ЗАДАЧІ XXVI ВСЕУКРАЇНСЬКОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ РОЗВ'ЯЗУВАННЯ

Бондаренко Віталій Вікторович,
асистент факультету кібернетики Київського Національного університету ім. Тараса Шевченка,
Ягіяєв Шаміль Ігорович
менеджер проектів компанії «Арісент Україна».

ЗАВДАННЯ ДРУГОГО ТУРУ

1. «Конструктор» (Данило Мисак)

На свій перший день народження Меґґі Сімпсон, персонаж мультсеріалу «Сімпсони», отримала в подарунок конструктор: ігровий набір, що складається з паличок різної довжини. Кінці паличок можна скріплювати, причому з'єднані так палички можуть утворювали довільний ненульовий кут, крім розгорнутого (180°). Меґґі хоче скласти опуклий многокутник, використавши якомога більшу кількість паличок із конструктора як сторони цього багатокутника.

Завдання. Напишіть програму **set**, що за розмірами паличок у конструкторі визначить, чи вдасться Меґґі скласти з паличок опуклий многокутник, і якщо вдасться, то визначить, яку найбільшу кількість паличок вона зможе для цього використати.

Вхідні дані. У першому рядку вхідного файлу **set.dat** указано кількість N паличок у наборі, $2 \leqslant N \leqslant 10^5$. У другому рядку записано N натуральних чисел, менших за 10^9 (не обов'язково попарно різних) — довжини паличок.

Вихідні дані. Вихідний файл **set.sol** повинен містити єдине число — найбільшу кількість паличок з набору, з яких можна скласти опуклий многокутник, або число 0, якщо скласти опуклий многокутник не вдасться.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 40% балів: $2 \leqslant N \leqslant 15$.
- 30% балів: $15 < N \leqslant 3000$.
- 30% балів: $3000 < N \leqslant 10^5$.

Крім того, у тестах на 25 % балів правильна відповідь не перевищує 4.

Приклади вхідних та вихідних даних

set.dat	set.sol
4 5 1000 5 5	3
3 1 2 3	0

Рекомендації щодо розв'язання

Під опуклим многокутником будемо розуміти строго опуклий многокутник, тобто такий, усі кути якого строго менші за 180°. Доведемо спершу допоміжне твердження, що є узагальненням нерівності трикутника для випадку опуклого N-кутника.

Лема 1. *Нехай задано $N \geqslant 1$ відрізків додатних довжин $d_1, d_2, \ldots, d_N$. Із них можна скласти опуклий многокутник тоді й лише тоді, коли довжина найбільшого відрізка менша за суму довжин решти $N-1$ відрізків, тобто коли $2\max\{d_1, d_2, \ldots, d_N\} < d_1 + d_2 + \ldots + d_N$.*

Доведення. Нехай із заданих відрізків вдалося скласти опуклий N-кутник $A_1A_2 \ldots A_N$. Хай, без утрати загальності, A_1A_N — найдовший з усіх відрізків. Послідовно використовуючи $N-2$ рази нерівність трикутника, матимемо

$$A_1A_2+A_2A_3+A_3A_4+ \ldots +A_{N-i}A_N > A_1A_3+A_3A_4+ \ldots$$
$$+A_{N-1}A_N > A_1A_4+ \ldots +A_{N-1}A_N > \ldots > A_1A_N.$$

Отже, нерівність з умови леми справджується.

Доведімо тепер зворотне твердження: якщо нерівність для відрізків довжин $d_1, d_2, \ldots, d_N$ справджується, то з них можна скласти опуклий багатокутник. Легко бачити, що при $N=1$ та $N=2$ нерівність виконуватися не може. Отже, $N \geqslant 3$. Не втрачаючи загальності, припустимо, що найдовшим із відрізків є d_N, тобто $d_i \leqslant d_N$, $1 \leqslant i \leqslant N$.

Нехай на площині зафіксували коло радіуса $R \geqslant d_N/2$ із центром у точці O, а також деяку декартову систему координат, причому коло дотикається до прямої $x=0$ (вісь ординат) у точці (0, 0) (початок координат). Визначимо на колі точки $A_1, A_2, \ldots, A_N$ так:

- A_1 — точка (0, 0);
- A_2 — перша за рухом годинникової стрілки точка після A_1 на колі така, що $A_1A_2 = d_1$;
- A_3 — перша за рухом годинникової стрілки точка після A_2 на колі така, що $A_2A_3 = d_2$;

...

- A_N — перша за рухом годинникової стрілки точка після A_{N-1} на колі така, що $A_{N-1}A_N = d_{N-1}$.

Оскільки за побудовою діаметр кола не менший за кожен із відрізків $d_1, d_2, \ldots, d_{N-1}$, конструкцію визначено коректно. Приклад розташування точок зображено на рис. 1.

Доведемо, що, взявши коло достатньо великого радіуса, ми зможемо забезпечити одночасне виконання таких двох умов:

1) $\angle A_1OA_2 + \angle A_2OA_3 + \ldots + \angle A_{N-1}OA_N < 180^\circ$ (звідси, зокрема, випливатиме, що ламана $A_1A_2 \ldots A_N$ не має самоперетинів);

2) $A_1A_N > d_N$.

Щоб довести цей факт, розглянемо довільні дві послідовні хорди $A_{k-1}A_k$ та A_kA_{k+1}, $2 \leqslant k \leqslant N-1$ (рис. 2). Нехай $\angle A_{k-1}OA_k = 2\alpha$, а $\angle A_kOA_{k+1} = 2\beta$.

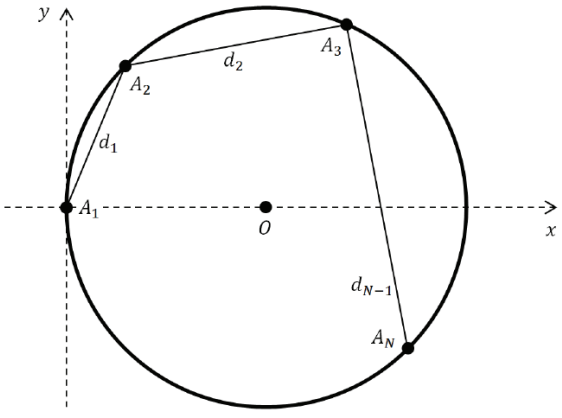

Рис. 1

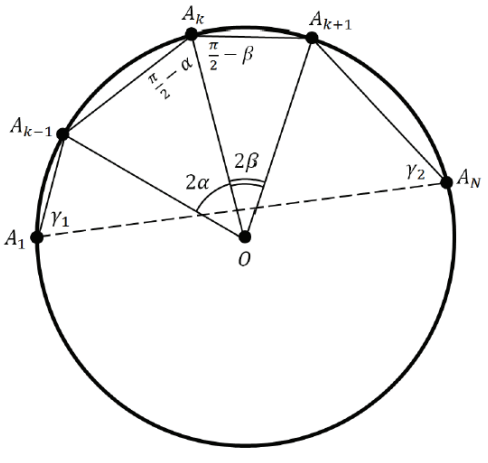

Рис. 2

Оскільки $OA_{k-1}=OA_k=OA_{k+1}=R$, трикутники $A_{k-1}OA_k$ та A_kOA_{k+1} рівнобедрені, а тому

$$\angle A_{k-1}\mathbf{A_k}O=\pi/2-\alpha,\quad \angle A_{k+1}A_kO=\pi/2-\beta,$$
$$\angle A_{k-1}A_kA_{k+1}=\angle A_{k-1}A_kO+\angle A_{k+1}A_kO=\pi-\alpha-\beta.$$

Як відомо, $A_{k-1}A_k=2R\sin\alpha$, звідки

$$\alpha=\arcsin\frac{A_{k-1}A_k}{2R}=\arcsin\frac{d_{k-1}}{2R}\to 0,\ R\to\infty.$$

Аналогічно $\beta\to 0, R\to\infty$. Тому

$$\angle A_1OA_2+\angle A_2OA_3+\ldots+\angle A_{N-1}OA_N\to 0, R\to\infty;$$
$$\angle A_{k-1}A_kA_{k+1}=\pi-\alpha-\beta\to\pi, R\to\infty.$$

Отже, виконання умови 1) при достатньо великих значеннях R доведено. Щоб довести виконання умови 2), розглянемо многокутник $A_1A_2\ldots A_N$ при значеннях R, що задовольняють умову 1). Сума кутів N-кутника складає, як відомо, $(N-2)\pi$. Якщо позначити кути $A_NA_1A_2$ і $A_1A_NA_{N-1}$ через γ_1 і γ_2 відповідно, матимемо

$$\gamma_1+\gamma_2=(N-2)\pi-\sum_{k=2}^{N-1}\angle A_{k-1}A_kA_{k+1}\to 0,\ R\to\infty\Rightarrow$$
$$\Rightarrow\max\{\gamma_1,\gamma_2\}\to 0,\ R\to\infty.$$

Позначимо через φ_i, $1\leqslant i\leqslant N-1$, кут, який утворюють прямі A_iA_{i+1} та A_1A_N. Якщо прямі паралельні, цей кут дорівнює 0. Якщо точка перетину прямих лежить на промені A_NA_1, то кут φ_i не перевищує γ_1 (дорівнює γ_1, коли $i=1$, та менший за γ_1, коли $i>1$). А якщо точка перетину прямих лежить на промені A_1A_N, то кут φ_i не перевищує γ_2 (дорівнює γ_2, коли $i=N-1$, та менший за γ_2, коли $i<N-1$). Отже, $\varphi_i\leqslant\max\{\gamma_1,\gamma_2\}$, $1\leqslant i\leqslant N-1$. А тому

$$A_1A_N=\sum_{i=1}^{N-1}A_iA_{i+1}\cos\varphi_i\geq\sum_{i=1}^{N-1}A_iA_{i+1}\cos\max\{\gamma_1,\gamma_2\}\to$$
$$\to\sum_{i=1}^{N-1}A_iA_{i+1}=\sum_{i=1}^{N-1}d_i>d_N,\ R\to\infty.$$

Отже, виконання умови 2) при достатньо великих R також доведено.

Візьмемо деяке достатньо велике число $R_1>d_N/2$, для якого умови 1) і 2) виконані, і почнемо неперервно зменшувати радіус кола. При цьому точки $A_2, A_3, \ldots, A_N$ «ковзатимуть» по колу в напрямку годинникової стрілки. Припинимо процес «стискання» кола, щойно справдиться хоча б одне з двох тверджень:

- радіус кола стане рівним $d_N/2$ або
- точка A_N збігатиметься з точкою A_1.

Ураховуючи природу першої умови зупинки, рано чи пізно ми обов'язково припинимо процес. Позначимо радіус кола, на якому відбулася зупинка, через Ro. Нехай функція $f:[Ro,R_1]\to[0,+\infty)$ визначає відстань між точками A_1 та A_N при заданому радіусі $R\in[Ro,R_1]$. Незалежно від того, яка з умов послужила причиною зупинки, виконується нерівність $f(Ro)\leqslant d_N$. Справді, якщо $Ro=d_N/2$, то відстань між будь-якими двома точками на колі не перевищує $2Ro=d_N$. А якщо $A_N=A_1$, то взагалі $f(Ro)=0$. У той же час із визначення числа R_1 маємо $f(R_1)>d_N$. Оскільки функція f неперервна, з цих співвідношень випливає, що існує число $R\in[Ro,R_1]$, для якого $f(R)=d_N$. Зважаючи на те, що це значення R ми «пройшли» до того, як точка A_N уперше збіглася з A_1, можемо стверджувати, що при відповідному радіусі кола замкнена ламана $A_1A_2\ldots A_N$ — уписаний (а отже, опуклий) N-кутник без самоперетинів із довжинами сторін, що дорівнюють $d_1, d_2, \ldots d_N$. Лему доведено.

Повернімося тепер до самої задачі — нехай N знову позначає кількість заданих у вхідному файлі довжин, а через M позначимо найбільше значення, якого може набувати довжина одного відрізка (в даному випадку $M=10^9-1$).

Ідейно найпростіший спосіб розв'язати задачу — перебрати всі підмножини заданої множини відрізків і, скориставшись твердженням леми, для кожної підмножини визначити, чи задає вона сторони опуклого багатокутника. Час виконання такого алгоритму дорівнює $O(N\cdot 2^N)$; або $O(N^2\cdot 2^N)$, якщо перевіряти виконання нерівності многокутника не лише для найбільшої, а для всіх його сторін. Перебірні алгоритми набирають 40 % від загальної кількості балів.

Задачу можна розв'язувати так. Спершу відсортуємо заданий масив чисел. Позначимо послідовність

чисел, утворену в результаті, як $d_1 \leqslant d_2 \leqslant \ldots \leqslant d_N$. Підрахуємо суму $S_N = d_1 + d_2 + \ldots + d_N$. Якщо подвоєне найбільше число в масиві d_N є меншим за цю суму (тобто задовольняє нерівність многокутника), то відповідь — N. Інакше незалежно від того, які відрізки стануть у підсумку сторонами многокутника, відрізка довжини d_N бути серед них не може, адже нерівність многокутника це число задовольнити ніяк не зможе. Тому відкинемо його та, перерахувавши суму $S_{N-1}=d_1+d_2+\ldots+d_{N-1}=S_N-d_N$, повторимо ті самі дії для числа d_{N-1}: якщо $2d_{N-1}<S_{N-1}$, то з відрізків довжин $d_1, d_2, \ldots, d_{N-1}$ можна утворити опуклий многокутник, а тому відповідь — число $N-1$. А якщо $2d_{N-1} \geqslant S_{N-1}$, то відрізок довжини d_{N-1} не може бути стороною многокутника, тож слід відкинути число d_{N-1} і перерахувати суму $S_{N-2}= S_{N-1}- d_{N-1}$. Такі операції повторюємо доти, доки не знайдемо відповідь або не відкинемо всі відрізки. Останнє означатиме, що із заданих відрізків скласти опуклий многокутник неможливо.

Під час підрахунку суми чисел треба зважити на те, що вона може виявитися досить великою і, на відміну від самих чисел, не вміститися у чотирибайтову змінну. Тому слід або використати відповідний тип даних, або припиняти підрахунок суми, коли стає зрозуміло, що вона більша за $2M$, а отже, перевищує подвоєну довжину будь-якого із заданих відрізків.

Складність наведеного алгоритму лінійна, якщо знехтувати сортуванням масиву на початку виконання. Отже, залежно від реалізації сортування можна досягти ефективності $O(N^2)$ і набрати 70% балів або вкластися в $O(N \log N)$ і заробити повний бал.

Інший ефективний спосіб розв'язати задачу ґрунтується на твердженні такої леми.

Лема 2. Перед тим як вищенаведений алгоритм завершить свою роботу, він відкине щонайбільше $[\log_2 M]+2$ відрізки (де $[\log_2 M]$ позначає найбільше ціле число, що не перевищує $\log_2 M$).

Доведення. Нехай алгоритм відкинув l відрізків завдовжки $d_N, d_{N-1}, \ldots, d_{N-l+1}$, $l \leqslant N$.

Якщо $l<N$, то маємо $d_1 \geqslant 1$, $d_2 \geqslant 1$, ..., $d_{N-l} \geqslant 1$, а далі, враховуючи, що наступні числа алгоритм відкинув,

$d_{N-l+1} \geqslant d_1+d_2+\ldots+d_{N-l} \geqslant N-l$,

$d_{N-l+2} \geqslant d_1+d_2+\ldots+d_{N-l}+d_{N-l+1} \geqslant$

$\geqslant (N-l)+(N-l)=2(N-l)$,

$d_{N-l+3} > d_1+\ldots+d_{N-l}+d_{N-l+1}+d_{N-l+2} \geqslant$

$\geqslant (N-l)+(N-l)+2(N-l)=2^2(N-l)$,

...

$d_N \geqslant (1+1+2+2^2+\ldots+2^{l-2})(N-l)=2^{l-1}(N-l)$.

Тому $M \geqslant d_N \geqslant 2^{l-1}(N-l) \geqslant 2^{l-1}$, звідки $l \leqslant [\log_2 M]+1$.

Якщо $l=N$, то маємо $d_1 \geqslant 1$, $d_2 \geqslant 1$ а далі

$d_3 \geqslant d_1+d_2 \geqslant 2$.

$d_4 \geqslant d_1+d_2+d_3 \geqslant 1+1+2=2^2$.

$d_5 \geqslant d_1+d_2+d_3+d_4 \geqslant 1+1+2+2^2=2^3$.

...

$d_N \geqslant 1+1+2+2^2+\ldots 2^{N-3}=2^{N-2}$.

Тому $M \geqslant d_N \geqslant 2^{N-2}=2^{l-2}$, звідки $l \leqslant [\log_2 M]+2$.

Лему доведено.

Отже, алгоритм, який не сортуватиме масив, а просто щоразу після відкидання елемента заново шукатиме найбільше число, буде мати час виконання порядку $O(N \log M)$. Така ефективність теж оцінюється повним балом.

Однак твердження леми 2 дозволяє побудувати навіть швидший метод розв'язання задачі, який учасникам зовсім не обов'язково було втілювати, щоб отримати повний бал, але який ми наведемо з теоретичного інтересу.

Не будемо сортувати масив, а натомість за лінійний час упорядкуємо елементи масиву так, щоб на останніх $m=[\log_2 M]+3$ місцях стояли (у довільному порядку) саме ті числа, які б стояли на цих m місцях, якби масив було відсортовано в порядку неспадання. За лінійний час цю операцію дозволяють виконати так звані алгоритми вибору порядкових статистик (selection algorithms), деталі реалізації яких можна знайти в інтернеті. Після цього елементи на останніх m місцях слід відсортувати (за $O(m \log m)$ часу), а далі діяти так, начебто відсортовано увесь масив. Оскільки згідно з лемою 2 алгоритм відкине щонайбільше $m-1$ елемент, то до невідсортованої частини масиву він просто не дійде. Сумарний час виконання програми складатиме $O(N+\log M \log\log M)$, що на практиці означає лінійну (за кількістю чисел у вхідному файлі) ефективність.

Насамкінець кілька слів стосовно того, наскільки реально було написати розв'язок задачі учасникам, не знайомому до олімпіади із твердженням леми про нерівність многокутника. Думка автора така: виходячи з того, наскільки широко відомою є нерівність трикутника і наскільки просто обґрунтовується твердження про нерівність многокутника в один бік, учасник, пробуючи розв'язати задачу, цілком міг щонайменше висунути гіпотезу про нерівність многокутника. З огляду на засади проведення олімпіади, перевірка гіпотези не вимагала від учасника побудови доведення в інший бік: достатньо було написати програму, що ґрунтується на гіпотезі, здати її й подивитися на результат.

2. «Ковпачок» (Данило Мисак)

Нудьгуючи на одному з уроків, відмінник Петро П'яточкін придумав собі розвагу. Він замалював ручкою деякі клітинки прямокутного аркуша, вирваного із зошита, зняв із ручки ковпачок та поставив його на одну із зафарбованих клітин. Далі Петрик послідовно переставляє ковпачок з однієї замальованої клітинки на іншу замальовану клітинку, яка міститься в тому ж рядку або в тому ж стовпчику, що і попередня. Петрик вибрав деяку зафарбовану клітину й хоче перемістити туди ковпачок із початкової клітини за якомога меншу кількість ходів.

Завдання. Напишіть програму **cap,** що за даними про розміри аркуша паперу, конфігурацію зафарбованих клітин, розміщення ковпачка та цільової клітини знайде найменшу можливу кількість переставлянь, за які Петрик зможе перемістити ковпачок з по-

чаткової клітини до цільової, керуючись придуманими ним правилами.

Вхідні дані. У першому рядку вхідного файлу **cap.dat** записано два цілих числа: кількість рядків N і кількість стовпчиків M клітинок, із яких складається аркуш, $2 \leqslant M \leqslant N \leqslant 1000$. Кожен із наступних N рядків містить по M символів:

- **x** (маленька літера латинського алфавіту) — зафарбована клітина.
- **.** (крапка) — порожня клітина.
- **o** (маленька літера латинського алфавіту) — початкова клітина.
- **+** (плюс) — цільова клітина.

У вхідних даних задано рівно одну початкову та рівно одну цільову клітину.

Вихідні дані. Вихідний файл **cap.sol** повинен містити єдине число — найменшу кількість переміщень ковпачка, які потрібно зробити Петрику задля досягнення мети. Якщо ж відповідно до заданих правил не можна досягти цільової клітини, то слід вивести –1.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 30% балів: $2 \leqslant M \leqslant N \leqslant 10$;
- 30% балів: $10 < M \leqslant N \leqslant 100$;
- 40% балів: $100 < M \leqslant N \leqslant 1000$.

Приклади вхідних та вихідних даних

cap.dat	cap.sol
3 2 x+ xx o.	2
4 4 .o.x x.x. .x.x x.+.	–1

Рекомендації щодо розв'язання

Аркуш паперу можна подати як граф, вершинами якого є зафарбовані клітинки, а ребро між двома вершинами проведене тоді й лише тоді, коли відповідні їм клітинки містяться в одному рядку або в одному стовпчику. Природний підхід до розв'язання задачі — здійснити пошук у ширину на такому графі, починаючи з вершини, що відповідає початковій клітинці A, та знайти довжину найкоротшого шляху з неї до цільової клітини B. Однак час виконання такого алгоритму буде занадто великим, адже пошук у ширину працює $O(V+E)$ часу, де V — кількість вершин, а E — кількість ребер у графі. У нашому випадку кількість вершин не перевищує NM, але кількість ребер може досягати $NM(N+M-2)/2$ (і дорівнює цьому числу, якщо зафарбовано всі NM клітин). Тож час виконання програми складе $O(NM(N+M))$. Це дасть 60% від загальної кількості балів.

Суттєво пришвидшити алгоритм можна, зауваживши таке: якщо на деякому кроці пошуку в ширину ми розглядаємо клітинку C, а раніше вже було розглянуто хоча б одну клітинку D, що стоїть у тому ж рядку, що й клітинка C , то немає сенсу розглядати ребра графа, які сполучають клітину C з іншими клітинками в її рядку. Справді, припустимо, що D — перша з клітинок у даному рядку, пройдена алгоритмом. Якщо деяке ребро (C, E) сполучає C з клітинкою E, яка міститься в тому ж рядку, що й C, то або $E = D$, або у графі наявне ребро (D, E) (бо D та E містяться в одному рядку). А отже, вершину E вже було додано в чергу або навіть пройдено раніше — і розгляд ребра (C, E) не дасть жодного результату. Отже, можна не розглядати багато «зайвих» ребер — достатньо пам'ятати клітини, у яких рядках алгоритм уже встиг пройти.

Звичайно, ті самі міркування справедливі не тільки для рядків, але й для стовпчиків таблиці. Тому в кожному рядку буде розглянуто щонайбільше $M-1$ ребро, а в кожному стовпчику — не більше ніж $N-1$ (ребра, що виходять з деякої однієї клітини даного рядка або стовпчика). Усього алгоритм розглядає $O(NM)$ ребер, а тому й час виконання дорівнюватиме $O(NM)$. Це дасть повний бал.

Задачу можна розв'язувати дещо інакше. Знов побудуємо граф, що відповідає таблиці у вхідному файлі, але тепер зробимо це іншим способом. Новий граф матиме $N+M$ вершин і не більше ніж NM ребер. Нехай N вершин $a_1, a_2, \ldots, a_N$ графа відповідають рядкам 1, 2, ..., N, а інші M вершин $b_1, b_2, ..., b_M$ відповідають стовпчикам 1, 2, ..., M таблиці. Ребро між вершинами a_i та b_j проведене тоді й лише тоді, коли на перетині i-го рядка та j-го стовпчика стоїть зафарбована клітинка. Інших ребер у графі немає.

Поставимо у відповідність ходу з клітинки (i, j_1) у клітинку (i, j_2) таблиці переміщення з вершини a_i у вершину bj_2 графа, а ходу з клітинки (i_1, j) у клітинку (i_2, j) — переміщення з вершини b_j у вершину a_{i2}. Оскільки в оптимальному (найкоротшому) маршруті між клітинками A і B не може бути двох послідовних ходів у межах одного рядка або двох послідовних ходів у межах одного стовпчика (інакше їх можна було б замінити на один еквівалентний хід), то оптимальному маршруту між A і B відповідає деякий шлях по ребрах графа. При цьому якщо $A=(i_1, j_1)$ а $B=(i_2, j_2)$, то шлях починається або у вершині a_{i_1}, або у вершині b_{j_1}, а закінчується ребром, що сполучає вершини a_{i_2} та b_{j_2} (або в напрямку від вершини a_{i_2} до b_{j_2}, або в протилежному). І навпаки: кожному такому шляху по ребрах графа відповідає маршрут тієї ж довжини між клітинками A і B.

Отже, задача зводиться до пошуку довжини найкоротшого шляху, що починається в одній з вершин a_{i_1} або b_{j_1}, а закінчується ребром, що сполучає вершини a_{i_2} та b_{j_2} графа. Якщо позначити через $d(x, y)$ довжину найкоротшого шляху між вершинами x та y, шукана величина дорівнює

$$\min\{d(a_{i_1}, a_{i_2}), d(a_{i_1}, b_{j_2}), d(b_{j_1}, a_{i_2}), d(b_{j_1}, b_{j_2})\} + 1.$$

А якщо між відповідними парами вершин шляхів не існує, то нема й маршруту між клітинками A і B.

Зауважимо, що відстані між парами вершин можна підрахувати з допомогою двох пошуків у ширину (один з вершини a_{i_1}, а інший — із b_{j_1}). Утім, для зна-

ходження відповіді можна обійтися й одним пошуком у ширину. Для цього додамо до графа фіктивну вершину c й проведемо з неї два ребра: у вершини a_{i_1} та b_{j_1}. Тоді шукане значення — це довжина найкоротшого шляху з вершини c в одну з вершин a_{i_2} або b_{j_2}, тобто $\min\{d(c, a_{i_2}), d(c, b_{j_2})\}$.

Отже, маємо альтернативний алгоритм розв'язання задачі з таким же порядком часу виконання $O(NM)$.

3. «Тетрис» (Сергій Нагін)

Тінкі-Вінкі грає у модулярний тетрис. Поле складається з N стовпчиків, у кожному з яких може міститися від нуля до трьох кубиків. Після того, як у стовпчику опиняється четвертий кубик, усі чотири кубики зникають. За один хід гравець може вибрати довільну кількість від 1 до N послідовних стовпчиків, на які впаде по одному кубику, як зображено на рисунку. Тінкі-Вінкі хоче, починаючи з наявної конфігурації кубиків на полі, якомога скоріше досягти певної цільової конфігурації.

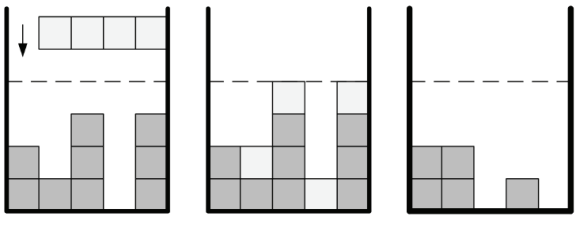

Рис. 3

Завдання. Напишіть програму **tetris**, яка за інформацією про кількість стовпчиків на полі, початкову та цільову конфігурації кубиків визначить найменшу кількість ходів, які має зробити Тінкі-Вінкі.

Вхідні дані. У першому рядку вхідного файлу **tetris.dat** міститься ціле число N ($1 \leqslant N \leqslant 1000$) — кількість стовпчиків на полі тетриса. У другому рядку записано N цілих чисел від 0 до 3, які задають початкову конфігурацію кубиків на полі. У третьому рядку записано N цілих чисел від 0 до 3, які задають кінцеву конфігурацію кубиків. Початкова та кінцева конфігурації не збігаються.

Вихідні дані. Єдиний рядок вихідного файлу **tetris.sol** повинен містити єдине ціле число — мінімальну можливу кількість ходів Тінкі-Вінкі для досягнення цільової конфігурації.

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 30 % балів: $N \leqslant 8$.
- 20 % балів: $N \leqslant 1000$ та правильна відповідь не перевищує 10.
- 25 % балів: $N \leqslant 100$.
- 25 % балів: немає додаткових обмежень.

Приклади вхідних та вихідних даних

tetris.dat	tetris.sol
5 2 1 3 0 3 2 2 0 1 0	1
4 0 1 2 3 3 2 1 0	5

Рекомендації щодо розв'язання

Позначимо початкову конфігурацію як A, а кінцеву — як B.

Розв'язок за $O(4^N \cdot N^3)$. Нехай кожній конфігурації кубиків відповідає вершина графа, а між конфігураціями, між якими можна зробити перехід за один хід, є орієнтоване ребро. Запустимо стандартний пошук у ширину від вершини, що відповідає конфігурації A, до вершини, що відповідає конфігурації B. Отримана найкоротша відстань і буде відповіддю.

Розв'язок за $O(N^3)$. Зрозуміло, що зміна порядку операцій не змінює результуючої конфігурації. Впорядкуємо ходи гравця — відрізки кубиків — за лівим краєм.

Будемо використовувати динамічне програмування: нехай dp(pref, opened) — мінімальна кількість операцій, яка потрібна, щоб прирівняти конфігурації в перших pref символах за умови, що на теперішньому кроці буде відкрито opened відрізків від деяких попередніх операцій. Перехід можна робити так:

- закінчимо деяку кількість попередніх операцій;
- почнемо деякі інші операції, ліві кінці яких будуть збігатися з теперішньою позицією;
- після перших двох пунктів числа в обох конфігураціях на теперішній позиції повинні збігатися.

Більш формально динаміку можна записати так (deleted — кількість закінчених операцій; new — кількість розпочатих операцій):

- dp(pref, opened) + new → dp(pref + 1, opened− deleted+new): deleted ⩽ opened; new ⩽ 3; (A[pref+1] + opened − deleted + new) mod 4 = B[pref+1].
- dp(0, 0) = 0.

Розв'язок за $O(N^2)$. Додамо оптимізацію до попереднього розв'язку — обмеження deleted ⩽ 3. Зрозуміло, що на жодному кроці не треба скасовувати більше ніж три операції, інакше їх не треба було б починати.

Неасимптотичні оптимізації. Замість операцій mod 4 (Pascal) та % 4 (C++) можна використовувати операції and 3 (Pascal) та & 3 (C++). Дана оптимізація може скоротити час виконання в 10 разів.

Також можна помітити, що ми або скасовуємо деякі операції, або створюємо нові, але не робимо цього одночасно. Скориставшись цим, можна зменшити час виконання в 4 рази.

Також можна перебирати тільки досяжні стани динаміки, це теж може скоротити час виконання в 4 рази.

4. «Камелот» (Роман Єдемський)

Король Артур вирішив зібрати лицарів задля термінової військової наради. На землях, якими правив Артур, розташовувалися оборонні фортеці, побудовані у формі кола, — такі вважалися найбільш неприступними. Певні фортеці були розташовані всередині інших, що забезпечувало їм іще більшу захищеність. Як тільки лицарі отриму-

ють наказ від Артура, вони вирушають у дорогу зі свого маєтку в супроводі охорони. Якщо шлях лицаря проходить ззовні фортеці всередину чи навпаки, лицар повинен заплатити данину за перетин брами. Кожна фортеця встановлює розмір данини за прохід однієї людини, тобто лицарю потрібно заплатити за себе та своїх охоронців.

Артур бажає вибрати таке місце проведення наради, щоб мінімізувати сумарні витрати лицарів, адже вони будуть відшкодовані з державної скарбниці. Місце проведення наради може розташовуватися будь-де на землях Артура, крім границь фортець. Крім того, Артур може зменшити свої витрати, скасувавши данину не більш ніж у K вибраних ним фортецях. На свій шлях з Камелота до місця наради Король нічого не витрачає, а всі лицарі завжди вибирають найдешевший маршрут.

Завдання. Напишіть програму **camelot,** яка за інформацією про карту земель Артура, розміри данини кожної з фортець, кількість охоронців у лицарів і кількість фортець, де данина може бути скасована за наказом Короля, знайде мінімальну кількість грошей, яку він має витратити, щоби провести нараду. Карту земель може бути подано як площину з колами, що задають фортеці, і точками, що задають маєтки лицарів.

Вхідні дані. Перший рядок вхідного файлу **camelot.dat** містить три цілих числа N, M і K ($2 \leqslant N \leqslant 35\,000$, $1 \leqslant M \leqslant 35\,000$, $0 \leqslant K \leqslant N$), де N — кількість фортець на землях Артура, M — кількість лицарів, викликаних на нараду, а K — кількість фортець, у яких Артур може скасувати данину. Наступні N рядків задають фортеці й містять по чотири цілих числа x, y, R, C ($-10^6 \leqslant x \leqslant 10^6$, $-10^6 \leqslant y \leqslant 10^6$, $1 \leqslant R \leqslant 2 \cdot 10^6$, $1 \leqslant C \leqslant 10^5$), де (x, y) — координати центра фортеці на мапі, R — радіус кола, що задає фортецю, а C — розмір данини з людини. Наступні M рядків задають інформацію про лицарів і містять по три цілих числа x, y, L ($-10^6 \leqslant x \leqslant 10^6$, $-10^6 \leqslant y \leqslant 10^6$, $1 \leqslant L \leqslant 10^5$), де (x, y) — координати маєтку лицаря на мапі, а L — кількість охоронців, що подорожують разом із лицарем, включаючи самого лицаря. Вхідні дані гарантують:

- жодні два кола, що задають фортеці, не мають спільних точок;
- жодні дві точки, що задають маєтки лицарів, не збігаються та не лежать на колах.

Вихідні дані. Єдиний рядок вихідного файлу **camelot.sol** повинен містити ціле число — мінімальну суму грошей, які Король Артур має витратити, щоб зібрати лицарів.

Оцінювання. Набір тестів складається з 6 блоків, для яких додатково виконуються такі умови:

- 10% балів: $N \leqslant 1000$, $M \leqslant 1000$ і немає фортець, що розташовані на території інших фортець;
- 10% балів: $N \leqslant 35\,000$, $M \leqslant 35\,000$ і немає фортець, що розташовані на території інших фортець;
- 10% балів: $N \leqslant 1000$, $M \leqslant 1000$, $K = 0$;
- 20% балів: $N \leqslant 35\,000$, $M \leqslant 35\,000$, $K = 0$;
- 15% балів: $N \leqslant 1000$, $M \leqslant 1000$;
- 35% балів: немає додаткових обмежень.

Приклад вхідних та вихідних даних

camelot.dat	camelot.sol
4 9 1 6 10 2 1 5 4 2 1 10 7 1 200 7 7 7 1 5 3 10 6 10 1 7 10 1 10 7 1 10 10 1 9 11 1 9 12 1 13 1 1 14 1 1	12

Пояснення. Щоб досягти оптимальності витрат, Королю Артуру необхідно скасувати данину у фортеці 3 і зібрати нараду у фортеці 2.

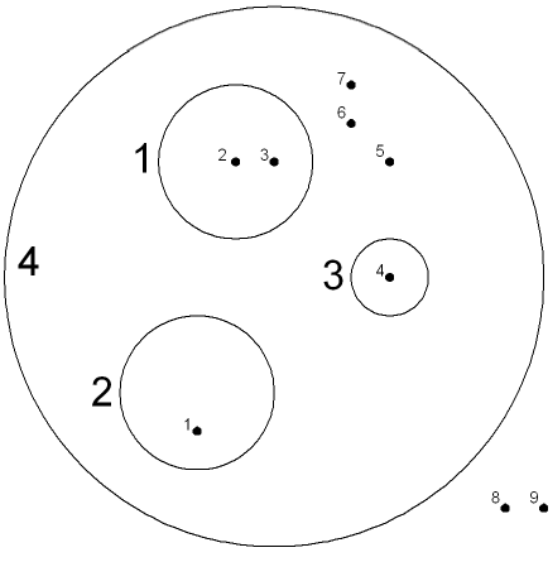

Рис. 4

Рекомендації щодо розв'язання

Зазначимо, що кола, які не перетинаються, утворюють дерево, причому кола відповідають ребрам, а області зв'язності — вершинам. Тому задачу можна переформулювати так: у кожній вершині дерева перебуває деяка кількість людей, що можуть переходити з вершини до вершини по ребрах, сплачуючи деяку суму грошей за цю операцію. Дозволено зробити ціну переходу нульовою не більш ніж для K ребер. Треба зібрати всіх людей в одній вершині, витративши на це якомога менше грошей.

Отже, задачу можна розбити на дві підзадачі: геометричну й оптимізаційну. *Геометрична підзадача* полягає у побудові дерева і визначенні початкової кількості людей у кожній вершині.

Геометрична підзадача: наївний розв'язок за $O(N^2+NM)$. Для кожного кола знайдемо коло мінімального радіуса, що містить дане, і позначимо область зв'язності, що лежить між цими двома колами, як предка у дереві до вершини, що відповідає області зв'язності внутрішнього кола. У випадку, якщо немає кола, що містить задане, вважатимемо його предком деякий глобальний корінь нашого

дерева — вершину, що відповідає області зв'язності, яка лежить поза межами всіх кіл.

Геометрична підзадача: метод скануючої прямої (sweep Hne) за $O((N+M)\log(N+M))$. Для полегшення подальшого викладу будемо вважати, що на додаток до всіх кіл, що задані у вхідному файлі, ми маємо деяке коло із центром у точці (0, 0) і радіусом таким, що всі інші точки та кола лежать усередині нього. Під час «сканування» ми не будемо розглядати події початку і закінчення цього кола, натомість дуги цього кола будуть постійно міститися у допоміжних структурах даних.

Будемо «сканувати» прямою площину зліва направо, підтримуючи у деякому бінарному збалансованому дереві верхні і нижні дуги кіл, що перетинають пряму в її поточному розташуванні. Ці дуги будемо впорядковувати у дереві за ординатою точки перетину з прямою. Кожну з можливих подій будемо опрацьовувати так.

Якщо скануюча пряма **натрапила на нове коло,** ми знаходимо у дереві дуг дугу, найближчу (або зверху, або знизу — неважливо) до точки дотику прямої і кола, й перевіряємо, чи коло, якому належить ця дуга, є таким, що містить нове коло. У випадку, якщо це так, позначаємо його «батьком» (предком) нового кола; якщо ні, то, як нескладно зрозуміти, батьком нового кола є батько кола, якому належала знайдена дуга. Після того як вершина — батько нового кола визначена, додамо дві його дуги до дерева дуг.

Якщо скануюча пряма **виходить за межі кола,** видалимо з дерева дуг вершини, що відповідають дугам цього кола.

Якщо скануюча пряма **натрапила на точку,** знаходимо коло мінімального радіуса, яке містить точку, у спосіб, аналогічний до наведеного вище методу знаходження батька нового кола: точку можна вважати колом радіуса 0.

Зауваження щодо реалізації дерева дуг:

- у мові C++ можна використовувати стандартний контейнер **set** із динамічним компаратором, що порівнює дуги за значенням y-координат точок перетину дуг зі скануючою прямою в її поточному розташуванні. При цьому є нюанс: дуги, що належать одному колу, треба порівнювати безпосередньо (нижня під верхньою), не порівнюючи y-координати цих дуг. Це пов'язано з тим, що в перший і в останній моменти, коли пряма перетинатиме коло, вона буде перетинати обидві дуги в одній і тій самій точці.

Існує також розв'язок, що не використовує динамічний компаратор: він порівнює дуги за відносним розташуванням кіл, до яких ці дуги належать. Такий розв'язок є більш ніж удвічі швидшим за розв'язок із динамічним компаратором.

До *оптимізаційної підзадачі* також є кілька підходів.

Оптимізаційна підзадача: випадок $K=0$. У цьому випадку в Артура немає можливості скасовувати плату за перехід по ребрах. Тоді єдине, що залежить від Артура, вибір вершини збору. Маємо наївний розв'язок з асимптотикою $O(N^2)$: перебрати варіанти вершини збору і для кожного варіанта за допомогою пошуку в глибину знайти ціну збору в даній вершині. Зробити це можна, рахуючи дві допоміжні величини: кількість людей у піддереві й сумарну ціну збору всіх людей з піддерева у корені цього піддерева. Такий розв'язок оптимізується до лінійного за допомогою підтримання під час пошуку в глибину описаних величин не тільки у піддеревах, а також і в наддереві (тобто в дереві без піддерева).

Оптимізаційна підзадача: розв'язання з допомогою структур даних. Нехай зафіксовано деяку вершину збору. Зорієнтуємо всі ребра так, як по них будуть іти люди до вершини збору. З цих міркувань для кожного ребра можна оцінити те, яку знижку отримає Артур, якщо зробить ціну проходу по даному ребру нульовою. Ця величина дорівнює кількості людей у відповідному піддереві, помноженій на ціну переходу. Так отримаємо розв'язок за $O(N^2 \log N)$: перебираємо можливі варіанти вершини збору й за допомогою пошуку в глибину вираховуємо знижки для кожного ребра; далі вибираємо K ребер, для яких знижки найбільші.

Для оптимізації цього розв'язку до $O(N \log N)$ подивимось, як змінюється множина знижок при перенесенні вершини збору до сусідньої вершини: змінюється лише знижка на ребро, через яке це перенесення здійснювалося. Отже, необхідно запровадити деяку структуру даних, що підтримує набір чисел (разом із дублікатами) і може швидко опрацьовувати запити двох типів:

- замінити деякий елемент x на y;
- знайти K максимальних елементів.

Зауважимо, що перша операція еквівалентна послідовності операцій: додати елемент y до набору, видалити елемент x із набору. Цю структуру даних можна промоделювати за допомогою контейнера **map** зі стандартних бібліотек C++, підтримуючи поточне значення K-го найбільшого елемента та кількість елементів, строго менших за K-й найбільший елемент. Альтернативою до цієї структури даних є дерево відрізків зі стисканням координат або розріджена його модифікація.

Оптимізаційна підзадача: розв'язання з допомогою жадібного алгоритму. Розглянемо деяке ребро (U, V) дерева. Нехай його ціна — cost. Позначимо кількість людей у піддереві вершини U через L, а у піддереві вершини V — через R. Доведемо, що до ціни збору це ребро додає $\min\{L, R\} \cdot$ cost грошей при оптимальному виборі вершини збору. Справді, нехай, наприклад, $L < R$, а вибрана вершина збору X міститься у піддереві L. Хай зібрати всіх людей з L у X коштує A, а зібрати всіх з R у V коштує B. Позначимо ціну проходу однієї людини по шляху з V у X через W. Тоді ціна збору наради у вершині X складає $A+B+W \cdot R$, а у вершині V — $A+B+W \cdot$ L, що, очевидно, менше. Отже, X не є оптимальним вибором вершини збору.

Зауважимо, що оцінка $\min\{L, R\} \cdot$ cost не залежить від того, чи ми скасовували плату за перехід через інші ребра. Тому достатньо вирахувати дану величину для всіх ребер та вилучити K найбільших значень — сума решти і є відповіддю.