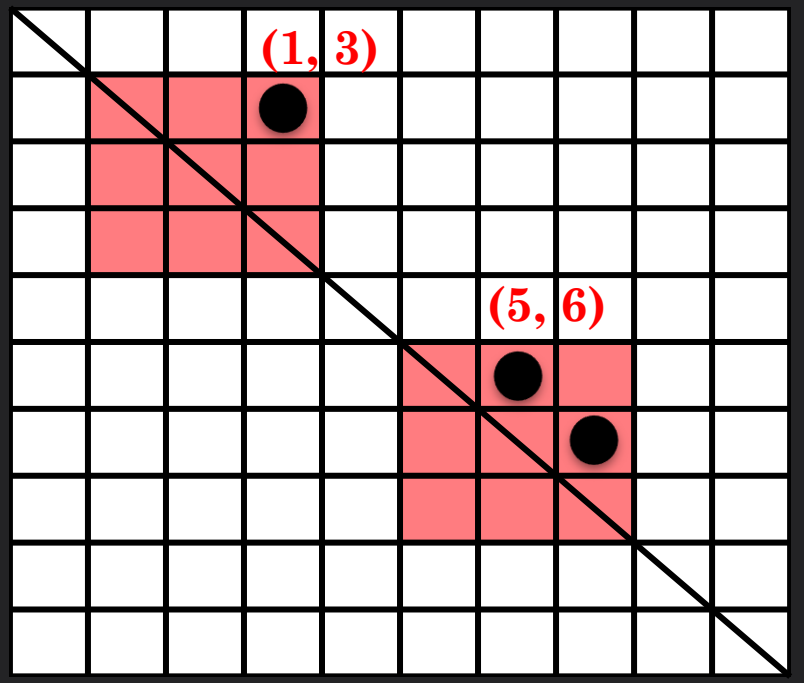
- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- Photos may overlap
- If we already cover point i-1 and want to take a new photo to cover point i to j
- Area of overlap only depends on point i-1 and point i

Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- Area of overlap = $\max(0, c_{i-1} - r_i + 1)^2$



Area of overlap = $max(0, 3 - 2 + 1)^2 = 4$



Area of overlap = $max(0, 3 - 5 + 1)^2 = 0$

Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- $cost(i,j) = area\ of\ photo\ - area\ of\ overlap$
- $$= \begin{cases} (c_j - r_i + 1)^2, i = 0 \\ (c_j - r_i + 1)^2 - \max(0, c_{i-1} - r_i + 1)^2, i > 0 \end{cases}$$

Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- Modify our dp recurrence from subtask 2
- $dp[i][j] = \begin{cases} cost(0, j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \leq t < j} dp[i-1][t] + cost(t+1, j)\right), i > 1 \end{cases}$
- $Ans = dp[k][n-1]$

Subtask 3

- $1 \leq n \leq 500, 1 \leq m \leq 1000$
- $dp[i][j] = \begin{cases} cost(0, j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \leq t < j} dp[i-1][t] + cost(t+1, j)\right), i > 1 \end{cases}$
- Time Complexity = $O(N^2K)$
- Space Complexity = $O(NK)$

Subtask 4, 5

- 4. $1 \leq n \leq 4000, 1 \leq m \leq 10^6$
- 5. $1 \leq n \leq 50000, 1 \leq k \leq 100, 1 \leq m \leq 10^6$

- DP solution from last subtask is not fast enough
- Each transition take O(N) time
- Can apply some DP optimizations

Subtask 4, 5

- Knuth optimization (subtask 4)
- Divide and Conquer optimization (subtask 4, 5)
- Convex Hull Trick (subtask 4, 5)

- Last two have mentioned in DP(III)

Subtask 4, 5

- $dp[i][j] = \begin{cases} cost(0, j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \le t < j} dp[i-1][t] + cost(t+1, j)\right), i > 1 \end{cases}$
- For simplicity, let A[i][j] to be the smallest t that give minimum value to dp[i][j]
- Using the property of A[i][j], we can optimize our dp transition

Subtask 4, 5

- Knuth optimization
- 四邊形不等式優化
- Property : $A[i\ -1][j] \leq A[i][j] \leq A[i][j+1]$

Subtask 4, 5

- Property : $A[i\ -1][j] \leq A[i][j] \leq A[i][j+1]$

- Sufficient condition 1 – cost(i, j) satisfy quadrangle inequality (四邊形不等式)
- If $a \leq b \leq c \leq d$, $\mathrm{f(a,c)} + \mathrm{f(b,d)} \leq f(a,d) + f(b,c)$ $(convex)$
- If $a \leq b \leq c \leq d$, $\mathrm{f(a,c)} + \mathrm{f(b,d)} \geq f(a,d) + f(b,c)$ $(concave)$

Subtask 4, 5

- Property : $A[i-1][j] \leq A[i][j] \leq A[i][j+1]$
- Sufficient condition 2 – cost(i, j) satisfy monotonicity
  - (區間包含單調性)
- If $a \leq b \leq c \leq d$, $f(b,c) \leq f(a,d)$
- We can easily know it is true for function cost(i, j)

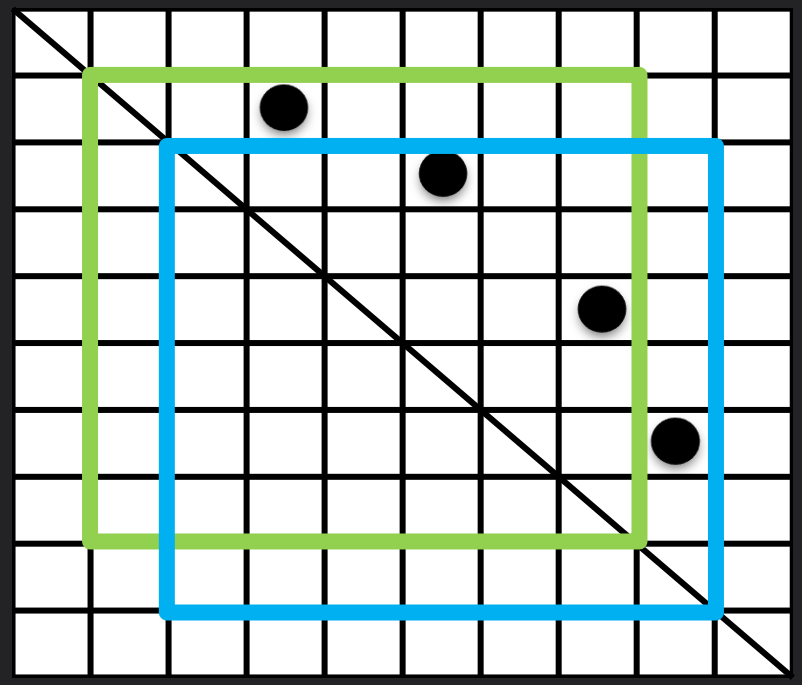
Subtask 4, 5

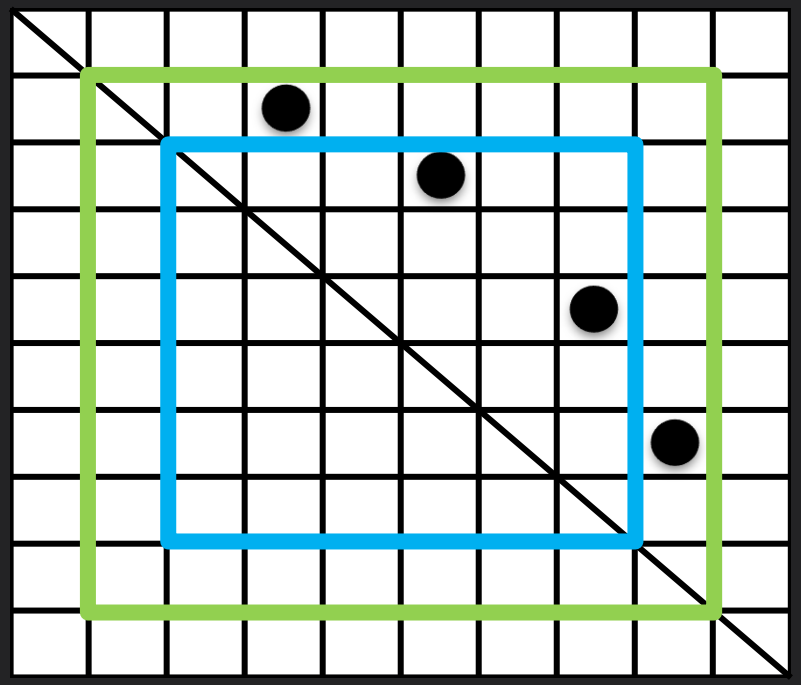
- Sufficient condition 1 – cost(i, j) satisfy quadrangle inequality (四邊形不等式)
- If $a \leq b \leq c \leq d$, $\mathrm{f(a,c)} + \mathrm{f(b,d)} \leq f(a,d) + f(b,c)$ $(convex)$
- We want to prove $cost(a,c) + cost(b,d) \leq cost(a,d) + f(b,c)$

- Area of overlap is same on both side, can ignore

Subtask 4, 5

- Sufficient condition 1 – cost(i, j) satisfy quadrangle inequality (四邊形不等式)
- $cost(a,c) + cost(b,d) \leq cost(a,d) + f(b,c)$





Subtask 4, 5

- Property : $A[i\ -1][j] \leq A[i][j] \leq A[i][j+1]$

- If we know cost(i,j) satisfy those two condition
- $A[i\ -1][j] \leq A[i][j] \leq A[i][j+1]$ holds
  - Hard to prove
  - You may refer to the reference material after

Subtask 4, 5

- Property : $A[i-1][j] \leq A[i][j] \leq A[i][j+1]$
- If we calculate A[i][j] in order of increasing i and decreasing j
- We know the value of A[i – 1][j] and A[i][j + 1] when we want to calculate A[i][j]

- To calculate A[i][j], just iterate from A[i – 1][j] to A[i][j + 1] and find the smallest t that give minimum value to dp[i][j]

Subtask 4, 5

- Pseudo Code :

```
for(int i = 0; i < n; i++){
        dp[1][i] = cost(0, i);
        A[1][i] = 0;
    }

    for(int i = 2; i <= k; i++){
        A[i][n] = n - 1;
        for(int j = n - 1; j >= 0; j--){
            int lb = A[i - 1][j];
            int ub = A[i][j + 1];
            dp[i][j] = dp[i - 1][j];

            for(int t = lb; t <= ub; t++){
                long long val = dp[i - 1][t] + cost(t + 1, j);
                if(val < dp[i][j]){
                    dp[i][j] = val;
                    A[i][j] = t;
                }
            }
        }
    }
```

Subtask 4, 5

- Time Complexity ?
- Consider c = j – i
- For every index i, j, such that j – i = fixed c
- Number of iteration = $\sum A[i][j+1] - A[i-1][j]$ = O(N)

- E.g. Let c = 1, N = 3, K = 3
- Number of iteration = (A[2][2] – A[1][1]) + (A[3][3] – A[2][2])
  (Calculate A[2][1], Calculate A[3][2])
- = A[3][3] – A[1][1] = O(N)

Subtask 4, 5

- Time Complexity ?
- Number of possible value c = O(N)
- For each fixed value c, it takes O(N) iteration to transit

- Therefore,
- Time complexity = $O(N^2)$
- Space Complexity = O(NK)

Subtask 4, 5

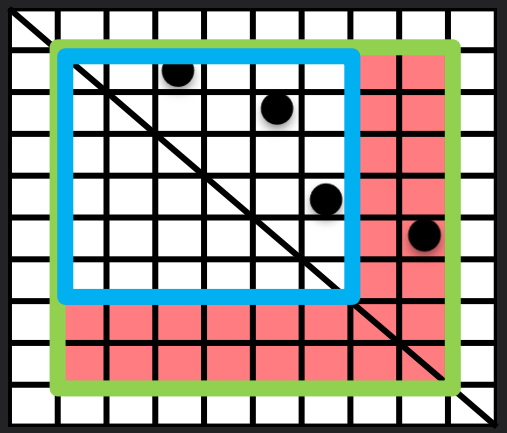
- Divide and Conquer optimization
- Property : $A[i][j] \leq A[i][j+1]$

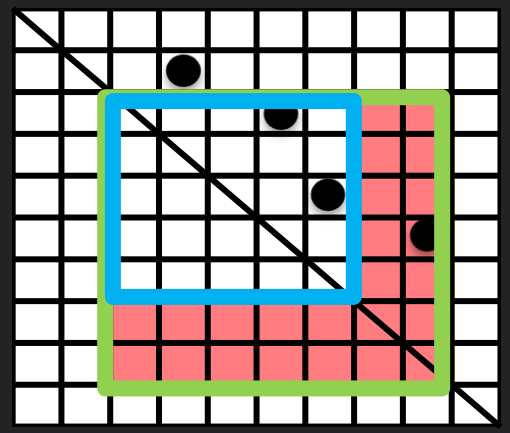
- We can apply D&C Optimization if $A[i][j] \leq A[i][j+1]$ $for\ all\ j$

Subtask 4, 5

- Property : $A[i][j] \leq A[i][j+1]$
- For simplicity, let i1 = A[i][j]

- For $i0 < i1, dp[i-1][i0] + cost(i0+1, j) > dp[i-1][i1] + cost(i1+1, j)$
- $\because cost(i0+1, j+1) - cost(i0+1, j) > cost(i1+1, j+1) - cost(i1+1, j)$





Subtask 4, 5

- For $i0 < i1, dp[i-1][i0] + cost(i0+1, j) > dp[i-1][i1] + cost(i1+1, j)$
- $\because cost(i0+1, j+1) - cost(i0+1, j) > cost(i1+1, j+1) - cost(i1+1, j)$
- $\therefore dp[i-1][i0] + cost(i0+1, j+1) > dp[i-1][i1] + cost(i1+1, j+1)$

- So, $A[i][j+1] \geq i1, i.e. A[i][j] \leq A[i][j+1]$

Subtask 4, 5

- Property : $A[i][j] \leq A[i][j+1]$
- With this property, we can calculate our dp value recursively
- void solve(int i, int L, int R, int optL, int optR);
- Let M = (L+R)/2
- Find dp[i][M] and A[i][M]. Then call solve() for the left and the right parts.

Subtask 4, 5

```
void solve(int i, int L, int R, int optL, int optR){
1. if (L > R) return;
2. int opt = optL; //opt represents C[i][M]
        for(p = optL + 1; p <= optR; p++)
                if(dp[i-1][p] + cost(p+1, M) < dp[i-1][opt] + cost(opt+1, M))
                        opt = p;
3. dp[i][M] = dp[i-1][opt] + f(opt+1, M);
4. solve(i, L, M – 1, optL, opt);
   solve(i, M + 1, R, opt, optR);
}
```

Subtask 4, 5

- Divide and Conquer optimization
- Property : $A[i][j] \leq A[i][j+1]$

- Time Complexity = O(NK logN)
- Space Complexity = O(NK)

Subtask 4, 5

- Convex Hull Trick (subtask 4, 5)
  - 斜率優化
- $dp[i][j] = \begin{cases} cost(0, j), i = 1 \\ \min\left(dp[i-1][j], \min\limits_{0 \le t < j} dp[i-1][t] + cost(t+1, j)\right), i > 1 \end{cases}$
- Consider the transition of dp[i][j] and two indices t and p
  - $(0 \le p < t < j)$
- When will we pick t instead of p ?

Subtask 4, 5

- When will we pick t instead of p ?
- t is better than p, $(0 \leq p < t < j)$
- => $dp[i-1][t] + cost(t+1, j) < dp[i-1][p] + cost(p+1, j)$
- $cost(i,j) = \left(c_j - r_i + 1\right)^2 - \max(0, c_{i-1} - r_i + 1)^2 = c_j^2 - 2c_j(r_i - 1) + (r_i - 1)^2 - \max(0, c_{i-1} - r_i + 1)^2$
- => $(dp[i-1][t] + (r_{t+1}^2 - 1)^2 - \max(0, c_t - r_{t+1} + 1)^2) - 2c_j(r_{t+1} - 1) + c_j^2 < \left(dp[i-1][p] + \left(r_{p+1}^2 - 1\right)^2 - \max\left(0, c_p - r_{p+1} + 1\right)^2\right) - 2c_j(r_{p+1} - 1) + c_j^2$
- Let $f(x) = dp[i-1][x] + (r_{x+1}^2 - 1)^2 - \max(0, c_x - r_{x+1} + 1)^2$
- => $f(t) - f(p) < 2c_j\left(r_{t+1} - r_{p+1}\right)$
- => $\frac{f(t) - f(p)}{r_{t+1} - r_{p+1}} < 2c_j$

Subtask 4, 5

- When will we pick t instead of p ?
- t is better than p, $(0 \leq p < t < j)$
- $\frac{f(t)-f(p)}{r_{t+1}-r_{p+1}} < 2c_j$
- Looks like slope form
- Let m(p, t) be $\frac{f(t)-f(p)}{r_{t+1}-r_{p+1}}, (p < t)$

Subtask 4, 5

- Recall HKOI training DP(III)
- Property 1: If p < t < l and m(p, t) > m(t, l), then there is no need to consider t.
- Property 2: If m(t, p) < $2c_j$, there is no need to consider j in subsequent steps (steps j+1, ..., N).

- Using those property, you can find A[i][j] quickly by using deque

Subtask 4, 5

1. while (q's size >= 2) and (m(q[L], q[L+1]) < $2c_j$)
   L++;
2. if (q not empty) dp[i][j] = dp[i-1][q[L]] + cost(q[L]+1,j);
3. while (q's size >= 2) and (m(q[R-2], q[R-1]) > m(q[R-1], j))
   R--;
4. q[R++] = i;

- You may change division to multiplication when comparing slopes
  - Avoid precision error

Subtask 4, 5

- Convex Hull Trick (subtask 4, 5)

- Time Complexity = O(NK)
- Space Complexity = O(NK)

- Can be used when the recurrence looks like this
- $dp[i] = \min_{j<i} dp[j] + b[j] * a[i]$

Subtask 6

- Since number of DP state = NK
- Seems like it is impossible to improve more
  - Faster than O(NK)

- Because of the limitation of k
- We need one more dimension for our dp

Subtask 6

- Let’s think about a similar problem

- Remove the restriction on number of photo
- Each photo cost some constant penalty C
- Find the minimum cost of covering all N points

Subtask 6

- We can easily come up with a dp recurrence
- $dp[i] = \min_{j<i} dp[j] + cost(j+1, i) + C$

- Looks similar to our original problem's dp recurrence!

- We can apply Convex Hull Tricks to compute it in O(N)
- Also, we can compute the minimum number of photo used to get optimal solution too

Subtask 6

- How is two problems related?

- If the constant C is very small,
  - The optimal solution will use N photos
- If the constant C is very big,
  - The optimal solution will use 1 photo

Subtask 6

- How is two problems related?
- If we can find a constant C such that the optimal solution use exactly K photos
- Ans of original problem = dp[n – 1] – K * C

Subtask 6

- So, how do we can find out such C ?
- When do such C exists?

- Theorem. $cost(\mathrm{i}-1,\mathrm{j})-\mathrm{cost}(\mathrm{i},\mathrm{j}) \geq cost(i,j)-cost(i+1,j)$
  - Concave Monge Property
  - Hard to prove

Subtask 6

- Theorem. $cost(\mathrm{i}-1,\mathrm{j})-\mathrm{cost}(\mathrm{i},\mathrm{j}) \geq cost(i,j)-cost(i+1,j)$
- Let g(i) = dp[i][n-1] = minimum cost to cover all points by i photos

- If Concave Monge Property holds, $g(i-1)-g(i) \geq g(i)-g(i+1)$

- Difference between g(i) and g(i+1) is decreasing

Subtask 6

- When $C \geq g(i) - g(i+1)$, which means taking i photos is better than taking i+1 photos in the optimal solution with penalty C

- Also, as $g(i-1) - g(i) \geq g(i) - g(i+1)$ holds
- When $C \geq g(i) - g(i+1)$, taking i photos is better than taking i+1,i+2,i+3,…. Photos

- $C \geq g(i) - g(i+1) \geq g(i+1) - g(i+2) \geq g(i+2) - g(i+3) \geq \cdots$

Subtask 6

- So we can binary search the constant value C
  - i.e. binary search the difference between g(k) and g(k+1)
- Our target => value C such that the number of photo used to achieve minimum cost = K
- Then we can use that value C to compute the dp again
- Ans = dp[n-1] – K * C (remove the penalty)

Subtask 6

```
pair<long long, int> solve_dp(long long C){
        ..............................................
        return {minimum cost, minimum number of photo used to achieve minimum cost};
}
while(ub – lb > 1){
        long long mid = (lb+ub)/2;
        if(solve_dp(mid).second <= k) ub = mid; //penalty too big
        else lb = mid; //penalty too small
}
```

- Ans = solve_dp(ub).first – K * ub

Subtask 6

- Ans = solve_dp(ub).first – K * ub

- Sometime solve_dp(ub).second may not = K
- But it is okay

- We can always use K photos to achieve same cost

Subtask 6

- “Alien Trick” / wqs 二分

- Upper bound of C = M
- Time Complexity = O(NlogN + NlogM)
- Space Complexity = O(N)

- Full solution!

Conclusion

- DP optimization is useful and often appear on OI contests
- Proof for applicability of DP optimization is hard

- You should not write a formal proof during contest time
- You may test those property on small test case and/or prove it intuitively

Practice Tasks

- Please refer to DP(III) for CHT and D&C optimization tasks

- Knuth Optimization
- https://tioj.ck.tp.edu.tw/problems/1449
- http://acm.hdu.edu.cn/showproblem.php?pid=2829
- http://acm.hdu.edu.cn/showproblem.php?pid=3506

Practice Tasks

- “Alien Trick”
- https://codeforces.com/contest/674/problem/C
- https://codeforces.com/contest/739/problem/E
- https://www.lydsy.com/JudgeOnline/problem.php?id=2654
- https://tioj.ck.tp.edu.tw/problems/1986

Reference

- Task from IOI 2016
- https://codeforces.com/blog/entry/8219
  - A summary of different types of DP Optimization
- http://ioinformatics.org/locations/ioi16/contest/IOI2016_analysis.pdf
  - IOI 2016 alien official solution
- http://www.tsinsen.com/resources/Train2012-sol-wqs.pdf
  - IOI2012 中國國家集訓隊 王欽石 淺析一類二分方法

Reference

- https://blog.csdn.net/NOIAu/article/details/72514812
- http://chino.taipei/code-2016-0402Algorithm-DP優化之四邊形不等式優化/
  - Materials about Knuth optimization
- http://assets.hkoi.org/training2018/dp-iii.pdf
  - HKOI training DP(III) ppt
- https://link.springer.com/content/pdf/10.1007%2FBF02574380.pdf
- http://www.cs.ust.hk/mjg_lib/bibs/DPSu/DPSu.Files/sdarticle_204.pdf
  - Materials about Concave Monge Property

Appendix – Proof for Concave Monge Property

Thanks **Alex Tung** for helping and writing the proof

You may contact Alex Tung if you have any problems

Meaning of symbols is different from lecture point, meaning of symbols in there follow IOI official solution

https://link.springer.com/content/pdf/10.1007%2FBF02574380.pdf (P.267-271)

Use the result of this paper to apply on "Aliens"

Recall the dp formulation:

$$\mathrm{dp[i][k]} := \min_{0 \le j < i} (dp[j][k-1] + (r_i - l_{j+1})^2 - \max(0, r_j - l_{j+1})^2$$

$$\begin{pmatrix} l_1 < l_2 < \ldots < l_n, \ l_i < r_i \\ r_1 < r_2 < \cdots < r_n \end{pmatrix}$$

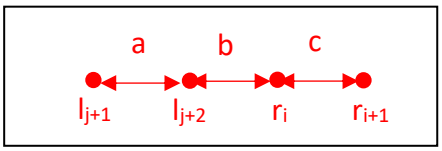
So we build the complete DAG with nodes $V_0, \ldots, Vn$ and

(for $\mathrm{j} \le \mathrm{i}$)

$$\mathrm{w(j,i)} = (\mathrm{r_i} - l_{j+1})^2 - \max(0, r_j - l_{j+1})^2$$

$$= (\mathrm{r_i} - l_{j+1})^2 - h(j)$$

Depends on j only



Then the Concave Monge is immediate:

- for $\mathrm{j} + 1 < \mathrm{i}$,

  LHS = $\mathrm{w(j,i)} + \mathrm{w(j+1,i+1)}$

  $= (\mathrm{r_i} - l_{j+1})^2 - h(j) + (r_{i+1} - l_{j+2})^2 - h(j+1)$

  $\le (\mathrm{r_i} - \mathrm{l_{j+2}})^2 - h(j+1) + (r_{i+1} - l_{j+1})^2 - h(j)$

  $= \mathrm{w(j+1,i)} + \mathrm{w(j,i+1)}$

Because $(a+b)^2 + (b+c)^2 \le (a+b+c)^2 + b^2$ $for + ve\ a, b, c$

- for $\mathrm{j} + 1 = \mathrm{i}$ it is similar. Pictorially:

LHS: RHS: