2019牛客暑期多校训练营 第二场

eddy1021

牛客竞赛
AC.NOWCODER.COM

A Eddy Walk

- Math, Ad hoc

A Eddy Walk

- **Math, Ad hoc, brute force**

  Try some combinations of small N, M.
  Found that probability = 1/(N - 1) for all m except 0

A Eddy Walk

- **Math, Ad hoc, brute force**

Try some combinations of small N, M.
Found that probability = 1/(N - 1) for all m except 0

Corner case: (N=1, M=0) = 1

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

prob[i] = (prob[i-1] + prob[i-2] + ... + prob[i-k]) / k
prob[0] = 1
prob[i < 0] = 0

prob[x -> \infinity] = 2 / (k + 1)

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

  prob[i] = (prob[i-1] + prob[i-2] + ... + prob[i-k]) / k

  Results in an O(NK) solutions. Clearly TLE

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

  Solving Linear Recurrence faster? Matrix Multiplication!

  Results in an O(K^3 \lg N). Still TLE...

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

  Solving Linear Recurrence faster? Matrix Multiplication!

  Actually, we can solve LR in **O(K^2 \lg N)**!

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

Solving LR in **O(K^2 \lg N)**!

Core Idea: How to represent dp[n] from dp[0], dp[1], ..., dp[k]
If we can do that, we can compute dp[n] in O(K)

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

Solving LR in **O(K^2 \lg N)**!

Core Idea: How to represent dp[n] from dp[0], dp[1], ..., dp[k]

We can represent dp[n] from dp[n-1], dp[n-2], ... dp[n-k] so as dp[n-1] from dp[n-2], dp[n-3], ..., dp[n-k-1]

Then, we can represent dp[n] from dp[n-2], dp[n-3], ..., dp[n-k-1]

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

Solving LR in **O(K^2 \lg N)**!
Core Idea: How to represent dp[n] from dp[0], dp[1], ..., dp[k]

dp[n] = W_1 dp[n-1] + W_2 dp[n-2] + ... + W_k dp[n-k]
dp[n-1] = W_1 dp[n-2] + W_2 dp[n-3] + ... + W_k dp[n-k-1]
->
dp[n] = V_1 dp[n-2] + V_2 dp[n-3] + ... + V_k dp[n-k-1]
If we do this till first k terms, it’ s in O(NK)! We need to be faster!!

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

Solving LR in **O(K^2 \lg N)**!
Core Idea: How to represent dp[n] from dp[0], dp[1], …, dp[k]
We need to be faster!! -> Do multiple steps at once!(Expand them all)

$$
\begin{aligned}
F_n &= \sum_{i=0}^{k-1} w_{p,i} F_{n-i-(pk-k+1)} \\
&= \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} w_{p,i} w_{q,j} F_{n-i-j-(pk-k+1+qk-k+1)} \\
&= \sum_{i=0}^{2k-2} v_i F_{n-i-((p+q)k-k+1)+k-1}
\end{aligned}
$$

B Eddy Walker 2

- **Math, Linear Recurrence, Dynamic Programming**

Solving LR in **O(K^2 \lg N)**!
Core Idea: How to represent dp[n] from dp[0], dp[1], ..., dp[k]
We can expand the parameters twice in O(K^2) now.

Then, we can compute the required parameters in O(K^2 \lg N)

Bonus: Exists an **O(K \lg K \lg N)** algorithm to solve LR

C Go on Strike!

- **Data structue and data structures(Segment Tree, Link-Cut Tree)**

For those queries, we can construct a timeline segment tree to maintain the lifetime of each edge. Then, add an edge when going down the node of segment tree and remove(undo the add) when going back of it.

C Go on Strike!

- **Data structue and data structures(Segment Tree, Link-Cut Tree)**

If we can maintain minimum spanning tree(MST) and its diameter in O(f).
We can solve it in O(Q f \lg Q) now!.

Maintaining a dynamic MST is a classic problem which can be solved in O(\lg N) with only addition of edge.(We can undo it with the power of timeline segment tree)
One way to achieve it is by link-cut tree in O(\lg N) per addition.

C Go on Strike!

- **Data structue and data structures(Segment Tree, Link-Cut Tree)**

Maintaining a dynamic MST is a classic problem which can be solved in O(\lg N) with only addition of edge.(We can undo it with the power of timeline segment tree)

While maintaining MST, we can use another link-cut tree to maintain diameter.
To compute diameter, it’ s another classic dynamic programming problem on tree.
With the power of link-cut tree, we can solve it dynamically in O(\lg N).

C Go on Strike!

- **Data structue and data structures(Segment Tree, Link-Cut Tree)**

Since each modification will take $O(\lg N)$.

Overall time complexity will be $O(Q \lg N \lg Q)$

D Kth minimum Clique

- **Brute force, bitmask**

We just need to brute force!
By maintaining the current clique, potential new vertex to add, and current weight sum, we can find out next possible clique in O(1)!.

D Kth minimum Clique

- **Brute force, bitmask**

By maintaining the current clique, potential new vertex to add, and current weight sum, we can find out next possible clique in O(1)!.

By extracting out the lowest bit of potential set(maintaining in bitmask), we can either construct another valid clique(Yeah!) or found that sum of weights exceeds V. Thus, for each clique, we will either find out another clique or stop searching in O(1). We only need to find out at most K cliques for each V. Results in an O(K \lg MAXV) solution.

D Kth minimum Clique

- **Brute force, bitmask**

First, let sort all the vertice by their weight.

Then, binary search the answer V.

We need to find out whether there are K or more or less cliques with value <= V.

We just need to brute force!

E Maze

- **Data Structure(Segment tree), Dynamic Programming**

  We can construct a DP as:

  dp[i][j] = sum(dp[i-1][k] for (k < j and b_ik=b_i{k+1}=...=b_ij=0))+
  sum(dp[i-1][k] for (k > j and b_ik=b_i{k-1}=...=b_ij=0))

  It can be represented as matrix multiplication from dp[i][*] to dp[i+1][*].

E Maze

- **Data Structure(Segment tree), Dynamic Programming**

$$dp[i][j] = \text{sum}(dp[i-1][k] \text{ for } (k < j \text{ and } b_{ik}=b_{i\{k+1\}}=\ldots=b_{ij}=0))+$$
$$\text{sum}(dp[i-1][k] \text{ for } (k > j \text{ and } b_{ik}=b_{i\{k-1\}}=\ldots=b_{ij}=0))$$

Construct a segment tree, each node consists of the weight of each state. The final answer will be the product of them.

Then, modify time will be $O(m^3 \lg N)$, query time will be $O(1)$.

F Partition problem

- **Brute force**

There are C(2N, N) ways of assignments.
For each assignment, the cost can be calculated in O(N^2).

Results in O(C(2N, N) N^2) time complexity.

F Partition problem

- **Brute force**

O(C(2N, N) N^2) may **NOT** be fast enough.

You can solve it in **O(C(2N, N) N)**.

First, put all people in one team. The value will be 0.
Whenever pull one people to another team, we can compute the delta of the value.
In each step, it takes O(N).

G Polygons

- **Geometry**

By Euler characteristic, there will be at most O(N^2) planes since there are at most O(N^2) intersections.

We can find all of them!

G Polygons

- **Geometry**

Compute each intersections and put intersections on each line.

For each segment, create two directed segment in each direction.

From an start intersection, go through any unused directed segment. When you reach next intersection, find another unused directed segment which is has largest(or smallest) angle. Keep going until reach the starting point. You will find out an closed polygon.

G Polygons

- **Geometry**

To avoid unbounded polygon, you can create an large rectangle enclosing all the intersection. But, the size of this rectangle would by very large(not just order of 1000).

Or, you can carefully deal with it.

H Second Large Rectangle

- **Dynamic Programming**

The largest rectangle is a classic problem.

We can solve it in O(NM) by DP.
For each (i, j), we can find out the largest rectangle whose bottom contains (i, j).

Suppose it’ s X by Y.

Then, we can maintain a sorted list(or heap) containing first several largest rectangle. Actaully, we only need to keep 2!.

H Second Large Rectangle

- **Dynamic Programming**

The largest rectangle is a classic problem.

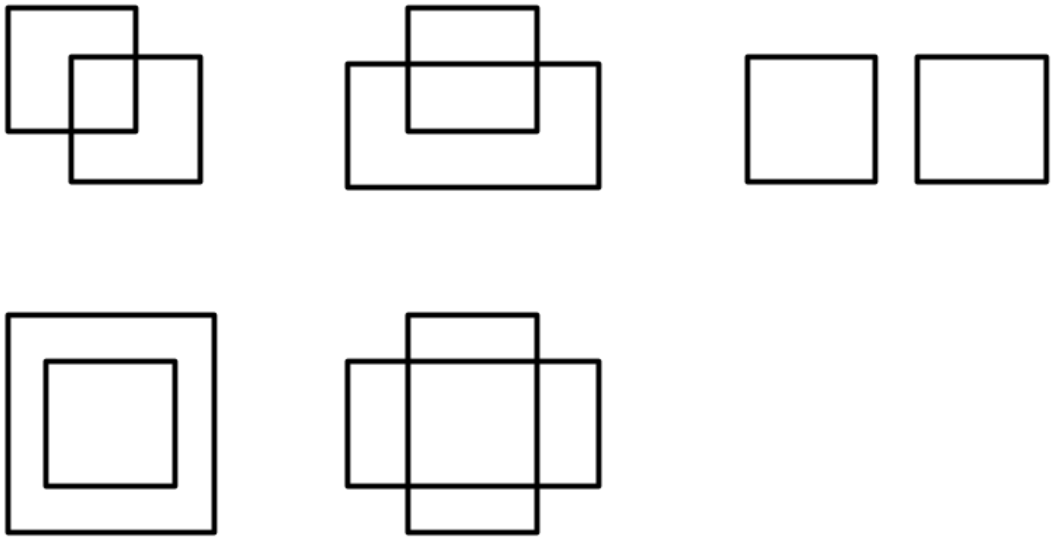
We can solve it in O(NM) by DP.

For a rectangle with size X by Y. We only need to push XY, (X-1)Y, X(Y-1) into the sorted list. And, always resize it into 2 or less items to keep the insertion running in O(1).

Resulting an solution in O(NM).

I Inside A rectangle

- **Dynamic Programming, Case analysis**



I Inside A rectangle

- **Dynamic Programming, Case analysis**

  All of them can be computed in $O(N^4)$ dynamic Programming.

  Two separated rectangles would be an easy one.
  For others, the idea is to iterate through the overlapped rectangles.
  In pre-dp out some useful values, such as which rectangle has largest value with a fixed point (x_2, y_2) and its (x_1, y_1) satisfy $x_1 <= X'$, $y_1 <= Y'$.

J Subarray

- **Data structure, brute force**

Since there are at most 10^7 ones, possible index included in any positive sum segment will be O(10^7).

We can first find out all these indexes. Then, for each possible indexes range we can compute the result almost brute forcely.

Thanks

牛客竞赛
AC.NOWCODER.COM