Вказівки щодо розв'язання завдання № 21 відбірково-тренувальних зборів команди міста Києва

1. Згортка

Для розв'язання потрібно, по-перше, зрозуміти, *що* рахувати, і, по-друге — *як* рахувати. Очевидним є "прямий спосіб" — обчислення згідно з означенням, поданим в умові. Такий метод пройде невелику кількість тестів. Впевнившись у швидкому виході за межі базових типів цілих чисел, учасник може прийняти вбивче для себе рішення — реалізувати "довгу арифметику", а саме: додавання, множення, та ділення на 2. Згаявши багато часу, він одержить коректне, але дуже повільне розв'язання зі складністю O(N^4).

Існує цілком інший підхід: при довільному N відповіддю є $C_{2(N-1)}^{N-1} = \binom{2(N-1)}{N-1}$.

Математичне доведення цього факту досить громіздке і набагато складніше, ніж можна було б вимагати на олімпіаді з математики аналогічного рівня — див. ст. 213–214 монографії Грэхем Р., Кнут Д., Паташник О. Конкретная математика. Основание информатики. Пер. с англ. — М.: Мир, 1998. — 703 с., ил. Але саме тим і відрізняється олімпіада з ін форматики від олімпіади з математики: доводити не потрібно, потрібно лише успішно пройти тести.

Тепер подивимось, як саме обчислювати відповідь. Можна реалізувати додавання "довгих чисел" та скористатися трикутником Паскаля. Таке розв'язання достатньо швидке (O(N^3)) і витримує чимало тестів.

Але, мабуть, найбільш оптимальним є такий варіант:

$$C_{2X}^{X} = \frac{(2X)!}{X!(2X-X)!} = \frac{(2X)!}{X!X!}.$$

Розглянемо розклад шуканого числа на прості множники:

$$C_{2X}^{X} = \frac{(2X)!}{X!X!} = p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}.$$

Визначимо, який степінь k_i матиме просте число p_i у цьому розкладі: якщо через $F(K, p)$ позначити степінь простого числа p у розкладі $K!$ на прості множники, то

$$k_i = F(2X, p_i) - 2F(X, p_i), \quad \text{де} \quad F(K,p) = \left[\frac{K}{p}\right] + \left[\frac{K}{p^2}\right] + \left[\frac{K}{p^3}\right] + \ldots .$$

Залишається лише реалізувати множення довгого числа на коротке, та перебрати всі прості множники від 2 до 2(N–1), поступово обчислюючи значення виразу $p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$. Верхня оцінка ефективності такого алгоритму не перевищує O(N^2).

Подане далі авторське розв'язання, прийнятне в умовах проведення олімпіади, можна удосконалити до ідеального щодо кількості виконуваних дій під час проходження тестів без урахування часу компіляції програми. Шукані числа або результати проміжних розрахунків (наприклад, прості числа, що менші за 2000, та їхні степені у розкладі факторіалів натуральних чисел потрібно попередньо

обчислити і зберегти масивом сталих програми-відповіді.

2. Комарі

Умову задача можна сформулювати й так. На координатній площині дано скінчену множину точок. Знайти максимальну кількість точок, які можна накрити прямокутником заданих розмірів зі сторонами, паралельними осям координат.

Потрібно розглянути дві орієнтації прямокутника: "горизонтальну" та "вертикальну", що розрізняються поворотом прямокутника на 90º. Нехай орієнтацію зафіксовано. Позначимо *висоту* прямокутника через *h*, *ширина* — через *w*. Надалі точки множини будемо називати просто *точками* без уточнень. Одне з оптимальних розміщень прямокутника таке, що на його лівій стороні лежить принаймні одна точка. Розглянемо саме такі розташування прямокутника і серед них виберемо найкраще.

Спочатку зробимо деякі допоміжні операції. Впорядкуємо координати точок $(x; y)$ в лексикографічному порядку, а також окремо їх y-координати за час $O(N \cdot \ln N)$. Видалимо дублікати з впорядкованої послідовності y-координат, позначимо її через $\{v_j\}$. Розглянемо функцію $u(y) = \max\{j \mid v_j \le y\}$ (у тексті авторського розв'язання `upper`). Вона повертає номер по порядку найбільшої y-координати, що не перевищує аргументу. Маючи впорядковану послідовність $\{v_j\}$, можна організувати запит $u(y)$ за час $O(\ln N)$ за допомогою бінарного пошуку.

Також знадобиться структура даних DYNAMIC-RANGE-MAX (далі DRM(a)). Вона підтримує послідовність цілочисельних значень `a[1]`, `a[2]`, ... , `a[n]` з початковими нульовими значеннями й має такі:

- функцію `MAX` — повертає max(`a[1]`, `a[2]`, ... , `a[n]`);
- процедуру `UPDATE(i,j,delta)` — для кожного k такого, що $i \le k \le j$, змінює значення `a[k]` на `a[k] + delta`.

Спочатку покажемо, як розв'язати задачу, використовуючи DRM із часом обробки запитів $<O(1), O(\ln N)>$, а потім — як ефективно реалізувати DRM.

Будемо обробляти точки в лексикографічному порядку. На кожному кроці ми розглядаємо деяку множину точок W. Позначимо через x_0 абсцису найлівішої точки з множини W: $x_0 = \min\{x \mid (x; y) \in W\}$. При цьому (на кожному кроці):

- для всіх $(x; y)$ з W справджується нерівність $x - x_0 \le w$;
- W — максимальна за включенням.

На першому кроці W містить точку з найменшою абсцисою, на другому x_0 дорівнює другій по величині абсцисі, і т. і. Таким чином, увесь час точки W можна накрити вертикальною смугою ширини w, яка весь час зсувається праворуч. Позначимо через opt(W) оптимальний розв'язок задачі для кожної такої смуги. Розв'язком всієї задачі буде максимум серед усіх значень opt(W).

Для кожної фіксованої смуги W будемо підтримувати `DRM S` з такою властивістю: `S[i]` дорівнює кількості точок W, які накриває прямокутник з координатами лівого верхнього кута (x_0, v_i). Щоб ефективно підтримувати `S`, будемо робити таке:

- для кожної точки (x; y), яку щойно долучили до W, виконаємо `S.UPDATE(upper(y),upper(y+h),+1);`
- для кожної точки, яку щойно вилучили з W, виконаємо `S.UPDATE(upper(y),upper(y+h),-1).`

На кожному кроці opt(W) — це `S.MAX`. При переході до наступного розташування W вилучимо всі точки з абсцисою x_0, змінимо значення x_0 на наступне значення за x_0 і долучимо до W всі точки, що задовольняють нерівність $x - x_0 \le w$. Таким чином, ми зберігаємо властивість `S` і знайдемо opt(W) для кожного розташування W такого, що на лівій стороні W лежить принаймні одна точка.

Оцінимо складність алгоритму. Для кожної точки, що потрапляє у W, виконується $O(\ln N)$ операцій, а також не більше n запитів `S.MAX`. Таким чином, загальна складність основного етапу теж складає $O(N \cdot \ln N)$.

Тепер покажемо, як побудувати DRM з потрібними оцінками складності. Побудуємо бінарне дерево наступним чином: кореню відповідає інтервал `[1..N]`, і кожен нетривіальний інтервал `v ~ [l..r]` має 2-х нащадків `left[v] ~ [l..m]` та `right[v] ~ [m+1..r]`, де `m = (l+r) div 2`. Кожній вершині співставимо (див. далі псевдокод) значення `M[v]` і `C[v]` таким чином, що справджуються рівності:

- якщо `v` — внутрішня вершина, то

  `M[v]= max{M[left[v]], M[right[v]]} + C[v]`

  інакше

  `M[v] = C[v],`

  тобто

$$\mathrm{M[v]} = \max_u \sum_{w \in T_u} C[w] \qquad (1)$$

  де u – будь-який лист, досяжний від v, а T_u — шлях від v до u;
- для кожного *листа* (тривіального інтервалу) `v ~ [k..k]` справджується рівність:

$$\mathtt{a[k]} = \sum_{w \in T_v} \mathtt{C[w]}, \qquad (2)$$

  де T_v — шлях від кореня дерева до v.

З рівностей (1–2) випливає: `M[]` *на корені дерева дорівнює* `MAX`.

Вибрана структура дозволяє швидко опрацьовувати запити другого роду. Ось його псевдокод:

```
UPDATE(v, a, b, delta)
    # v ~ [l..r]
    # додаємо delta в інтервалі [a..b], що є підінтервалом [l..r]
    if [l..r] = [a..b] then
            C[v] := C[v] + delta
            M[v] := M[v] + delta
            return
    m = (l+r) div 2
```

```
if [l..m] перетинається з [a..b] then
        UPDATE(left[v], a, min{b, m}, delta)
if [m+1..r] перетинається з [a..b] then
        UPDATE(right[v], max{m+1, a}, b, delta)
M[v] = max{M[left[v]], M[right[v]]} + C[v]
return
```

Неважко впевнитись, що час роботи процедури має порядок $O(\ln N)$. Таким чином, DRM можна побудувати з часом запитів $<O(1), O(\ln N)>$.

3. Пірати

Спочатку розберемо наступний приклад: 5 піратів ділять 100 монет. Розглянемо хід думок кожного з піратів:

1. Якщо так станеться, що гроші буде ділити пірат №1 (тобто пірати №2–№5 будуть вбиті), то він, зрозуміло, забере собі всі гроші і сам же проголосує за свій план. Отже, в цьому випадку пірат №1 отримає 100 монет.
2. Якщо ділити гроші буде пірат №2 (пірати 3, 4 і 5 — вбиті), то скільки б він не дав грошей пірату №1, той не підтримає такий розподіл. Крім того, голос пірата №1 не потрібен пірату №2 — він сам може проголосувати за свій план, бо піратів лишилося лише двоє, і це буде необхідна половина голосів. Тому пірат №2 розділить гроші наступним чином: собі — всі 100 монет, пірату №1 — 0 монет.
3. Якщо гроші буде ділити пірат №3, то йому немає сенсу давати гроші пірату №2 — той все одно не підтримає його план (якщо план 3-го пірата не пройде, ділити буде 2-ий і він дістане собі 100 монет). Для того, щоб пропозиція отримала підтримку, пірату №3 потрібен ще один голос, крім свого. Йому залишається тільки купити голос пірата №1. Це можна зробити, давши першому пірату лише одну монету (перший пірат знає, що якщо він не проголосує «за», то ділити гроші буде пірат №2 і тоді 1-му взагалі нічого не дістанеться). Таким чином, 3-ий пірат поділить гроші так: №1 — 1 монета, №2 — 0 монет, №3 — 99 монет.
4. Якщо гроші буде ділити пірат №4, то для підтримки своєї пропозиції йому треба купити голос лише одного пірата. Найдешевше купити голос пірата №2 — йому можна дати одну монету, і №2 підтримає такий розподіл, бо якщо гроші буде ділити №3 — другий пірат отримає 0. Тому розподіл грошей буде таким: №1 — 0, №2 — 1, №3 — 0, №4 — 99.
5. Пірату №5 треба купити 2 голоси. Він дасть по одній монеті піратам №1 і №3 (якщо вони не підтримають такий розподіл, то отримують по 0 монет від пірата №4). П’ятий пірат розділить гроші так: №1 — 1, №2 — 0, №3 — 1, №4 — 0, №5 — 98.

Таким чином, розподіл монет в залежності від того хто ділить буде таким:

Номер пірата, що розподіляє гроші	Розподіл грошей
1	100

2	0 100
3	1 0 99
4	0 1 0 99
5	1 0 1 0 98

Тепер узагальнимо опис поведінки піратів. Для N піратів *альтернативним розподілом* назвемо такий остаточний розподіл, який би мав місце, якщо гроші ділив би пірат з номером $N - 1$ (навіть, якщо його вб'ють). Наприклад, альтернативним розподілом для 5 піратів і 100 монет є такий: 0 1 0 99.

Кожен пірат знає, що якщо він не проголосує за розподіл N-го пірата, то він отримає гроші за альтернативним розподілом. Тому за розподіл пірата з номером N, проголосують тільки ті, що отримають більше ніж при альтернативному розподілі. Для того, щоб його пропозицію підтримали, старшому пірату потрібно «купити» голоси $[(N–1)/2]$ інших піратів (ціла частина від частки $(N–1)/2$). Для цього, він дивиться, які пірати отримують найменше при альтернативному розподілі і дає потрібній кількості з них на одну монету більше (ці пірати проголосують за його розподіл). Все інше золото старший пірат лишає собі. Якщо ж, у пірата є вибір кому давати золото, він спочатку дасть його піратам з меншими номерами (за умовою задачі).

Розглянемо ситуацію, коли пірату не вистачить грошей, щоби купити необхідні йому голоси. У цьому випадку, щоб він не запропонував, його вб'ють, а всі інші пірати отримають гроші за альтернативним розподілом.

Пірати, яких при альтернативному розподілі вбивають, однозначно проголосують за будь-який розподіл старшого пірата (навіть якщо їм дати по 0 монет).

Алгоритм розв'язку можна описати так:

Якщо $N = 1$, то пірат №1 забирає всі гроші. Інакше:

1) рекурентно визначаємо альтернативний розподіл (розподіл при $N –1$);
2) даємо всім піратам, яких при альтернативному розподілі вбивають, по 0 монет;
3) якщо треба докупити ще голосів (всього потрібно $[(N–1)/2]$ голоси), то вибираємо необхідну кількість піратів, що отримують найменше грошей при альтернативному розподілі (якщо таких виборів декілька, обирати треба піратів $[(N–1)/2]$ з найменшими номерами). Даємо кожному з цих піратів на одну монету більше, ніж вони отримують при альтернативному розподілі.
4) якщо пірату вистачає грошей для розподілу, то решту він бере собі, інакше його вбивають, а інші пірати отримають гроші за альтернативним розподілом.

Реалізація цього алгоритму «в лоб» буде працювати не швидше ніж за час $O(N^2)$. Для отримання швидшого розв'язку, потрібно відмовитись від моделювання поведінки піратів. Натомість, можна дослідити деякі властивості шуканого розподілу грошей.

Твердження 1. *Якщо* $M \geq [(N–1)/2]$, *то старший пірат здійснить такий розподіл, при якому пірати з номерами такої ж парності як і номер старшого пірата отримають по одній монеті, а всі інші пірати по* 0 *монет. Решту грошей*

забере старший пірат.

Доведення (методом математичної індукції за N).

1. База $N = 1$ (очевидно, твердження вірне)
2. Припустимо, що твердження вірне для $N - 1$ пірата.
3. N-ий пірат має купити $[(N–1)/2]$ голосів. Знайдемо, скільки піратів отримують 0 монет при альтернативному розподілі:
   $(N - 1) - ([(N - 2)/2] + 1) = N - [N/2] - 1 = [(N–1)/2]$.
   Отже, старшому пірату достатньо заплатити тільки тим піратам, що при альтернативному розподілі отримують 0 монет. Він дасть їм по 1 монеті, а всім іншим піратам 0. Парність номерів піратів, що отримають гроші буде відрізнятися від парності номерів, що отримали б гроші за альтернативним розподілом. Твердження доведено.

Завдяки доведеному твердженню 1, можна одразу ж отримати розв'язок для випадку $M \geq [(N - 1)/2]$ (складність роботи алгоритму $O(N)$).

При $M < [(N–1)/2]$ необхідно модифікувати алгоритм. Старшого пірата не вбивають, якщо за нього проголосує M піратів, яким він дає по одній монеті, і ще принаймні $[(N–1)/2]–M$ піратів, яких уб'ють, якщо вони проголосують проти. Відповідний підрахунок можна здійснити за один лінійний прохід.

Для цього запровадимо функцію $K(n, M)$ — кількість піратів що вб'ють, якщо n піратів будуть ділити M монет. Для $K(n, M)$ справджується таке:

1) Якщо $M \geq [(n–1)/2]$ то $K(n, M) = 0$,
2) Якщо $M+K(n–1, M) \geq [(n–1)/2]$, то $K(n, M) = 0$,
3) Якщо не (1) чи (2) то $K(n,M) = K(n–1, M) +1$.

$K(N, M)$ можна обчислити за $O(N)$. $K(N, M)$ найстарших піратів буде вбито, а золото буде розподіляти пірат з номером $N–K(N, M)$.

Складність роботи алгоритму є $O(N)$.