Юрій Володимирович Знов'як,
студент Київського національного університету імені Тараса Шевченка,

Дмитро Олександрович Кордубан,
студент Національного технічного університету України «Київський політехнічний інститут»,

Данило ПетровичМисак,
студент Київського національного університету імені Тараса Шевченка

Михайло Валерійович Рибак, приватний підприємець

Олександр Борисович Рудик,
доцент Київського міського педагогічного університету ім. Б.Д.Грінченка

Володимир Леонідович Ткачук
аспірант Міжнародного науково-навчального центру інформаційних технологій та систем

Київська олімпіада з інформатики у 2007-2008 навчальному році

Стаття містить умови завдань III (міського) етапу олімпіади з основ інформатики й обчислювальної техніки у місті Києві у 2008 році та авторські розв'язання цих завдань. Публікацію адресовано учням класів з поглибленим вивченням математики, учасникам олімпіад з інформатики, студентам математичних спеціальностей, учителям і викладачам вищих навчальних закладів.

Проведенню III етапу у 2007 році у місті Києві передувало проведення II (районного) етапу таким чином.Завдання для II етапу Всеукраїнської учнівської олімпіади з інформатики у місті Києві у 2007 році готував КМПУ ім. Б.Д.Грінченка. Ці завдання можна було повністю виконати, використовуючи інтегровані середовища програмування Turbo Pascal 7.0 чи Turbo C++ 3.0 або потужніші.

1. Орієнтовні завдання II етапу олімпіади з інформатики опубліковано на сайті kievoi.narod.ru в день проведення II етапу олімпіади з 9^{45} до 9^{50} і містило умови трьох задач:
   - перші дві задачі — це випадковим чином вибрані задачі відбірково-тренувальних зборів команди міста Києва, розташовані на вказаному сайті разом з тестовими файлами. У 2007 році це було завдання № 8 відбірково-тренувальних зборів (теорія графів + теорія цілих чисел + "довга арифметика"). Ці алгоритмічно змістовні задачі задумано як кваліфікаційні

завдання щодо реалізації відомих алгоритмів;

- третю задачу задумано як задачу з нескладною моделлю і простим для реалізації алгоритмом (визначення кінцевого стану термостату, що у початковий момент містив кригу та розплавлений свинець).

2. Порядок проведення ІІ етапу олімпіади майже повністю збігався з порядком проведення ІІІ етапу олімпіади. Журі та організаційні комітети ІІ етапу олімпіади повинні були забезпечити неможливість доступу учасників олімпіади до ресурсів глобальної мережі, матеріалів відбірково-тренувальних зборів команди міста Києва та робочих каталогів інших учасників з моменту розташування їх за робочими місцями.
3. Архів тестових файлів до орієнтовних завдань ІІ етапу олімпіади з інформатики разом з системою тестування було вперше опубліковано на вказаному сайті в день проведення ІІ етапу олімпіади з 15^{30} до 15^{35}. Цей архів містив прокоментоване авторське розв'язання третьої задачі, за яким математична модель й алгоритм легко відновлюються.

Принагідно зазначимо, що учасники ІІІ етапу олімпіади після проведення кожного туру могли отримати електронні копії умов завдань, ідеї розв'язання та тестові файли, використовуючи вказаний сайт. Учасники олімпіади, таким чином, мали можливість ґрунтовно підготуватися до апеляції результатів перевірки найкращим чином.

1. Умови завдань

1. Сновида-2 (автор — Володимир Ткачук)

Максимальна оцінка: 100 балів
Обмеження за часом: 1 сек.
Обмеження за пам'яттю: 32 МБ
Вхідний файл: sleep2.in
Вихідний файл: sleep2.out
Програма: sleep2.pas/.c/.cpp

Грабіжникам стало відомо про звичку президента Першого національного Банку майора Томаса Б. Кінгмена кожну ніч перекладати вміст сейфів, у яких клієнти банку зберігають свої коштовності. Шляхом погроз і підкупу співробітників банку злочинці з'ясували як змінилось розташування коштовностей після двох ночей. На жаль для грабіжників, ця інформація не дозволяє однозначно встановити алгоритм, за яким майор перекладає вміст сейфів.

Завдання

Створіть програму sleep2.* , яка обчислює кількість можливих алгоритмів перекладання, що за дві ночі приводять до відомого злочинцям результату, якщо алгоритм перекладання щоночі однаковий.

Вхідні дані

В першому рядку файлу sleep2.in записано єдине натуральне число N — кількість сейфів у банку ($N \le 100$). У другому рядку через прогалину записано N

натуральних чисел a_1, a_2, ... , a_N, що описують вміст сейфів після другої ночі. Кожне таке число належить діапазону [1..*N*] і зустрічається в рядку лише один раз. Тут a_i — номер сейфа в який коштовності потрапили з *i*-го сейфа за дві ночі перекладань.

Вихідні дані

Єдиний рядок вихідного файлу sleep2.out має містити одне ціле число — результат обчислень.

Приклади

№	sleep2.in	sleep2.out
1	4 3 4 2 1	0
2	4 2 1 4 3	2
3	5 1 2 3 4 5	26

2. Карти (автор — Юрій Знов'як)

Максимальна оцінка: 100 балів
Обмеження за часом: 1 сек.
Обмеження за пам'яттю: 32 МБ
Вхідний файл: cards.in
Вихідний файл: cards.out
Програма: cards.pas/.c/.cpp

Колода складається з парної кількості карт. Карти занумеровано послідовними натуральними числами від 1 до 2*N* (для певного натурального *N*) включно. З колодою роблять таке:

1. Спочатку колоду ділять на дві рівні частини. В першу частину потрапляють карти, що стояли на непарних місцях, в другу — ті карти, що були на парних місцях: (1, 2, ... , 2*N*) → (1, 3, 5, ... , 2*N* – 1) + (2, 4, ... , 2*N*).
2. З обох утворених частин викидають карту, що розташована у відповідній частині на місці 1 + [*aN*/*b*] при $0 \le a < b \le 1000$. Наприклад, якщо $a = 0$, $b = 1$, то буде викинуто першу карту:
   (1, 3, 5, ... , 2*N* – 1) → (3, 5, 7, ... , 2*N* – 1);
   (2, 4, 6, ... , 2*N*) → (4, 6, 8, ... , 2*N*).
3. Другу частину кладуть поверх першої:
   (3, 5, 7, ... , 2*N* – 1) + (4, 6, 8, ... , 2*N*) → (3, 5, 7, ... , 2*N* – 1, 4, 6, 8, ... , 2*N*).

Дії 1–3 повторюють доти, поки в колоді не залишиться лише 2 карти. Наприклад, якщо $a = 1$, $b = 2$, а початкова кількість карт — 5, маємо:
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) → (1, 3, 5, 7, 9) + (2, 4, 6, 8, 10) →
(1, 3, 7, 9) + (2, 4, 8, 10) → (1, 3, 7, 9, 2, 4, 8, 10) → (1, 7, 2, 8) + (3, 9, 4, 10) →
(1, 7, 8) + (3, 9, 10) → (1, 7, 8, 3, 9, 10) → (1, 8, 9) + (7, 3, 10) → (1, 9) + (7, 10) →
(1, 9, 7, 10) → (1, 7) + (9, 10) → (1) + (9) → (1, 9),
бо 1 + [5•1/2] = 3, 1 + [4•1/2] = 3, 1 + [3•1/2] = 2, 1 +[2•1/2] = 2.

Завдання

Визначте кінцевий стан колоди.

Вхідні дані

Вхідний файл містить 3 невід'ємні цілі числа *N, a, b*. $2 \le N \le 500\ 000$.

Вихідні дані

Вихідний файл має містити два натуральних числа: номери карт, що залишаться. Номер нижньої карти потрібно вказати першим.

Приклад

cards.in

5 1 2

cards.out

1 9

3. Гра (автор — Михайло Рибак)

Максимальна оцінка: 100 балів
Обмеження за часом: 1 сек.
Обмеження за пам'яттю: 32 МБ
Вхідний файл: game.in
Вихідний файл: game.out
Програма: game.pas/.c/.cpp

Полем гри є декартова площина із двома групами занумерованих точок. Точки першої групи мають координати (0; *i*), де *i* — номер точки в групі (нумерація починається з 0). Точки другої групи мають координати (1; *j*), де *j* — номер точки в групі (нумерація також починається з 0). Кожну точку пофарбовано у білий або у чорний колір. Гравці ходять по черзі. За один хід гравець сполучає відрізком деяку точку *A* з першої групи з деякою точкою *B* з другої групи. При цьому справджуються такі умови:

- точки *A* і *B* — одного кольору;
- відрізок AB не має спільних точок з довільним уже проведеним відрізком.

Гравець, що не може зробити наступний хід, програє.

Завдання

Для заданого ігрового поля вкажіть, чи може перший гравець забезпечити собі перемогу. Якщо відповідь ствердна, вкажіть усі переможні для нього перші ходи за умови оптимальності гри його суперника.

Вхідні дані

Перший рядок містить число M ($0 \le M \le 20$) — кількість точок у першій групі.
Другий рядок містить число N ($0 \le N \le 20$) — кількість точок у другій групі.
Третій рядок містить M чисел, що задають розфарбування точок першої групи (0 — чорний колір, 1 — білий колір).
Четвертий рядок містить N чисел, що задають розфарбування точок другої групи

таким самим чином.

Вихідні дані

Якщо виграє другий гравець, єдиний рядок вихідного файлу містить слово LOSE. Якщо виграє перший гравець, перший рядок вихідного файлу містить слово WIN, а наступні рядки перераховують виграшні перші ходи. Кожний рядок містить один з можливих ходів у такому форматі:

I J

Тут *I* — номер точки з першої групи, *J* — номер точки з другої групи. Кожен виграшний хід повинен зустрічатися один та лише один раз. Пари (*I*; *J*) потрібно подати в лексикографічному порядку.

Приклади

№	game.in	game.out
1	2 2 0 1 1 0	WIN 0 1 1 0
2	2 2 0 1 0 1	LOSE
3	2 3 0 1 1 0 1	WIN 1 0

4. Школа (автор — Данило Мисак)

Максимальна оцінка: 100 балів
Обмеження за часом: 1 сек.
Обмеження за пам'яттю: 32 МБ
Вхідний файл: school.in
Вихідний файл: school.out
Програма: school.pas/.c/.cpp

Петро П'яточкін учиться у школі, а тому має кожен день до неї ходити. Він живе у місті Ідеальне, що відоме у тому числі й своєю розвинутою тролейбусною мережею (щоправда, іншого транспорту в Ідеальному немає). Зокрема, відомо, що

всі тролейбусні маршрути у місті мають *n*-цифрові номери, і кожне *n*-цифрове число є номером деякого тролейбусного маршруту. Крім того, на двох маршрутах є спільна зупинка тоді й лише тоді, коли з номера одного маршруту можна шляхом переставляння цифр або зменшення однієї з ненульових цифр на одиницю отримати номер іншого. Для чотирицифрових номерів можна подати такі приклади.

Маршрути	**Пересадка**	**Причина**
1233— 3231	можлива	цифри першого номера можна переставити так, щоб отримати другий
5871— 5861	можлива	другий номер можна отримати з першого, зменшивши третю цифру на один
5861— 5871	можлива	перший номер можна отримати з другого, зменшивши третю цифру на один
9075— (0)957	неможлива	957 — не чотирицифрове число, такого маршруту нема
5681— 5782	неможлива	щоб отримати перший номер, необхідно зменшити одразу дві цифри другого
1234— 3321	неможлива	щоб отримати з першого номера другий, треба і змінити порядок цифр, і зменшити одну з цифр на один

Відомі номери маршрутів тролейбусів, що зупиняються біля дому Петрика, а також номери маршрутів, що проходять повз його школу.

Завдання

Допоможіть Петрикові їздити до школи, роблячи якомога менше пересадок.

Вхідні дані

У першому рядку вхідного файлу вказані три натуральні числа: *n* — кількість цифр у номерах тролейбусних маршрутів Ідеального; *m* — кількість різних маршрутів, які проходять повз будинок Петрика; *k* — кількість маршрутів, що проходять повз школу, в якій Петрик учиться. У другому рядку перераховано *m*, а у третьому — *k* різних чисел, що є номерами відповідних маршрутів. Відомо, що $2 \le n \le 1000$ та $1 \le m, k \le 100$.

Вихідні дані

Перший рядок вихідного файлу має містити єдине число *x* — найменшу можливу кількість пересадок, що доведеться зробити Петрику на шляху до школи.

Другий рядок має містити перелік із $x + 1$ числа, де у порядку пересадок зазначено номери маршрутів, якими Петрик П’яточкін їздитиме до школи. Якщо є кілька варіантів пересадок, необхідно вказати лише один із них.

Вхідні дані гарантують існування вихідних даних обсягом до 2Мб.

Приклад

school.in	**school.out**
3 1 2	6

560 879 138	560 561 156 157 147 137 138

5. Коло (автор — Андрій Гриненко)

Максимальна оцінка: 100 балів
Обмеження за часом: 6 сек.
Обмеження за пам'яттю: 32 МБ
Вхідний файл: circle.in
Вихідний файл: circle.out
Програма: circle.pas/.c/.cpp

На площині задано N різних точок, жодні три з яких не лежать на одній прямій.

Завдання

Знайдіть найменший радіус кола, всередині якого і на якому разом розташовано не менше ніж K даних точок.

Вхідні дані

Перший рядок вхідного файлу містить два натуральні числа N, K при $1 \le K \le N \le 200$.

Наступні N рядків містять по два дійсних числа, які не перевищують 1000, — координати даних точок.

Вихідні дані

Єдиний рядок вихідного файлу має містити шуканий радіус, округлений до трьох знаків після крапки, яку використовують замість десяткової коми.

Приклад

circle.in

3 3

0.0 1.0

-1.0 -1.0

1.0 -1.0

circle.out

1.250

6. Канікули (автор — Дмитро Кордубан)

Максимальна оцінка: 100 балів
Обмеження за часом: 3 сек.
Обмеження за пам'яттю: 32 МБ
Вхідний файл: vacation.in
Вихідний файл: vacation.out

Програма: vacation.pas/.c/.cpp

Петрик успішно склав зимову сесію. Тепер він має вдосталь вільного часу, і хоче перетворити цей час на гроші, працюючи фотографом на сільських весіллях. Точніше, Петрик хоче виїхати з Києва, відвідати декілька сіл і повернутися додому. У кожному новому селі він заробить сталу суму грошей T — вартість зйомки весілля. Але лише при першому візиті до цього села — подальші візити прибутку не приносять. Відома вартість проїзду між населеними пунктами. Вважаємо, що початковий капітал Петрика достатній для того, щоб дістатись до будь-якого села будь-яким простим шляхом.

Завдання

Визначте найменше ціле значення T, за якого існує шлях селами з початком і кінцем у Києві, що принесе Петрику додатний прибуток. Існування такого шляху гарантовано вхідними даними при певному T.

Вхідні дані

Перший рядок вхідних даних містить натуральне число N — кількість сіл ($N \le 16$). Села занумеровано натуральними числами від 1 до N. Київ має номер 0. Кожний з наступних $N + 1$ рядка містять по $N + 1$ цілому числу — x_{ij}, де j — номер числа у i-му рядку у відповідній прямокутній таблиці чисел. У цій таблиці нумерацію рядків і нумерація чисел у кожному з рядків починають з нуля, $0 \le i, j \le N$. Число x_{ij} дорівнює вартості проїзду між населеними пунктами i та j. Ці вартості не перевищують 10 000, $x_{ij} = x_{ji}$, $x_{ii} = 0$ при всіх i та j. Від'ємна величина x_{ij} вказує на відсутність прямого шляху між пунктами i та j.

Вихідні дані

Єдиний рядок вихідного файлу має містити одне ціле число — найменше ціле значення T, за якого існує прибутковий цикл.

Приклади

vacation.in	**vacation.out**
2 0 2 2 2 0 1 2 1 0	3
2 0 2 2 2 0 -1 2 -1 0	5

2. Ідеї розв'язання

1. Сновида–2.

У задачі вимагається знайти кількість коренів з відомої перестановки α, тобто, кількість таких перестановок β, що

$$\beta(\beta(i)) = \alpha(i), i = \overline{1,N}$$

Розглянемо циклічне подання перестановки β:

$$\beta = (b_1^1, b_2^1, \dots, b_{N_1}^1)(b_1^2, b_2^2, \dots, b_{N_2}^2) \dots (b_1^k, b_2^k, \dots, b_{N_k}^k),$$

де в окремих дужках записані цикли перестановки (всього k циклів):

$$\begin{matrix} \beta(b_i^j) = b_{i+1}^j, & 1 \le i < N_j \\ \beta(b_{N_j}^j) = b_1^j \end{matrix}, \quad j = \overline{1,k}$$

Розглянемо, що станеться з циклом при подвійному застосуванні перестановки:

1. Якщо цикл має непарну довжину (кількість елементів), то він перетвориться на цикл такої самої довжини з тими самими елементами:

$$(b_1, b_2, \dots, b_N) \xrightarrow{\beta^2} (b_1, b_3, b_5, \dots, b_{N-2}, b_N, b_2, b_4, \dots, b_{N-1}).$$

2. Якщо цикл має парну довжину, то квадрат перестановки міститиме два цикли однакової довжини:

$$(b_1, b_2, \dots, b_{N-1}, b_N) \xrightarrow{\beta^2} (b_1, b_3, \dots, b_{N-3}, b_{N-1})(b_2, b_4, \dots, b_{N-2}, b_N).$$

Отже, при подвійному застосуванні перестановки, кожен її цикл непарної довжини перетворюється на цикл такої самої довжини, а кожен цикл парної довжини розпадається на два цикли однакової довжини. Звідси випливає критерій існування кореня у перестановки: якщо перестановка має цикли однакової парної довжини $2t$, то кількість таких циклів має бути парною.

Алгоритм побудови кореня з перестановки такий:

1. Всі цикли парної довжини розбити на пари однакової довжини й об'єднати кожну пару в один цикл.
2. Деякі з циклів непарної довжини розбити на пари однакової довжини і об'єднати кожну пару в один цикл. В інших циклах лише змінити порядок елементів.

Для обчислення кількості коренів, необхідно врахувати такі моменти:

1. Якщо ми маємо N циклів однакової парної довжини, то розбити їх на пари можна $(N-1)(N-3) \cdot \dots \cdot 3 \cdot 1 = (N-1)!!$ способами.
2. Якщо ми маємо N циклів однакової непарної довжини, то розбити *деякі* з них на пари можна такою кількістю способів:

$$\sum_{i=1}^{[N/2]} C_N^{2 \cdot i} \cdot (2i-1)!! + 1$$

3. Об'єднати два цикли однакової довжини L можна L різними способами.

Алгоритм розв'язання задачі такий:

1. Знайти всі цикли перестановки і кількість циклів кожної довжини.
2. Якщо для деякої парної довжини кількість циклів непарна, відповідь: 0.

3. Для кожної довжини t, для якої кількість циклів $N_t > 0$ обчислити таке число:

$$A_t = \begin{cases} t^{\frac{N_t}{2}}(N_t-1)!!, & t = 2k \\ \sum_{i=1}^{[N_t/2]} C_{N_t}^{2i}\, t^i (2i-1)!! + 1, & t = 2k+1 \end{cases}$$

4. Відповіддю є добуток всіх тих A_t, для яких $N_t > 0$.

Наприклад, для перестановки (1 2 4 3 6 5):

- перестановка складається з 4-х циклів: (1)(2)(3,4)(5,6). Є два цикли довжини 1, і два цикли довжини 2;
- циклів однакової парної довжини парна кількість, тому корінь існує;
- $A_1 = C_2^2 1^1 (2-1)!! + 1 = 1 + 1 = 2, A_2 = 2^1 (2-1)!! = 2$;
- $A_1 \cdot A_2 = 2 \cdot 2 = 4$;
- дана перестановка має 4 корені: (1 2 5 6 4 3), (1 2 6 5 3 4), (2 1 5 6 4 3) і (2 1 6 5 3 4).

2. Карти

Давайте з'ясуємо, де знаходилась карта, що після виконання однієї ітерації опинилась на місці p. Якщо $p \le N - 1$, то це число було на непарному місці, а якщо $p \ge N$, то це число було на парному місці. Справді, дія 1 розбиває колоду на «парні» та «непарні» карти, а дія 3 (склеювання) «парні» карти ставить попереду «непарних». Зауважимо, що на місце p стане карта, яка до виконання дій 1–3 була на місці:

- $2p - 1$ при $p < 1 + [Na / b]$;
- $2p + 1$ при $1 + [Na / b] \le p < N$;
- $2(p - (N - 1))$ при $N \le p < N + [Na / b]$;
- $2(p - (N - 1)) + 2$ при $N + [Na / b] \le p < 2N - 1$.

У розв'язанні реалізуємо функцію *prev*(p, N), що повертатиме номер місця, де знаходилась карта, що після виконання ітерації опинилася на місці p.

Для того, щоб визначити кінцевий стан колоди, будемо відстежувати розташування останніх двох карт „з кінця":

- при $N = 1$ маємо (1, 2);
- при $N = 2$ маємо (prev(1, 2), prev(2, 2));
- при $N = 3$ маємо (prev(prev(1, 2), 3), prev(prev(2, 2), 3)) і т.і. .

3. Гра

Елементарною позицією гри, яку позначатимемо четвіркою невід'ємних цілих чисел (i_0 ; i_1 ; j_0 ; j_1) при справдженні нерівностей

$$0 \le i_0 \le i_1 < M, \quad (1)$$
$$0 \le j_0 \le j_1 < N, \quad (2)$$

назвемо фрагмент з початкового поля, отриманого вилученням з першої групи точок усіх, окрім точок з номерами i_0 , $i_0 + 1$, … , i_1 та вилученням з другої групи точок усіх, окрім точок із номерами j_0 , $j_0 + 1$, … , j_1 . Інакше кажучи, елементарна позиція (i_0 ; i_1 ; j_0 ; j_1) є множиною точок двох типів:

- або (0; *i*) при справдженні нерівностей (1);
- або (1; *j*) при справдженні нерівностей (2).

Порожнє поле також вважатимемо элементарною позицією.

Довільним дозволеним ходом ми переходимо від такої *елементарної* позиції або до іншої елементарної позиції, або до комбінації (об'єднання) двох незалежних елементарних позицій. Незалежних у тому розумінні, що кожним наступним ходом можна з'єднати або пару точок з першої позиції, або пару точок з другої, але не можна з'єднати точку з першої позиції із точкою з другої (адже їх розділяє відрізок, що його щойно провели). Саме тому можемо безпосередньо застосувати метод Шпраґе-Ґранді:

1) *індекс позиції* (так зване *число Шпраґе-Ґранді*) дорівнює найменшому з індексів позицій, які *неможливо* отримати ходом з даної позиції;
2) індекс комбінації двох незалежних позицій дорівнює побітовій сумі (див. операцію xor у мові Паскаль) індексів цих позицій.

Детальний виклад теорії див. у публікації: О.Рудик. Вибрані питання дискретної математики й теорії ігор. Функціонал Шпраґе — Ґранді // Інформатика та інформаційні технології у навчальних закладах, 2008, (готується до публікації).

Переберемо всі можливі позиції (i_0; i_1; j_0; j_1) у лексикографічному порядку, і для кожної такої позиції обчислимо її індекс, використовуючи попередньо підраховані індекси.

Після цього ми для кожного можливого першого ходу першого гравця обчислюємо індекс позиції, що утворюється (за наведеним вище означенням). Виграшними є ті перші ходи першого гравця, які ведуть до позиції з індексом 0.

Наприклад, нехай $M = 4$, $N = 10$, і нас цікавить, чи є виграшним перший хід 2 5 (за умови, що точка 2 першої групи та точка 5 другої мають однаковий колір). Нехай id[][][][] — масив обрахованих індексів позицій. Тоді індекс позиції, що утворюється після першого ходу 2 5, дорівнює id[0][1][0][4] xor id[3][3][6][9] (відповідно до другого пункту означення індексу). Рівність цього числа нулю означає виграшність ходу 2 5, відмінність цього числа від нуля — програшність.

50% балів можна набрати без використання вказаного методу, побудувавши в оперативній пам'яті граф гри та провівши аналіз його позицій «з кінця гри» згідно з означенням виграшних і програшних позицій.

4. Школа

Назвемо операцією кожну з таких двох дій: зміну порядку цифр числа (таку, що після неї на першому місці не стоїть нуль) та зменшення або збільшення однієї довільної цифри числа на один (при цьому нуль не може бути зменшено, а 9 — збільшено).

Спочатку визначимо, як отримати з *n*-цифрового числа *a* *n*-цифрове число *b* за найменшу можливу кількість операцій. Така кількість є невід'ємною, а отже

обмеженою знизу. Існує щонайменше одна послідовність операцій, що переводить a у b (наприклад, послідовні $|a_i - b_i|$-кратні зміни i-ї цифри a_i числа a при кожному $1 \le i \le n$). Тому існує й така послідовність операцій, що має найменшу можливу довжину. Ця послідовність може мати щонайбільше одну операцію, яка змінює порядок цифр. Справді, нехай після всіх операцій цифра, що стояла на місці i, стане на місце $j(i)$ (при цьому, можливо, буде збільшена чи зменшена на певну кількість одиниць). Тоді загальна кількість операцій дорівнює $\sum_{i=1}^{n} \left| a_i - b_{j(i)} \right| + p$, де p — кількість операцій, що змінюють порядок цифр. Тоді значення p маємо покласти рівним 1, бо порахована кількість залежить лише від кінцевого розташування цифр, а розташувати цифри відповідним чином ми можемо за допомогою єдиної операції переставляння. Останнє твердження не є очевидним, враховуючи те, що в процесі здійснення операцій на першому місці у числі не може стояти нуль. Припустимо, що ми здійснили операцію, що переставляє цифри потрібним чином уже на першому ході. Якщо після цього на першому місці опиниться відмінна від нуля цифра, то на нуль вона в подальшому не перетвориться, бо у кінцевому числі (b) на першому місці також стоїть ненульова цифра. Якщо ж на першому місці все ж виявиться нуль, то замість того, щоб переставляти цифри першою операцією, збільшимо спочатку відповідний нуль у числі a на один, а вже після цього змінимо порядок цифр. Таким чином ми уникаємо проблем із нулем у старшому розряді номера маршруту. Тепер залишилось знайти таку підстановку $j(i)$, що вираз $\sum_{i=1}^{n} \left| a_i - b_{j(i)} \right|$ набуває найменшого можливого значення, а іншими словами — таку взаємно однозначну відповідність цифр чисел a та b, що загальна кількість операцій з перетворення однієї цифри на іншу у всіх парах стає мінімальною.

Припустимо, що цифри числа a впорядковано за неспаданням. Доведемо від супротивного, що у цьому випадку впорядкування цифр числа b за неспаданням забезпечить потрібний мінімум (якщо кожній цифрі числа a ставити у відповідність цифру b, що стоїть на місці з тим самим номером).

Оберемо таке розташування цифр числа b, яке серед тих, що забезпечують мінімум операцій, має найменшу можливу кількість порушень порядку (тобто таких пар сусідніх цифр, що більша стоїть лівіше від меншої). Це можна зробити, бо число всіх можливих розташувань скінченне. Якщо утвориться не впорядкований набір цифр, то в розташуванні існуватиме принаймні одне порушення порядку. Обмінявши відповідні цьому порушенню дві цифри місцями, ми по-перше залишимо властивість мінімальності, а по-друге зменшимо кількість порушень порядку принаймні на один: «початкове» порушення зникне, а нові з'явитися не можуть. Отримана суперечність свідчить про хибність припущення щодо немонотонності розташування цифр числа b.

Не можна забувати, що ми можемо зовсім не використовувати операції переставляння цифр. Якщо обмежитися лише операціями збільшення/зменшення цифр, то загальна їх кількість, необхідна для перетворення числа a на число b,

дорівнює $\sum_{i=1}^{n}|a_i - b_i|$. Тут проблема із нулем у першому розряді відсутня. Обидві знайдені кількості (з використанням і без використання операції зміни порядку цифр) потрібно порівняти. Правильна відповідь буде збігатися із тією кількістю, що виявиться не більшою за іншу.

Для повного ж розв'язання задачі достатньо перебрати всі пари номерів маршрутів, на тролейбуси за якими Петрик може сісти біля дому, і на яких може приїхати до школи, для кожної пари визначити мінімальну кількість операцій, що перетворюють один номер пари на інший. Далі — обрати мінімальну для усіх пар кількість, і для відповідної пари вже відшукати маршрут пересадок Петрика (що робиться фактично із тих самих конструктивних міркувань, за допомогою яких ми знаходили кількість пересадок у маршруті).

Дослідимо складність наведеного алгоритму. Для його виконання необхідно перебрати mk пар, для кожної провести $O(n)$ операцій — сортування можна використовувати цифрове (лінійне). Доцільно впорядкувати цифри кожного числа окремо, ще до початку роботи основного алгоритму. Отже, складність алгоритму можна оцінити як $O(nmk)$.

5. Коло

Коло найменшого радіусу завжди буде проходити щонайменше через три задані точки або через дві, якщо вони є кінцями діаметра.

Алгоритм $O(N^4)$ набирає 65% балів. Перебираючи по три точки з даних, визначаємо коло, що їх містить, та кількість точок, що лежать всередині кола. Вибираємо найменший радіус серед тих, що задовольняють умові.

Алгоритм $O(N^2 \ln N \ln X)$, де X — точність, з якою необхідно знайти радіус, набирає 100% балів.

Позначимо радіус кола через r. Виберемо серед даних одну точку A з координатами $(x ; y)$. Коло, що містить цю точку, має центр у точці з координатами вигляду $(x + r\cos\alpha ; y + r\sin\alpha)$.

Для кожної точки крім A визначимо, чи лежить вона всередині кола, що має радіус r і проходить через A, а якщо лежить, то до якого відрізка при цьому належить α.

Нехай A_i належить до кола, якщо α належить до $[a_i ; b_i]$. Впорядкуємо всі точки a_i , b_i , пам'ятаючи при цьому, яка з них була початком, а яка — кінцем. Надамо змінній c початкового значення 1. Перебираючи послідовно елементи впорядкованого масиву, робимо таке:

- якщо елемент є початком деякого відрізка $[a_i ; b_i]$, то збільшуємо c на 1;
- якщо елемент є кінцем деякого відрізка $[a_i ; b_i]$, то зменшуємо c на 1.

Отримана таким чином величина c дорівнює кількості точок, що належать до кола при певному α (за початкову величину взято 1, бо точка A також належить до кола).

Для кожної точки A запам'ятаємо максимальну з отриманих величин c при

проходженні масиву. Максимальна величина c для всіх варіантів вибору A є найбільшою кількістю точок, які можна покрити кругом з радіусом r.

Шуканий радіус r знаходимо двійковим пошуком, використовуючи описані дії.

6. Канікули

У будь-який момент часу подальші затрати і прибуток Петрика залежать лише від населеного пункту v, де він зараз знаходиться, і множини S вже відвіданих сіл. Назвемо таку пару (v; S) станом. Для подальшого руху Петрика не має значення, яким шляхом ми потрапили в поточний стан.

Розв'язання базується на знаходженні мінімальних затрат $t(v, S)$ (без врахування виплат за фотографування весіль), необхідних для досягнення кожного стану (v; S), а потім — вибору найменшого значення $[t(\text{Київ}, S)/|S|] + 1$ за всіма S. Тут $[x]$ — ціла частина числа x.

Інакше кажучи, $t(v, S)$ — найкоротший шлях на графі G переходів між станами з вартостями ребер, що дорівнюють вартостям переходів між станами, від вершини (Київ ; {}) (другий елемент пари — порожня множина) до вершини (v; S). Не обмежуючи загальності міркувань, вважаємо, що граф G повний, у якому відсутні згідно з умовою ребра мають нескінченну вартість.

Множина вже відвіданих сіл не зменшується з часом. Тому ми можемо послідовно знаходити оптимальні значення $t(v, S)$ в порядку збільшення S. Маємо: $t(\text{Київ}, \{\}) = 0$, $t(v, \{\}) = 0$ при $v \neq$ Київ.

Ми можемо потрапити у стан (v; S) двома способами:

а) перейти на останньому кроці зі стану (u; $S \setminus \{v\}$), якщо u належить до $S \setminus \{v\}$ (тобто відвідати село v *вперше*);

б) перейти на останньому кроці зі стану (u; S), якщо u належить до S (тобто відвідати село v *не вперше*).

В обох випадках v належить до S. Вартість кожного переходу — $c(u; v)$.

У будь-якому оптимальному шляху до стану (v; S) існує одне ребро вигляду (а), а всі наступні (можливо, порожня множина) матимуть вигляд (б) з тим самим S *в усіх випадках*. Нехай ми знаємо $t(u, S')$ при всіх u і всіх S', що є власними підмножинами S. Тоді ми можемо знайти найменшу вартість шляху, що закінчується переходом (а) у стан (v; S). Потім будемо шукати найменшу вартість шляху, що закінчується переходами (б) (можливо, з нульовою кількістю таких переходів) у стан (v; S) за допомогою модифікації алгоритма Дейкстри. Псевдокод для сталого S має такий вигляд:

```
для всіх i
 t[i,S] := infinity;

для всіх i з множини S
 для всіх j з множини S\{i}
  t[i,S] := min(t[i,S], t[j,S\{i}] + c(j,i));
```

```
M := {};
поки M відмінне від S
 k := argmin t[k,S] за всіма k, що не належать до M;
 M := M + {k};
 для всіх j  з множини S - M
  t[j,S] := min(t[j,S], t[k,S] + c(k,j));
```

Виконавши це для всіх S у порядку зростання, ми знайдемо t(Київ, S) при всіх S і шукану вартість. Кількість операцій пропорційна $2^N \cdot N^2$.

3. Авторські розв'язання

1. Сновида-2

```
/* GCC */

#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;

const char* inputfile  = "sleep2.in";
const char* outputfile = "sleep2.out";

typedef vector<int> bignumber;

//Нормування числа, щоб в кожному розряді
//значення не більше 9
void Norm(bignumber &a){
     int i = 0;
     while (i<a.size()){
          if (a[i]>9){
               if (i+1==a.size())
                    a.push_back(0);
               a[i+1]+=a[i]/10;
               a[i]%=10;
          }
          ++i;
     }
     while(a.size()>1 && a.back()==0)
          a.pop_back();
}
//Реалізація додавання для великих
//чисел
bignumber operator +(const bignumber& a,
                          const bignumber& b){
     bignumber c(max(a.size(),
                          b.size()),0);
     for(int i = 0; i<a.size()
                          || i<b.size(); ++i){
          if (i<a.size()) c[i]+=a[i];
```

```
          if (i<b.size()) c[i]+=b[i];
     }
     Norm(c);
     return c;
}
//Реалізація множення для великих
//чисел
bignumber operator *(const bignumber& a,
                     const bignumber& b){
     bignumber c(a.size()+b.size(),0);
     for(int i = 0; i<a.size(); ++i)
          for(int j=0; j<b.size(); ++j)
               c[i+j]+=a[i]*b[j];
     Norm(c);
     return c;
}

//Масив C з n по k
bignumber Cnk[101][101];

//Обчислення біноміального коефіцієнта
bignumber& C(int n, int k){
     if (Cnk[n][k].empty()){
          if (n==k || k==0)
               Cnk[n][k].assign(1,1);
          else
               Cnk[n][k]=C(n-1,k-1)
                         +C(n-1,k);
     }
     return Cnk[n][k];
}

vector<bignumber> F(1,bignumber(1,1));

//Обчислення подвійного факторіалу
bignumber& Fnn(int n){
     n=n/2;
     if (n>=F.size()){
          for(int i = int(F.size());
                        i<=n; ++i){
               F.push_back(F.back());
               for(int j = 0;
                    j<F.back().size(); ++j)
                    F.back()[j]*=2*i+1;
               Norm(F.back());
          }
     }
     return F[n];
}

vector<int> permutation;
vector<int> cicle_count;
```

```
//Зчитування вхідних даних
void ReadData(){
     ifstream in(inputfile);
     int N,p;
     in>>N;
     for(int i = 0; i<N; ++i){
          in>>p;
          permutation.push_back(p-1);
     }
     in.close();
}

//Підрахунок кількості циклів
void CalculateNumberOfCicles(){
     cicle_count.assign(
          permutation.size()+1,
          0
     );
     for(int i = 0; i<permutation.size();
                                   ++i){
          if (permutation[i]>=0){
               int c = 0;
               int j = i;
               while(permutation[j]>=0){
                    ++c;
                    int k = permutation[j];
                    permutation[j]=-1;
                    j = k;
               }
               ++cicle_count[c];
          }
     }
}

//Запис відповіді у файл
void WriteData(const bignumber& answer){
     ofstream out(outputfile);
     for(int i = int(answer.size())-1;
                                  i>=0; --i)
          out<<answer[i];
     out<<endl;
     out.close();
}
//Обчислення a^n за O(log n)
bignumber Pow(int a, int n){
          bignumber A(1,1);
          bignumber B(1,a);
          Norm(B);
          while(n>0){
                    if (n&1){
                         A=A*B;
                         B=B*B;
                    }
```

```
                    else{
                         B=B*B;
                    }
                    n/=2;
          }
          return A;
}
//Обчислення відповіді за формулою
bignumber CalcAnswer(){
     bignumber answer(1,1);
     for(int i = 1; i<cicle_count.size();
                                          ++i)
          if (cicle_count[i]>0){
               bignumber tmp(1,1);
               if (i&1){
                    for(int j = 1;
                         2*j<=cicle_count[i];
                                          ++j)
                         tmp=tmp
                         +
                         C(cicle_count[i],2*j)
                         *Fnn(2*j-1)*Pow(i,j);
               }
               else{
                    tmp = (cicle_count[i]&1)?
                              bignumber(1,0) :
                         Fnn(cicle_count[i]-1)
                         *
                         Pow(i,
                         cicle_count[i]/2);
               }
               answer=answer*tmp;
          }
     return answer;
}
```

2. Карти

```
/* GCC */
// Ефективність O(N^2). 30% балів.
#include <cstdio>

const int MaxN = 512000;

FILE *fIn, *fOut;
int N, a, b;
int seq[2][MaxN*2];

int main()
{
     // Read input
     fIn = fopen("cards.in", "rt");
     fscanf(fIn, "%d %d %d", &N, &a, &b);
```

```
     fclose(fIn);

     // Solve
     for (int i = 0; i < 2*N; i++)
          seq[0][i] = i+1;

     while (N > 1) {
/*
          for (int i = 0; i < 2*N; i++)
               printf("%2d ", seq[0][i]);
          puts("");
*/
          for (int i = 0; i < N; i++) {
               seq[1][i]   = seq[0][i*2];
               seq[1][i+N] = seq[0][i*2+1];
          }
          int L = N*a/b;
          int k = 0;
          for (int i = 0; i < N; i++)
               if (i != L)
                    seq[0][k++] = seq[1][i];
          for (int i = N; i < 2*N; i++)
               if (i != L+N)
                    seq[0][k++] = seq[1][i];
          N--;
     }

     // Write answer
     fOut = fopen("cards.out", "wt");
     fprintf(fOut, "%d %d\n", seq[0][0],
                              seq[0][1]);
     fclose(fOut);

     return 0;
}


/* GCC */
// Ефективність O(N). 100% балів.
#include <cstdio>

int N, a, b;

inline int prev(int p, int N)
{
     int L = (N*a)/b;
     if (p < 1 + L)
          return 2*p-1;
     if (p < N)
          return 2*p+1;
     if (p < N+L)
          return 2*(p-(N-1));
     return 2*(p-(N-1))+2;
```

```
}

FILE *fIn, *fOut;

int main()
{
     // Read input
     fIn = fopen("cards.in", "rt");
     fscanf(fIn, "%d %d %d", &N, &a, &b);
     fclose(fIn);

     // Solve
     int x = 1, y = 2;
     for (int i = 2; i <= N; i++) {
          x = prev(x, i);
          y = prev(y, i);
     }

     // Write answer
     fOut = fopen("cards.out", "wt");
     fprintf(fOut, "%d %d\n", x, y);
     fclose(fOut);
}
```

3. Гра

```
/* GCC */
int main(){
     ReadData();
     CalculateNumberOfCicles();
     WriteData(CalcAnswer());
     return 0;
}

#include <fstream>

using namespace std;

ifstream fin("game.in");
ofstream fout("game.out");

int m, n;
int a[24];
int b[24];

char sg[24][24][24][24];

int main()
{
 fin >> m >> n;
 for (int i = 0; i < m; ++i)
  fin >> a[i];
```

```
for (int i = 0; i < n; ++i)
 fin >> b[i];

for (int di = 0; di <= m; ++di)
 for (int i = 0; i < m; ++i)
 {
  int i1 = i + di;

  if (i1 > m)
   break;

  for (int dj = 0; dj <= n; ++dj)
   for (int j = 0; j < n; ++j)
   {
    int j1 = j + dj;
    if (j1 > n)
     break;

    bool reached[24];
    memset(reached, 0, sizeof(reached));

    for (int i2 = i; i2 < i1; ++i2)
     for (int j2 = j; j2 < j1; ++j2)
      if (a[i2] == b[j2])
      {
       char up_code =
        sg[i][i2][j][j2];

       char down_code =
        sg[i2 + 1][i1][j2 + 1][j1];

       char code =
        up_code ^ down_code;

       reached[code] = true;
      }

     char code;
     for (code = 0; code < 24; ++code)
      if (!reached[code])
      {
       sg[i][i1][j][j1] = code;
       break;
      }
   }
 }

 if (sg[0][m][0][n] == 0)
 {
  fout << "LOSE\n";
 }
 else
 {
```

```
   fout << "WIN\n";
   for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
     if (a[i] == b[j])
     {
      char up_code =
       sg[0][i][0][j];

      char down_code =
       sg[i + 1][m][j + 1][n];

      char code =
       up_code ^ down_code;

      if (code == 0)
       fout << i << " " << j << "\n";
     }
  }
}
```

4. Школа

```
/* GCC */
#include <stdio.h>
#define max_routes 100
#define max_digits 1000

int n, m, k, i, j, ij, current_digit,
times, length1, length2, length,
min_length, min_route_home,
min_route_school;
bool min_length_type;
char home[max_routes][max_digits],
new_home[max_digits],
sorted_home[max_routes][max_digits],
school[max_routes][max_digits],
sorted_school[max_routes][max_digits];
int digits[10], places[10],
places_school[max_digits],
places_home[max_digits],
places_home_inv[max_digits];

inline int abs(int a)
{if(a<0) return -a; else return a;}
inline int min(int a, int b)
{if(a<b) return a; else return b;}

int main()
{
    freopen("school.in", "r", stdin);
    freopen("school.out", "w", stdout);
    scanf("%d %d %d\n", &n, &m, &k);
```

```
    for(i=0; i<m; i++)
// Зчитування номерів маршрутів,
// що проходять повз будинок Петрика
    {
        for(j=0; j<10; j++) digits[j]=0;
        for(j=0; j<n; j++)
        {
            scanf("%c", &home[i][j]);
            home[i][j]-='0';
            digits[home[i][j]]++;
        }
        current_digit=0;
        for(j=0; j<10; j++)
        for(ij=0; ij<digits[j]; ij++)
        sorted_home[i][current_digit++]=j;
        // Цифрове сортування цифр
        // щойно зчитаного номера
        if(i==m-1)
        scanf("\n"); else scanf(" ");
    }

    for(i=0; i<k; i++)
    // Зчитування номерів маршрутів,
    // що проходять повз школу Петрика
    {
        for(j=0; j<10; j++) digits[j]=0;
        for(j=0; j<n; j++)
        {
            scanf("%c", &school[i][j]);
            school[i][j]-='0';
            digits[school[i][j]]++;
        }
        current_digit=0;
        for(j=0; j<10; j++)
        for(ij=0; ij<digits[j]; ij++)
       sorted_school[i][current_digit++]=j;
        // Цифрове сортування цифр
        // щойно зчитаного номера
        if(i==m-1) scanf("\n");
        else scanf(" ");
    }

    min_length=max_digits*9;
    for(i=0; i<m; i++) for(j=0; j<k; j++)
    // Пошук двох „кінцевих" маршрутів, для
    // яких кількість пересадок мінімальна
    {
        length1=1;
        length2=0;
        for(ij=0; ij<n; ij++)
        {
           length1+=abs(sorted_home[i][ij]-
```

```
        sorted_school[j][ij]);
        length2+=abs(home[i][ij]-
        school[j][ij]);
    }
    length=min(length1, length2);
    if(length<min_length)
    {
        min_length=length;
        min_length_type=
       (length1<length2);
        min_route_home=i;
        min_route_school=j;
    }
}

printf("%d\n", min_length);
for(i=0; i<n; i++)
printf("%d", home[min_route_home][i]);
if(min_length_type>0)
// Якщо серед операцій
// є операція зміни порядку
{
    for(i=0; i<10; i++) digits[i]=0;
    for(i=0; i<n; i++)
    digits[home[min_route_home][i]]++;
    places[0]=0;
    for(i=1; i<10; i++)
    places[i]=places[i-1]+digits[i-1];
    for(i=0; i<n; i++)
    {
     places_home[i]=
     places[home[min_route_home][i]]++;
     // Місце, на яке після
     // сортування стане цифра, що
     // стоїть на місці з номером i
     places_home_inv[places_home[i]]=i;
     // Місце, на якому до сортування
     // стояла цифра, що стоїть на
     // місці з номером places_home[i]
    }

    for(i=0; i<10; i++) digits[i]=0;
    for(i=0; i<n; i++)
 digits[school[min_route_school][i]]++;
    places[0]=0;
    for(i=1; i<10; i++)
    places[i]=places[i-1]+digits[i-1];
    for(i=0; i<n; i++)
        places_school[i]=
 places[school[min_route_school][i]]++;
        // Місце, на яке після
        // сортування стане цифра, що
        // стояла на місці з номером i
```

```
    for(i=0; i<n; i++)
    new_home[i]=
    home[min_route_home]
   [places_home_inv[places_school[i]]];
    // Зміна порядку цифр першого
    // номера, для якої кількість
    // пересадок мінімальна

    if(new_home[0]==0)
    // Попереднє збільшення нульової
    // цифри у разі, якщо вона
    // опиняється на першому місці
    {
        printf(" ");
        home[min_route_home]
 [places_home_inv[places_school[0]]]=1;
        for(i=0; i<n; i++)
 printf("%d", home[min_route_home][i]);
        new_home[0]=1;
    }

    printf(" ");
    for(i=0; i<n; i++)
    printf("%d", new_home[i]);
    for(i=0; i<n; i++)
    {
        times=new_home[i]-
        school[min_route_school][i];
        for(j=0; j<times; j++)
        {
            new_home[i]--;
            printf(" ");
            for(ij=0; ij<n; ij++)
            printf("%d", new_home[ij]);
        }
        for(j=0; j>times; j--)
        {
            new_home[i]++;
            printf(" ");
            for(ij=0; ij<n; ij++)
            printf("%d", new_home[ij]);
        }
    }
}
else
// Якщо серед операцій
// немає операції зміни порядку
{
    for(i=0; i<n; i++)
    {
        times=home[min_route_home][i]-
        school[min_route_school][i];
```

```
            for(j=0; j<times; j++)
            {
                home[min_route_home][i]--;
                printf(" ");
                for(ij=0; ij<n; ij++)
                printf("%d",
                home[min_route_home][ij]);
            }
            for(j=0; j>times; j--)
            {
                home[min_route_home][i]++;
                printf(" ");
                for(ij=0; ij<n; ij++)
                printf("%d",
                home[min_route_home][ij]);
            }
        }
    }
    printf("\n");
    return 0;
}
```

5. Коло

```
/* GCC */

#include <cstdio>
#include <vector>
#include <algorithm>
#include <cmath>

#include <iostream>

using namespace std;

#define X first
#define Y second

typedef double cfloat;
typedef pair <cfloat, cfloat> PDD;
typedef vector <PDD> VPDD;

const cfloat eps = 1e-10;

// Функція нормування вектора

PDD norm (PDD a)
{
  cfloat x = hypotl (a.X, a.Y);
  return PDD (a.X/x, a.Y/x);
}
```

```
int main ()
{
  freopen ("circle.in", "r", stdin);
  freopen ("circle.out", "w", stdout);

  // Зчитування вхідних даних

  int n, k;
  scanf ("%d%d", &n, &k);

  VPDD a;
  for (int i=0; i<n; i++)
  {
    double x, y;
    scanf ("%lf%lf", &x, &y);
    a.push_back (PDD(x, y));
  }

  cfloat r1 = 0;
  cfloat r2 = 10000;

  vector <pair <cfloat, int> > b(n*n*2);


  // 35 ітерацій бінарного пошуку
  for (int t=0; t<35; t++)
  {
    // Фіксуємо радіус
    cfloat r = (r1+r2)/2;

    int res = 1;

    // Фіксуємо одну з точок на колі
    for (int i=0; i<n; i++)
    {
      int c = 1;
      int bc = 0;
      for (int j=0; j<n; j++)
        if (i!=j)
        {
          // Для кожної з точок визначаємо
          // при яких значеннях кута вона
          // лежатиме всередині кола

          cfloat d = hypotl (a[i].X-a[j].X,
                             a[i].Y-a[j].Y);
          if (d/2>r)
            continue;
          PDD v = norm( PDD(a[j].X-a[i].X,
                            a[j].Y-a[i].Y));

          cfloat l1 = d/2;
          cfloat l2 = sqrtl (r*r - l1*l1);
```

```
          PDD p1 (v.X*l1 + v.Y*l2,
                  v.Y*l1 - v.X*l2);
          PDD p2 (v.X*l1 - v.Y*l2,
                  v.Y*l1 + v.X*l2);

          p1.X/=r; p1.Y/=r;
          p2.X/=r; p2.Y/=r;

          cfloat ang1 = atan2(p1.Y, p1.X);
          cfloat ang2 = atan2(p2.Y, p2.X)
                               +eps;

          if (ang1>ang2)
            c++;
          b[bc++] = make_pair (ang1, 1);
          b[bc++] = make_pair (ang2, -1);
        }

      sort (b.begin(), b.begin()+bc);

      // Підрахунок максимальної кількості
      // точок, що лежатимуть всередині кола
      // фіксованого радіусу, та такого, що
      // проходить через зафіксовану вершину

      for (int j=0; j<bc; j++)
      {
        c+=b[j].Y;
        res = max (res, c);
      }
    }

    if (res>=k)
      r2 = r;
    else
      r1 = r;
  }

  // Виведення результату

  printf ("%.3lf\n", double((r1+r2)/2));

  return 0;
}
```

6. Канікули

```
/* GCC */
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;

#define FOR(i,s,n)for(int i=(s);i<(n);++i)
#define REP(i,n)FOR(i,0,n)

const int INF = 10000000;
int m[1<<17][20];
int a[20][20], t[20];

bool inline good(int mask, int k){
  return k == 0 || (mask&(1<<(k-1))) != 0;
}

int main(){
  freopen("vacation.in","r",stdin);
  freopen("vacation.out","w",stdout);
  int n;
  scanf("%d",&n);
  REP(i,n+1) REP(j,n+1){
    scanf("%d",&a[i][j]);
    if(a[i][j] < 0) a[i][j] = INF;
  }
  REP(i,1<<n) REP(j,n+1) m[i][j] = INF;
  m[0][0] = 0;
  int ans = INF;
  REP(i,1<<n){
    // знаходимо найкоротші шляхи
    // на підграфі {i} алгоритмом Дейкстри
    memset(t,0,sizeof(t));
    REP(k,n)
      if((i & (1<<k)) == 0)
        t[k+1] = 1;
    while(1){
      int best = INF, u = -1;
       // виберемо найдешевшу вершину u
      REP(k,n+1)
        if(t[k] == 0 && m[i][k] < best){
        best = m[i][k];
        u = k;
       }
      if(u == -1) break;
      t[u] = 1;
       // релаксуємо вартість по всіх
       // ребрах (u,k)
      REP(k,n+1)
        if(a[u][k] != INF && good(i,k) &&
```

```
          m[i][k]>m[i][u]+a[u][k])
          m[i][k]=m[i][u]+a[u][k];
     }
     // релаксуємо вартість по всіх ребрах
     // які ведуть з множини вершин {i}
     REP(j,n+1)
      if(a[0][j] != INF && j != 0){
        int t = i | (1<<(j-1));
        m[t][j] = min(m[t][j],
                          m[i][0] + a[0][j]);
      }
    FOR(k,1,n+1){
      if(a[k][0] != INF) m[i][0] =
        min(m[i][0], m[i][k] + a[k][0]);
      FOR(j,1,n+1) if(a[k][j] != INF){
        int t = i | (1<<(j-1));
        m[t][j] =
            min(m[t][j], m[i][k] + a[k][j]);
      }
    }
     // вибираємо найкращу відповідь
    if(m[i][0] < INF && i > 0){
      int cc = 0;
      REP(k,n) if(i & (1<<k)) ++cc;
      if(m[i][0]/cc + 1 < ans)
        ans = m[i][0]/cc + 1;
    }
  }
  printf("%d\n",ans);
  return 0;
}
```