Вказівки щодо розв'язання завдання № 32 відбірково-тренувальних зборів команди міста Києва

1. Голосування

Нехай D — деяка підмножина присутніх на виборах делегатів. Залишимо у таблиці голосування голоси лише тих делегатів, що належать до підмножини D. Числа в комірках нової таблиці, очевидно, не перевищують відповідні числа у початковій таблиці (але останнє число в кожному стовпці вже не обов'язково є найменшим).

Лема. *Усі делегати з підмножини D можуть бути послідовними в тому й лише у тому випадку, якщо в кожному рядку нової таблиці числа йдуть в порядку неспадання.*

Доведення. Якщо послідовний делегат голосує в деякому раунді за певну заявку, то він продовжуватиме голосувати за цю ж заявку доти, доки її не буде виключено. Тож якщо всі делегати з множини D послідовні, то всі, хто голосував за певну заявку в деякому раунді, будуть голосувати за неї і в наступному. Таким чином, числа в рядках ідуть у порядку неспадання.

Тепер, навпаки, нехай числа в рядках таблиці йдуть у порядку неспадання. Тоді можна безпосередньо побудувати приклад уподобань, які міг би мати кожен делегат із множини D. Позначимо число, що стоїть на перетині i-го рядка та j-го стовпця, через a_{ij}, $1 \le i \le n$, $1 \le j \le n + 1 - i$ (тут ми покладаємо a_{1n} рівним загальній кількості делегатів у множині D, тобто проводимо фіктивний додатковий n-й раунд, у якому всі голосують за єдину заявку, що лишилася). В цих позначеннях рівно у a_{11} делегатів першою у списку стоїть заявка 1, у a_{21} делегатів першою у списку стоїть заявка 2, ..., у a_{n1} делегатів першою у списку стоїть заявка n. Після першого раунду голоси тих, у кого першою в списку була заявка n, перерозподіляються на користь інших заявок. Інакше кажучи, другою в списку у кожного делегата, що голосував за n-ту заявку, маємо поставити одну з перших $n - 1$ заявок, а саме: у $a_{12} - a_{11}$ із цих делегатів другою в списку буде стояти перша заявка, у $a_{22} - a_{21}$ із них другою в списку буде друга заявка, ..., у $a_{(n-1)2} - a_{(n-1)1}$ із них другою в списку буде $(n - 1)$-ша заявка. Аналогічно після другого раунду голоси тих, у кого останньою на даний момент у списку була заявка $n - 1$, перерозподіляються на користь перших $n - 2$ заявок. Тобто наступною в списку у кожного делегата, що голосував за $(n - 1)$-шу заявку, маємо поставити одну з перших $n - 2$ заявок, а саме: у $a_{13} - a_{12}$ із них наступною в списку буде стояти перша заявка, у $a_{23} - a_{22}$ із них наступною в списку буде друга заявка, ..., у $a_{(n-2)3} - a_{(n-2)2}$ із них наступною в списку буде $(n - 2)$-га заявка. Продовжуємо такі самі операції далі. На останньому кроці, після $(n - 1)$-го раунду, $a_{2(n-1)}$ голосів «перерозподіляються» між єдиною заявкою, що залишилася, — тобто наступною у списку у кожного делегата, що голосував за другу заявку, маємо поставити першу заявку.

Звичайно, у такий спосіб ми не обов'язково повністю заповнимо списки кожного делегата. Наприклад, ті, хто голосував за першу заявку починаючи вже з першого раунду, голосуватимуть за неї до кінця, і, що йде у їхніх списках після першої заявки, ми не визначимо. Разом із тим у кінець кожного незаповненого списку ми можемо дописати решту заявок (яких іще не було у списку) в довільному порядку, не змінивши результатів голосування в жодному з раундів. Лему доведено.

Отже, задачу можна звести до такої: знайти таблицю T, числа в комірках якої не перевищують відповідні числа початкової таблиці, в кожному рядку числа йдуть у

порядку неспадання, а сума чисел у кожному стовпці однакова і якомога більша. Така сума (чисел одного стовпця) і є шуканим значенням m. За допомогою ж алгоритму, наведеного в доведенні леми, з таблиці T можна відновити і приклад уподобань усіх m послідовних делегатів. Залишається знайти відповідну таблицю T.

Спершу замінімо у початковій таблиці кожне число на найменше серед чисел, що стоять праворуч від нього, і його самого. Отриману таблицю позначимо через U. У таблиці U числа в рядках ідуть у порядку незменшення. За побудовою числа в комірках шуканої таблиці T не можуть перевищувати відповідні числа таблиці U, тому значення m не може бути більшим за суму чисел довільного стовпця таблиці U, зокрема й за найменшу з таких сум. Насправді найменша із цих сум і є шуканим значенням m: на базі таблиці U нам вдасться побудувати таблицю T таким чином, щоб кожен стовпчик таблиці T мав суму m. Для цього спершу, якщо потрібно, довільним чином зменшимо числа в першому стовпці таблиці, щоб сума чисел у ньому стала рівною m. Далі, якщо потрібно, зменшимо числа у другому стовпці так, щоб вони не стали меншими за своїх сусідів зліва, але їхня сума також виявилася рівною m (це можна зробити, адже якщо зменшити всі числа максимально, тобто так, щоб вони стали рівними своїм сусідам зліва, то сума чисел другого стовпця не буде перевищувати суму першого стовпця, тобто числа m). Далі зменшимо у такий же спосіб величини у третьому стовпці, не порушивши монотонності чисел у рядках і досягнувши того, що сума вже й у третьому стовпці стане рівною m. Продовжуватимемо цю операцію, поки сума не стане однаковою і рівною m в усіх стовпцях. Отримаємо таблицю T.

Додамо, що за умови оптимальної реалізації наведений розв'язок дає відповідь за $O(n(n + m))$ часу. Разом із тим у контексті олімпіади задача була розрахована більше на ідейне розв'язання і мала відносно невеликі обмеження, тож повний бал можна було набрати, не зосереджуючись занадто на деталях реалізації.

Насамкінець зауважимо, що в дійсності процедура голосування за місто, що прийматиме Олімпійські ігри, має незначні відмінності від описаної в задачі: по-перше, в раунді не можуть брати участь делегати, що представляють країни, заявки яких беруть у цьому раунді участь; по-друге, якщо якась із заявок набере понад 50 % голосів у довільному раунді, голосування припиняють достроково; по-третє, в разі рівності голосів у кількох заявок, що претендують на виліт, між ними проводять додатковий проміжний раунд. Пропонуємо читачу самостійно подумати над тим, як саме ці зміни впливають на задачу визначення найбільшої підмножини послідовних делегатів.

2. Перегони

Розглянемо спочатку інтуїтивно простіший випадок $b \le a$. Якщо оптимальний план харчування передбачає споживання батончиків до початку перегонів, то їх споживання можна пересунути вперед на моменту початку перегонів. Вимоги щодо енергозабезпечення не буде порушено, бо $b \le a$. Так і зробимо. Для того, щоб задовольнити початкову потребу у потужності p_0, потрібно спожити щонайменше (p_0 / a) батончиків у момент 0 (якщо частка не є цілою, потрібно округлити до найближчого цілого, що не менше за цю частку). Споживати більше нема сенсу, бо зсунувши споживання додаткових батончиків на 1 секунду у майбутнє, ми можемо лише виграти, бо $b \le a$.

Після розгляду першої секунди перегонів отримуємо задачу для $(n - 1)$ секунд з модифікованими (зменшеними) величинами p (бо частину потреб у перші $t_1 + t_2$ секунд

вже компенсовано). Таким чином, розв'язання зведено до використання жадібного алгоритму: рухаючись від початку до кінця перегонів, у кожен момент часу споживати найменшу кількість батончиків, що покривають поточні потреби з урахуванням того, що було спожито раніше.

У випадку $b > a$ розглянемо час у зворотному напрямку.

Викладену ідею можна реалізувати за лінійний час. Спочатку перейдемо до величин $q_j = p_j - p_{j-1}$, так що $p_j = q_0 + q_1 + ... + q_j$. Тоді споживання батончика у момент k призводить до зміни лише трьох таких чисел q_j при $j = k$, $k + t_1$, $k + t_1 + t_2$. Розглядаючи масиву q у порядку зростання індексу, ми завжди знатимемо поточну потребу у потужності.

Структура даних із швидкими запитами на відрізку (дерево відрізків, «коренева ідея» тощо) також дає можливість вибрати всі бали.

3. Перехрестя

Вилучаючи по черзі кожну точку графа — схеми шляхів разом з відповідними ребрами й використовуючи пошук у ширину від довільної іншої вершини, можна встановити, чи втрачено зв'язність решти графа після такого вилучення. Але таким чином можна набрати лише частину балів.

Відомий спеціальний алгоритм пошуку точок сполучення, який подано далі разом з елементами теорії графів для його обґрунтування.

Означення 1. *Запровадимо такі поняття:*

1. *Підграф $G'(V', E')$ графа $G(V, E)$ називають остовним (каркасним), якщо $V' = V$, тобто множини вершин графа G і підграфа G' збігаються.*
2. *Остовним (каркасним) деревом графа називають його довільний остовний (каркасний) підграф, що є деревом.*
3. *Вершину v зв'язаного неорієнтованого графа G називають точкою сполучення або вершиною, що розрізає (розділяє) граф, якщо вилучення цієї вершини разом з інцендентними їй ребрами (що мають її за кінець) призводить до порушення зв'язності графа*.

Теорема 1. *Нехай орієнтоване дерево T є остовним (каркасним) деревом нерієнтованого графа $G(V, E)$, побудоване пошуком у глибину, і вершини a, b сполучено ребром графа $G(V, E)$. Тоді або a є нащадком b, або b є нащадком a.*

Теорема 2. *Нехай орієнтоване дерево T є остовним (каркасним) деревом нерієнтованого графа $G(V, E)$, побудоване пошуком у глибину. Вершина a є точкою сполучення графа $G(V, E)$ тоді й лише тоді, коли вершина a:*

- *або є коренем дерева T, що має більше одного безпосереднього нащадка;*
- *або не є коренем дерева T, та існує такий нащадок a, який разом зі своїми нащадками не сполучено жодним ребром у графі $G(V, E)$ з жодним з предків a у дереві T.*

Доведення.

- Нехай вершина a є коренем дерева T. Тоді для довільних двох вершин, відмінних від a, існує ланцюг (шлях без врахування напряму дуг дерева T), що проходить через вершину a:

- якщо корінь a має лише одного безпосереднього нащадка b, то, вилучивши ланки (b; a) і (a; b), отримаємо ланцюг, що не проходить через a;
- якщо корінь a має щонайменше два різні безпосередні нащадки, то довільний ланцюг, що сполучає їх, проходить через корінь a (див. пошук у глибину, починаючи від одного з таких нащадків).

- Якщо a не є коренем побудованого дерева, позначимо через p безпосереднього предка a.
  - Нехай для деякого нащадка a немає жодного ребра графа $G(V, E)$, що сполучає його чи його нащадків з p. Тоді всі шляхи у графі $G(V, E)$, що ведуть від цього нащадка a до p, проходять через a. У цьому випадку вершина a є точкою сполучення, бо її вилучення разом з інцендентними їй ребрами робить неможливим сполучення шляхом вказаного нащадка a з p.
  - Нехай для всіх нащадків a існує ребро графа $G(V, E)$, що сполучає цього нащадка або котрогось з наступних нащадків з вершиною p або її предком. Тоді вилучення вершини a разом з інцендентними їй ребрами:
    - не порушує можливості сполучення шляхами вершин графа $G(V, E)$ без a та її нащадків;
    - не порушує можливості сполучення шляхами нащадків a з p, а через p — з іншими вершинами графа $G(V, E)$, відмінними від a.

    У цьому випадку вершина a не є точкою сполучення.

У початковий момент виконання поданого далі алгоритму:

- величина лічильника i дорівнює 0;
- кожна вершина має мітку «нова»;
- кількість виявлених безпосередніх нащадків кореня дорівнює 0.

У процесі виконання алгоритму обчислюємо такі велични:

- $g(v)$ — порядковий номер у порядку приєднання вершини v до дерева T, яке будуємо пошуком у глибину;
- $f(w)$ — найменший порядковий номер у порядку приєднання вершини x до дерева T, при якій існує ребро, що не перетворене на дугу дерева T і сполучає x або з w, або з нащадком w.

Нехай вершина w є безпосереднім нащадком вершини v і справджується така нерівність: $g(v) < f(w)$. Тоді немає ребра початкового графа, що не перетворене на дугу дерева T і сполучає нащадка w з безпосереднім предком v. Згідно з доведеною теоремою вершина v є точкою сполучення у цьому випадку.

За параметр v для першого виклику вибираємо довільну вершину заданого графа, наприклад з номером 1.

Рекурсивний алгоритм пошуку точок сполучення — РАПТС(v)

1. Змінюємо мітку v на «використана».
2. Збільшуємо величину i на 1.
3. Надамо величини $f(v) = g(v) = i$.
4. Перебираємо всі суміжні з v вершини w. Якщо w має мітку «нова», робимо таке:
   - долучаємо дугу (v; w) до дерева, вважаючи v безпосереднім предком вершини w;
   - виконуємо РАПТС(w);

- якщо $g(v) \leq f(w)$ і v не є коренем дерева, то оголошуємо вершину v точкою сполучення;
- якщо v є коренем дерева, то збільшуємо кількість виявлених безпосередніх нащадків кореня на 1. Якщо ця кількість більша ніж 1, оголошуємо корінь точкою сполучення (останню дію можна виконати і по завершенню виконання алгоритму першого виклику);
- замінюємо величину $f(v)$ на $\min(f(v), f(w))$.

Інакше, тобто якщо вершина w має мітку «використана», а v не є безпосереднім предком w, замінюємо величину $f(v)$ на $\min(f(v), g(w))$.

Саме цей алгоритм реалізовано у розв'язанні, за яким відкалібровано час на виконання тестів.