M1731 Mock Quizzes

Problem by Charlie Li

Problem statement

- Given N pair of numbers, (n_i, k_i)
- Find $\sum_{r=1}^{k_i} \binom{n_i}{r}$ for every pair of (n_i, k_i)

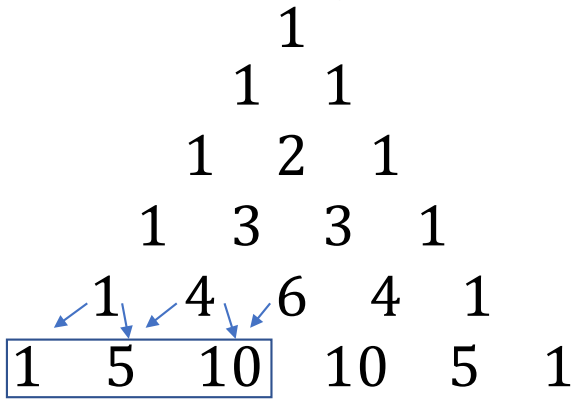
Constraints

- $1 \leq N \leq 100000$
- $0 \leq k_i \leq n_i \leq 100000$

Observation

- Let $f(n,k) = \sum_{r=1}^{k} \binom{n}{r}$
- It is obvious that $f(n,k+1) = f(n,k) + \binom{n}{k+1}$ for $0 \leq k < n$
- Similarly $f(n,k-1) = f(n,k) - \binom{n}{k}$ for $0 < k \leq n$

Observation

- Consider the pascal triangle
- 

- We have $f(n+1,k) = 2 \times f(n,k) - \binom{n}{k}$

Observation

- By reordering the terms, we also get

$$f(n-1,k) = \frac{f(n,k) + \binom{n-1}{k}}{2}$$

$$f(n+1,k) = 2 \times f(n,k) - \binom{n}{k}$$

$$f(n,k+1) = f(n,k) + \binom{n}{k+1}$$

$$f(n,k-1) = f(n,k) - \binom{n}{k}$$

Solution

- If we precompute all n! and $(n!)^{-1}$ mod p, we can calculate $\binom{n}{r}$ mod p in O(1).
- So, we can do every step in O(1).

- Then, we can use the Mo's algorithm, to compute all queries in O($n\sqrt{n}$)

Mo's Algorithm

- Sort all queries according to $\lfloor \sqrt{k} \rfloor$ we call all k with same value of $\lfloor \sqrt{k} \rfloor$ belongs to a same block.
- The sort all the queries in the same block according to n.
- Then answer the queries in that order with each steps change either n or k by 1.
- This will be O($(m+n)\sqrt{n}$)
  - In the same block, answering the next query is O($\sqrt{n}$), overall is O($m\sqrt{n}$)
  - If we change the block, answering the next query is O(n), however, we have at most $\sqrt{n}$ boundaries, so overall O($n\sqrt{n}$)

Solution

- Precompute: O($n \log n$) or O(n)
- Sorting queries: O($m \log m$)
- Answering queries: O($(m + n)\sqrt{n}$)

- Overall: O($m \log m + (m + n)\sqrt{n}$)

M1732 Mary goes around Peter

Solution 1

- Peter turns clockwise if and only if Mary turns clockwise
- Just check if Mary turns clockwise in $O(N)$

Solution 2

- Shoelace formula
  - given coordinates of a polygon, calculate area
- Compute the area of Mary's cycle
  - Positive $\rightarrow$ anticlockwise
  - Negative $\rightarrow$ clockwise
- Works on all kinds of polygons

M1733 Building on fire

General idea

Spiderman should climb as high as he can, as long as he can save all people

Solution

- Suppose Spiderman is at the $y_s{}^{th}$ floor at time s
- Person i jumps out from the $x_i{}^{th}$ floor at time t_i
- Let $f(s) = \min\{x_i + t_i - s \text{ for all } i \text{ where } t_i \geq s\}$
- Spiderman can save all people as long as $y_s \leq f(s)$
- Calculate $f(s)$ in $O(N)$ by the recursive formula:

$$f(s) = \begin{cases} \min(f(s+1)+1, x_i \text{ where } t_i = s) & \text{if exists } t_i = s \\ f(s+1)+1 & \text{otherwise} \end{cases}$$

- Optimal strategy is thus $y_{s+1} = \min(y_s + 1, f(s+1))$
- Simulate and output answer

M1734 Editorial

Alex Tung

Description

- Given number of Tui (頹), Easy, Medium, Hard problems ($N_T + N_E + N_M + N_H = N$), find number of different problem-sets with M problems, s.t. there is at least one problem of each difficulty level

$1 <= N <= 50$, $1 <= M <= 13$

Solution

- Exhaust C_T, C_E, C_M, C_H, the number of Tui, Easy, Medium, and Hard problems you select (e.g. using 4-layer loops)
- Make sure $1 <= C_T <= N_T$, $1 <= C_E <= N_E$, etc. and

  $C_T + C_E + C_M + C_H = M$
- In this case, number of problem-sets

$$= \binom{N_T}{C_T} \times \binom{N_E}{C_E} \times \binom{N_M}{C_M} \times \binom{N_H}{C_H}$$

- Sum up to get the answer :)

M1735 – Fibonacci Word II

- Solution 1:
- Compute string t = F(x) until the length of t is at least f + n – 1 where f is the smallest Fibonacci number >= n
- s is a substring of F(inf) if and only if it is a substring of F(x)
- This can be efficiently checked using `t.find(s) == string::npos` because the average number of matches before a mismatch is small
  - It is even faster than KMP

M1735 – Fibonacci Word II

- Solution 2:
- s is a substring of F(inf) if and only if it does not contain any of the ***minimal forbidden words***
  - 00, 111, 10101, 00100100, 1010010100101, ...
- How to generate the words?
  - Loop f = 2, 3, 5, 8, 13, 21, 34, ...
    - Let p = the prefix of F(inf) of length f
    - If the next character after p is 0, add "1p1"
    - Otherwise, add "0p0"

M1736
Palindromic Subsequences

Theo

Problem statement

Given a string **S**,

> The subsequences of **S** are the non-empty strings that can be obtained from deleting characters in **S**

E.g. S = “acbba”

“aba” occurs 2 times in S as subsequences

acbba

acbba

Problem statement

A subsequence is said to be palindromic if the subsequence equals itself reversed

E.g.

“aba”, “a”, “abacaba”, “aabbaa” and “abba” are palindromic

“ab”, “aab”, “abacada” are not

Problem statement

You are asked about the number of different palindromic subsequences which differ from subsequence positions

E.g. “aba” is counted twice in “acbba”

Solution

Dynamic programming

We can construct any palindrome starting from the center, each time adding equal characters to both sides of the palindrome

E.g. **bccb** -> **abccba**

Solution

Now, we fix the center at range [l..r], i.e. the center

lies from l to r

E.g.

theprobl[emistooeasyisn]tit

Then we want to find equal characters from red part and blue part each. To be exact, we want to count the number of ways to do so.

E.g.

theprobl[emistooeasyisn]tit

Solution

Let dp[l,r] be the number of palindromic subsequences if we fix the center at [l+1..r-1] (Notice the notation here, so the dp state is well defined iff (l + 2 <= r)

Then, we need to consider if we want to include l-th character into our constructed palindromic subsequence or not

Solution

First case: we do not include the **l**-th character

Then the number of cases equals dp[**l**-1,**r**]

Second case: we do include the **l**-th character

Then, we need to find a corresponding character on the right hand side.

For each **k** so that the **k**-th character equals the **l**-th character and **r** <= **k**, we can select it as the new layer and add it to the constructed palindrome

Therefore, the number of cases equals the sum of dp[l-1,**k**+1] for all satisfying **k**.

Solution

Let N be the length of S

The number of states are O(N^2), and we need to spend O(N) at worst to do the transition, therefore the total complexity is O(N^3)

Which cannot pass system test

Solution

Let H[l,r] be sum(r <= k, a[l]==a[k]: dp[l-1,k+1])

Then dp[l,r] = dp[l-1,r] + H[l,r]

Notice that H[l,r] = H[l,r+1] + 0 if a[l] != a[r]

dp[l-1,r+1] otherwise

Therefore, H[l,r] can be calculated in O(1) per state

Which leads to O(N^2) solution

Things remain

We need to extract the answer from the dp table

For odd-length palindrome, the center is a single character.

Enumerate the center character as **x**-th character, the number of palindrome centered at x is dp[**x**-1,**x**+1]

For even-length palindrome, the center is two identical characters.

Enumerate the center characters as **l**-th character and **r**-th character, then the number of palindrome centered at **l** and **r** is dp[**l**-1,**r**+1]

Both cases can be handled in O(N^2) total.

M1737 Editorial

Alex Tung

Description

- Convert OCT XX to DEC YY (or vise versa)

Conditions:

- XX is octal, YY is decimal
- $XX_8 = YY_{10}$
- OCT XX and DEC YY are valid dates

Solution

- Easy implementation

- For DEC YY -> OCT XX:
  - Convert YY to octal XX
  - Then check OCT XX is valid
- For OCT XX -> DEC YY:
  - Check XX is octal (i.e. both digits < 8)
  - Then convert XX to decimal YY

M1738 – Equation Puzzle

- Use **`next_permutation`** to try all permutations
- Be careful about the division operator
  - **`b != 0 && a / b == c`**
  - **`b * c == a`**

M1739 How to Run Fast

Problem by Charlie Li

Problem statement

- Given a graph with positive weighted edges.
- It is guaranteed that the graph does not contain self-loop.
- Note: the graph is not necessarily connected
- Find the number of shortest paths to all other nodes from node 1.

Observation

- It we collect all the edges which appear in at least one shortest path to any node, if will form a DAG.
  - There are no circles (Since all edges are having positive edge, if there are circle, it would not be a shortest path)
  - It is directed. (walk from smaller distance to larger distance)

- So, we can use DP to find the answer.
- And of course, we need to find the shortest path first.

Solution

- Let dis[i] be the shortest path from node 1 to node i.
- We can use Dijkstra to find all the dis[i] in O($(n+m)\log n$)

- Then we can build a DAG by checking whether a edge (u -> v) satisfy dis[u] + weight = dis[v]

- Define dp[i] = number of shortest paths, initially 0, except dp[1] = 1
- We can perform a topological sort and for every edge (u -> v) deleted, dp[v] += dp[u]

Solution

- There is another solution which is easier to implement, but the idea is the same.
- We can calculate dis[i] and dp[i] and the same time

- When calculating dis[i], if we find that walking through edge (u -> v) gives us better solution (dis[u] + weight < dis[v])
  - We update dis[v] = dis[u] + weight and also dp[v] = dp[u]
- If we find that walking through edge (u -> v) is not worse (dis[u] + weight == dis[v])
  - We update dp[v] += dp[u]