**Ю.Знов'як, Д.Кордубан, Д.Мисак,
М.Рибак О.Рибак, О.Рудик, В.Ткачук**

Олімпіада інформатики у місті Києві у 2006-2007 навчальному році

Передмова голови журі Олександра Рудика

Стаття містить умови завдань III (міського) етапу олімпіади з основ інформатики й обчислювальної техніки у місті Києві у 2007 році та авторські розв'язання цих завдань.

Проведенню III етапу у 2007 році у місті Києві передувало проведення II (районного) етапу таким чином.

1. Завдання для II етапу Всеукраїнської учнівської олімпіади з інформатики у місті Києві у 2006 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.
2. 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 2006 році це було завдання № 3 відбірково-тренувальних зборів (оптимальний розподіл фінансів на депозитних рахунках за допомогою методу динамічного програмування та визначення опору між певними клемами системи опорів). Ці алгоритмічно змістовні задачі задумано як кваліфікаційні завдання щодо реалізації відомих алгоритмів. Вони були обов'язкові для використання на II етапі олімпіади в усіх районах міста Києва;
   - третю задачу задумано як задачу з нескладною моделлю і простим для реалізації алгоритмом (встановлення кількості граней кожного типу регулярного многогранника за відомими кількостями граней і ребер). Для повнішого врахування особливостей районів міста Києва щодо підготовки учнів до олімпіади з інформатики журі II етапу олімпіади могло замінити цю (третю) задачу на довільну (свою) зі збереженням розподілу балів між задачами (не більше 3/8 загальної суми балів на третю задачу).
3. Порядок проведення II етапу олімпіади майже повністю збігався з порядком проведення III етапу олімпіади. Журі та організаційні комітети II 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.

4. Архів тестових файлів до завдань II етапу олімпіади з інформатики разом з системою тестування було вперше опубліковано на вказаному сайті в день проведення II 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 III 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. Сортувальник (автор – Олександр Рибак)

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

Корпорація планує розробку апарата-сортувальника, який впорядковує числа. Прилад влаштовано таким чином. У ньому є N елементів пам'яті, в кожному з яких зберігається число. Сортування здійснюється за допомогою операцій обміну вмістом між деякими парами елементів. Створити зв'язки між всіма парами елементів виявилося неможливим. Тому обміни зробили можливими лише між *першим* і будь-яким іншим елементом.

Завдання

Визначте, яку *найменшу* кількість операцій обміну між парами елементів потрібно зробити для даного початкового розташування, щоб відсортувати числа в елементах пам'яті за зростанням.

Вхідні дані

Перший рядок файлу містить натуральне число N $(1 \le N \le 30000)$.

Другий рядок файлу містить N *попарно різних* цілих чисел, що розташовані відповідно у першому, другому, ... , N-му елементі пам'яті на початку сортування. Всі числа знаходяться у межах від 0 до 30000 включно.

Вихідні дані

Файл має містити одне число M — найменшу можливу кількість операцій обміну між парами елементів пам'яті для досягнення впорядкування чисел.

Приклад

sort.in	sort.out
5 1 6 5 9 8	6

Для поданого прикладу даних можна запропонувати такі обміни.

1) **16**598 → **61**598,
2) **61**598 → **51**698,
3) **51**698 → **15**698,
4) **1**569**8** → **9**561**8**,
5) **9**561**8** → **8**561**9**,
6) **8**561**9** → **1**568**9**.

2. Квадрат Рубика (автор Данило Мисак)

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

Навчившись складати кубик Рубика і прагнучи до нових життєвих перемог, Сашко вигадав свою власну головоломку. Нажаль, він поки не знає, як її вирішити. Сашко розповідає про свою головоломку таке.

Комірки квадратної таблиці $2n \times 2n$ (по $2n$ комірок у кожному рядку та по $2n$ комірок у кожному стовпчику), заповнено натуральними числами, що не перевищують 100. За один крок дозволяється дзеркально відобразити («перегорнути») довільний рядок або стовпчик таблиці. Далі подано кілька прикладів здійснення такої операції.

Початкове заповнення таблиці	Заповнення таблиці після здійснення одного з можливих ходів. Виділено рядок або стовпчик, що було дзеркально відображено		
5 5 1 3 3 4 5 4 1 5 1 4 4 1 3 3	5 5 1 3 **4 5 4 3** 1 5 1 4 4 1 3 3	**4** 5 1 3 **1** 4 5 4 **3** 5 1 4 **5** 1 3 3	5 5 1 **3** 3 4 5 **4** 1 5 1 **4** 4 1 3 **3**

Невідомо, чи можливо за допомогою таких операцій отримати таблицю, у комірки кожного з чотирьох кутових квадратів $n \times n$ якої було б записано однакові числа. Числа у різних кутових квадратах можуть бути відмінними одне від

одного. Якщо це можливо, потрібно вказати послідовність операцій, яка приводить до бажаного результату. Наприклад, таку таблицю:

3	4	1	1
4	1	4	2
1	2	3	3
2	3	2	4

можна перетворити до потрібного вигляду за 4 кроки (виділено рядки та стовпчики, над якими було здійснено операцію на попередньому кроці):

2	4	1	1
1	1	4	2
4	2	3	3
3	3	2	4

1	**1**	**4**	**2**
1	1	4	2
4	2	3	3
3	3	2	4

1	1	4	2
1	1	4	2
3	**3**	**2**	**4**
3	3	2	4

1	1	**2**	2
1	1	**2**	2
3	3	**4**	4
3	3	**4**	4

Завдання

Визначте, чи можливо перетворити потрібним чином таблицю з даним початковим заповненням. Якщо це можливо, вкажіть послідовність ходів, після здійснення яких таблиця набуває потрібного вигляду.

Вхідні дані

У першому рядку вхідного файлу вказано число t — кількість тестів, дані до яких подано у файлі. Далі розташовано t однакових за форматом блоків, що описують тести. Порожніх рядків між блоками немає.

У першому рядку кожного з блоків вказано натуральне число n.

У наступних $2n$ рядках блока міститься по $2n$ натуральних чисел, що задають початкове заповнення таблиці.

При цьому $1 \le t \le 10$, $1 \le n \le 100$, а кожне з чисел, записаних в комірках таблиці, натуральне та не перевищує 100.

Вихідні дані

Вихідний файл має складатися з t однакових за форматом блоків, що відповідають t тестам, які подано у вхідному файлі. Порядок блоків має бути збережено: перший блок вихідного файлу має відповідати першому блоку вхідного, другий блок — другому, …, останній — останньому. Пустих рядків між блоками бути не має.

У першому рядку кожного з блоків потрібно розташувати число k — кількість операцій, після яких таблиця набуде потрібного вигляду.

У наступних *k* рядках блоку потрібно описати відповідні операції у порядку виконання таким чином: вказати номер рядка або стовпчика, над яким було здійснено операцію. Нумерацію починають з одиниці, стовпчики рахують зліва направо, а рядки згори донизу. Якщо дзеркально відображається рядок, перед його номером ставиться знак «+» (без лапок), якщо стовпчик — знак «–».

Якщо початкове розташування чисел у комірках таблиці не дає можливості звести її до потрібного вигляду, єдиний рядок блоку має містити число «–1».

Приклад

square.in	square.out
2 2 3 4 1 1 4 1 4 2 1 2 3 3 2 3 2 4 3 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	4 -1 +1 +3 -3 -1

3. Гнізда (автор Юрій Знов'як)

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

На вертикальній стіні будинку *N* сімей ластівок побудували свої гнізда таким чином, що жодні два гнізда не розташовані на одній вертикалі або горизонталі.

Нещодавно ластівки вирішили підключити до Інтернету кожне з гнізд. Для цього у ластівки, що живе найвище, розмістили супутникову антену. Усім ластівкам, що живуть нижче, Інтернет підключать через гнізда, розташовані вище.

Технологія підключення Інтернету така. Від найвищого гнізда проводять 2 кабелі: один до найвищого гнізда ліворуч від даного і один до найвищого гнізда праворуч. Після цього дві множини гнізд (ліворуч і праворуч від найвищого) розглядаються абсолютно незалежно. Для обох частин проводиться підключення без урахування протилежної частини. Звісно, якщо в одній з частин немає гнізд,

то кабель в цю сторону не протягують. Відстань між точками (x_1; y_1) та (x_2; y_2) вважається рівною $|x_1 - x_2| + |y_1 - y_2|$.

Завдання

Визначте *найменшу* загальну довжину кабелю, який потрібно придбати для підключення усіх гнізд. Розмірами гнізд знехтувати (вважайте їх точками).

Розв'язання за $O(N)$ набирає 100%, за $O(N \cdot \log N)$ — 75%, за $O(N^2)$ — 30% балів.

Вхідні дані

Перший рядок містить натуральне число N. Наступні N рядків містять прямокутні цілі координати гнізда (x; y). Усі числа вхідного файлу за абсолютною величиною не перевищують 1 000 000 000. Гнізда подано у порядку збільшення їхніх абсцис x.

Тести

У всіх тестах $N \le 250\,000$. У 30% тестів $N \le 3\,000$, а у 75% тестів $N \le 125\,000$.
Відповідь на всі тести не перевищує 10^{18}.

Вихідні дані

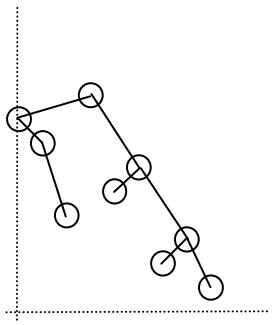
Єдиний рядок файлу містить шукану *найменшу* загальну цілу довжину кабелю.

Зауваження для учасників, що використовують C++

В цій задачі можливі дуже великі вхідні данні, тому не радимо користуватись модулем iostream для введення даних.

Також printf() та fprintf() неправильно виводить числа типу long long.

Приклад

nests.in	nests.out	Графічна ілюстрація
9 0 8 1 7 2 4 3 9 4 5 5 6 6 2 7 3 8 1	27	

4. Згортка (автор Михайло Рибак)

Максимальна оцінка: 100 балів
Обмеження за часом: 1 сек.
Обмеження за пам'яттю: 64 МБ
Вхідний файл: convol.in
Вихідний файл: convol.out
Програма: convol.pas/.c/.cpp

Згорткою кортежу (a_1; a_2; ... ; a_k) називають таку суму добутків:

$$a_1a_k + a_2\,a_{k-1} + a_3\,a_{k-2} + a_4\,a_{k-3} + ... + a_{k-1}\,a_2 + a_k\,a_1.$$

Послідовність додатних чисел $\{a_j\}$ задано таким чином: згортка перших j членів дорівнює 4^{j-1}. Легко впевнитись, що ця послідовність починається так: 1, 2, 6, 20, 70, … — див. таблицю:

	a_N	Перші N членів	Згортка перших N членів	Результат
	1	1	1×1	$1 = 4^0$
	2	1 2	1×2 + 2×1	$4 = 4^1$
	6	1 2 6	1×6 + 2×2 + 6×1	$16 = 4^2$
	20	1 2 6 20	1×20 + 2×6 + 6×2 + 20×1	$64 = 4^3$
	70	1 2 6 20 70	1×70 + 2×20 + 6×6 + 20×2 + 70×1	$256 = 4^4$

Завдання
Обчислити a_N за даним N.

Вхідні дані
Єдиний рядок файлу містить число N, де $0 < N < 2000$.
Принаймні у половині тестів $N < 200$.

Вихідні дані
Єдиний рядок файлу містить цілу частину a_N.

Приклади

convol.in	convol.out
4	20
6	252

5. Комарі (автор Дмитро Кордубан)

Максимальна оцінка: 100 балів
Обмеження за часом: 5 сек.
Обмеження за пам'яттю: 64 МБ
Вхідний файл: mosquito.in

Вихідний файл: mosquito.out
Програма: mosquito.pas/.c/.cpp

Нахабні комарі атакували кімнату Петрика і розсілись на стелі перед початком своїх підступних дій. Перед тим як лягти спати, Петрик хоче знищити максимальну кількість комарів. Для цього він бере улюблений підручник з програмування і сильно підкидає його вгору. Всі комарі, що потрапили під удар, гинуть (навіть ті, що знаходилися на границі області влучання).

Стеля кімнати має форму прямокутника, на якому запроваджено прямокутну систему координат. Осі абсцис *OX* та ординат *OY* розташовано паралельно стінам кімнати. Товщиною підручника нехтуємо. В момент удару підручник знаходиться у площині стелі, а його сторони паралельні осям *OX* та *OY*.

Завдання

Визначте *найбільшу* кількість комарів, яку можна знищити першим кидком, якщо задано координати всіх комарів і лінійні розміри підручника.

Вхідні дані

Перший рядок вхідних даних містить невід'ємне ціле число N — кількість комарів на стелі. Наступні N рядків містять пари невід'ємних цілих чисел x_j і y_j — координати j-го комара на стелі. Останній (N+2)-ий рядок містить 2 натуральних числа w і h — лінійні розміри підручника.

В усіх тестах $N \le 10^5$, $\max\{x_j, y_j\} \le 10^9$, $\max\{w, h\} \le 10^9$.

В 70% тестів $N \le 10^4$, $\max\{x_j, y_j\} \le 10^4$.

В 50% тестів $N \le 10^3$, $\max\{x_j, y_j\} \le 10^3$.

Вихідні дані

Єдиний рядок вихідного файлу має містити одне ціле число — найбільшу кількість комарів, яку можна знищити першим кидком.

Приклади

mosquito.in	mosquito.out
5 0 3 2 1 3 3 4 0 4 3 2 2	3
3 1 1 1 2 1 3 2 1	3

6. Пірати й золото (автор Володимир Ткачук)

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

Команда з N піратів знайшли скарб з M однакових золотих монет і хоче його розділити. Всі пірати мають різний вік, при чому пірат з номером $j + 1$ старший за пірата з номером j. Першим пропозицію, про те як поділити скарб висуває найстарший пірат. Він вказує, скільки золотих монет має отримати кожен учасник команди. Якщо цю пропозицію підтримує не менше половини всіх піратів, то її приймають, і гроші розподіляють відповідним чином. Інакше найстаршого з живих пірата вбивають, а скарб ділять між собою вже $N - 1$ пірат за таким самим принципом. Кожен пірат (перераховано у порядку важливості):

- хоче вижити;
- хоче отримати максимальну кількість монет;
- жадібний і голосує за будь-яку пропозицію тоді й лише тоді, коли він впевнений, що інакше його вб'ють або він отримає менший зиск;
- мислить логічно й повністю передбачає хід думок будь-якого іншого пірата щодо розподілу грошей;
- якщо може розподілити гроші декількома способами при кожному з яких він отримує однаковий зиск, він обирає такий розподіл, при якому молодші пірати отримують більше грошей (розподіл найбільший щодо лексикографічного порядку чисел).

Завдання

За відомими N і M визначте, скільки монет отримає кожний пірат.

Вхідні дані

В єдиному рядку файлу pirat.dat записано два цілих числа: N і M ($1 \le N \le 10000$, $0 \le M \le 1\ 000\ 000\ 000$).

Вихідні дані

Файл містить N рядків по одному цілому числу в кожному рядку.

У j-му рядку потрібно записати кількість монет, які отримає j-ий пірат. Якщо j-го пірата вб'ють, то в j-му рядку потрібно записати число –1.

Приклад

pirat.in	pirat.out
2 100	0

	100
7 2	0 1 0 1 0 0 -1
6 1	1 0 0 0 0 0

2. Ідеї розв'язання

1. Сортувальник

Будемо поділяти обміни на два типи: які повертають деякий *не найменший* елемент на своє місце (тобто на те місце, на якому елемент має стояти у відсортованому масиві), і які не повертають ніякий не найменший елемент на своє місце. Оцінимо мінімальну кількість ходів кожного типу.

За один хід неможливо повернути більше одного не найменшого елемента на своє місце. Тому ходів першого типу — не менше, ніж не найменших елементів, що стоять не на своїх місцях.

Для оцінки кількості ходів другого типу побудуємо наступну конструкцію. Нехай з кожного елемента x виходить стрілка, яка вказує на елемент, що стоїть на місці, яке має зайняти елемент x (якщо елемент x стоїть на своєму місці — стрілка вказуватиме на сам елемент x.) З кожного елемента виходить лише одна стрілка, і на кожен елемент вказує саме одна стрілка. Тому усі елементи за допомогою стрілок розділяються на *цикли*. Назвемо цикл *істотним*, якщо він складається більше, ніж з одного елемента, причому не містить найменшого елемента. Для кожного такого циклу ми маємо зробити щонайменше один хід другого типу. Дійсно, всі елементи істотного циклу стоять не на своїх місцях, бо інакше елемент вказував би сам на себе, отже в циклі був би 1 елемент. Причому жоден з елементів циклу не стоїть на першому місці, інакше на нього вказував би перший елемент. Тому елемент циклу, який *найпершим* прийме участь в деякому обміні, попаде не на своє місце (він попаде на перше місце, а за вибором циклу елемент не є найменшим). При цьому другий елемент, що приймає участь в обміні, *не належить* розглянутому циклу — усі елементи розглянутого циклу поки що стоять не на першому місці. Тому і другий елемент попаде не на своє місце. Отже, ходів другого типу — не менше, ніж істотних циклів.

Покажемо, як діяти, щоб відсортувати масив за N_1+N_2 обмінів, де N_1 —

кількість не найменших елементів, що стоять не на своєму місці, а N_2 — кількість істотних циклів.

Будемо на кожному кроці діяти таким чином: якщо на першому місці стоїть не найменший елемент, то розташуємо його на своєму місці. Якщо ж на першому місці знаходиться найменший елемент, то обміняємо його з будь-яким іншим елементом, який стоїть не на своєму місці. Так будемо діяти, поки всі не найменші елементи не стануть на свої місця. При цьому на своєму місці опиниться і найменший елемент.

При такому методі впорядкування ми на кожному кроці або розташовуємо деякий не найменший елемент на своєму місці, або перетворюємо деякий істотний цикл на не істотний. При цьому у першому випадку ми не змінюємо неістотні цикли, а в другому — не прибираємо не найменші елементи зі своїх місць. Тому всього нам потрібно саме N_1+N_2 обмінів.

Приклад, поданий в умові, якраз реалізує вказаний метод.

Один з варіантів програмної реалізації — розділити програму на два етапи. Перший етап: визначити, *яким по величині* є кожен елемент масиву, і замінити елемент його номером у порядку зростання. Це можна зробити за допомогою сортування. Другий етап: знайти цикли утвореної перестановки чисел від 1 до N (або від 0 до N–1) та кількість не найменших елементів, які не дорівнюють своєму порядку. Для знаходження циклів можна виконати посилання за індексами — тобто від елемента x переходити до елемента, що стоїть на x-тому місці. Можна замість цього відтворити процес впорядкування, описаний у попередньому розділі. Це не вимагає додаткової пам'яті, і при відповідній реалізації не поступається у швидкості.

У авторському розв'язанні сортування здійснюється таким чином. Для кожного можливого значення від 0 до 30000 у масиві `ind` ми відмічаємо, яким по порядку воно зустрілося (або 65535, якщо взагалі не зустрілося). Потім іде «утрушування» значень на початку масиву — отримуємо перестановку чисел від 0 до N–1. Перестановка задана дещо несподіваним чином — для кожного елемента ми вказуємо його місце, а не навпаки. Насправді, ці перестановки є взаємно оберненими та мають однакові значення N_1 та N_2, тому їх взаємозаміна несуттєва.

Далі іде процес впорядкування з підрахунком циклів та повернень елементів на своє місце.

2. Квадрат Рубика.

Назвемо «симетричною четвіркою» будь–яку з четвірок чисел, що стоять у клітинках з координатами (x, y) $(2n+1–x, y)$, $(x, 2n+1–y)$, $(2n+1–x, 2n+1–y)$ (перша координата є номером стовпчика клітинки, друга — номером рядка) для довільних натуральних x та y, що не перевищують $2n$. Необхідною умовою того, що квадрат можна звести до потрібного вигляду є однаковість всіх симетричних четвірок у початковому розташуванні чисел в квадраті Рубика. Наприклад, набір чисел, що стоять у лівій верхній, правій верхній, лівій нижній та правій нижній комірках, має збігатися з набором чисел, що стоять у клітинках з координатами (2, 3), ($2n$–1, 3), (2, $2n$–2), ($2n$–1, $2n$–2). Це випливає з того, що число, яке спочатку

знаходилося в комірці з координатами (x, y), після довільної кількості операцій може опинитися лише в одній з комірок $(2n+1-x, y)$, $(x, 2n+1-y)$, $(2n+1-x, 2n+1-y)$ та самій (x, y) (а отже, набір чисел у кожній з n^2 симетричних четвірок — інваріант). У розташування чисел, до якого необхідно звести квадрат, симетричні четвірки, вочевидь, однакові. Значить, вони мають бути однаковими і для будь–якого розташування чисел, яке можна звести до необхідного.

Покажемо тепер, яким чином можна циклічно переставити довільні три з чотирьох чисел симетричної четвірки, всі інші числа квадрата залишивши при цьому на своїх місцях. Нехай, наприклад, три числа, що треба переставити, стоять у клітинках з координатами (x, y), $(2n+1-x, y)$ та $(x, 2n+1-y)$ і циклічно переставити їх треба таким чином, щоб число, яке стояло в комірці (x, y) перейшло у клітинку з координатами $(2n+1-x, y)$. Тоді, провівши послідовно операції $-x$, $+y$, $-x$, $+y$ ($-x$ — відображення стовпчика з номером x, $+y$ — відображення рядка з номером y), досягнемо бажаного результату. Неважко зрозуміти, що обравши довільні x та y та одну з послідовностей операцій $-x$, $+y$, $-x$, $+y$ або $+y$, $-x$, $+y$, $-x$, ми зможемо довільним чином циклічно переставити будь–які три числа будь–якої симетричної четвірки, не зачіпаючи при цьому інших чисел квадрата.

Назвемо «впорядкованою симетричною четвіркою» симетричну четвірку, на якій зафіксовано порядок чисел. Скажімо, симетричною четвіркою (a, b, c, d) з координатами (x, y), де x та y не перевищують n (тобто, комірка (x, y) лежить у лівій верхній чверті великого квадрата), будемо називати четвірку чисел:

a, що записано в клітинці (x, y);
b, що записано у комірці $(2n+1-x, y)$;
c, що знаходиться в клітинці $(x, 2n+1-y)$;
d, що записане в $(2n+1-x, 2n+1-y)$.

Із зазначеного вище випливає таке твердження. Необхідною умовою того, що квадрат можна звести до потрібного вигляду, є те, що набори чисел в кожній впорядкованій симетричній четвірці однакові. Будемо вважати, що всі ці набори чисел — $\{a, b, c, d\}$ причому числа a, b, c, d попарно різні (інші випадки розглянемо згодом). Зафіксуємо впорядковану симетричну четвірку чисел з координатами (1, 1) (див. вище). Нехай, для визначеності, це буде (a, b, c, d). В силу того, що всі симетричні четвірки в певному розумінні симетричні і жодній з них не можна віддати якусь перевагу, можна вважати, що саме порядок симетричної четвірки (1, 1) буде в кінцевому підсумку збережено. Отже, тепер наше завдання — привести всі впорядковані симетричні четвірки до вигляду (a, b, c, d). Зважаючи на можливість циклічної перестановки трьох чисел четвірки, що не зачіпає інших чисел квадрата, в кожній впорядкованій симетричній четвірці незалежно від інших ми можемо виконувати таку операцію: обрати довільні три числа четвірки і циклічно їх переставити. Наприклад, з четвірки (a, b, c, d) можна отримати (c, b, d, a) або (d, b, a, c) (обрали перше, третє і четверте число). Всього існує 4!=24 різних можливих впорядкованих четвірки з набору попарно різних чисел $\{a, b, c, d\}$.

Рівно половину, 12 із них, за допомогою циклічних переставлень трьох

чисел можна звести до четвірки (a, b, c, d). Крім тривіального варіанту (a, b, c, d) це: (a, c, d, b), (a, d, b, c), (b, a, d, c), (b, c, a, d), (b, d, c, a), (c, a, b, d), (c, b, d, a), (c, d, a, b), (d, a, c, b), (d, b, a, c), (d, c, b, a). Інші ж 12 варіантів неможливо привести до такої четвірки, проте можливо звести до будь–якої з четвірок, яка отримана з четвірки (a, b, c, d) перестановкою двох чисел, наприклад, до четвірки (b, a, c, d). Такими четвірками є (a, b, d, c), (a, c, b, d), (a, d, c, b), (b, a, c, d), (b, c, d, a), (b, d, a, c), (c, a, d, b), (c, b, a, d), (c, d, b, a), (d, a, b, c), (d, b, c, a) і (d, c, a, b). Щоб звести такі четвірки до вигляду (a, b, c, d), нам необхідно відобразити якийсь рядок або стовпчик таблиці й таким чином змінити розташування чисел у інших впорядкованих четвірках. Кожна впорядкована симетрична четверка, отже, може перебувати у двох станах:

0, якщо, її можна звести до виду (a, b, c, d) циклічними перестановками трьох чисел;

1, якщо це неможливо.

При цьому відображення якогось стовпчика або рядка таблиці, що зачіпає четвірку, змінює її стан на протилежний.

Побудуємо ще одну таблицю, $n \times n$, що складається з нулів та одиниць. Число в комірці з координатами (x, y) — стан впорядкованої симетричної четвірки з координатами (x, y) для початкового розташування чисел у квадраті Рубика. У клітинці (1, 1) щойно побудованої таблиці стоїть 0. Тепер задача зводиться до такої. За одну операцію можна всі числа якогось рядка або стовпчика нової таблиці змінити на протилежні (0 на 1, а 1 на 0). Чи можливо вказати таку послідовність операцій, після якої у всіх клітинках таблиці будуть стояти нулі?

Ця задача вирішується вже доволі просто. Спочатку зазначимо, що зміни чисел у рядках і стовпчиках можна виконувати у довільному порядку — від цього результат не зміниться. Крім того, жоден з рядків або стовпчиків немає сенсу міняти два рази або більше, адже зміна стовпчика або рядка два рази повертає таблицю до того самого розташування чисел, яке було б в ній, якби ми жодного разу цей рядок або стовпчик не міняли. Маємо право вважати, що кожен рядок і стовпчик було змінено рівно раз або не було змінено взагалі. Більш того, можемо також вважати, що перший стовпчик не було змінено жодного разу (якби ми могли звести таблицю до нульової, змінивши перший стовпчик, то ми також могли б звести таблицю до нульової, не міняючи його — доведіть це самостійно). Отже, змінено було ті і тільки ті рядки, на перетині яких з першим стовпчиком стоять одиниці. Проведемо ці зміни. Тепер, якщо всі стовпчики складаються або цілком з нулів, або цілком з одиниць, ми, очевидно, зможемо привести таблицю до бажаного вигляду. Інакше ж це зробити неможливо, а значить, і звести квадрат до потрібного вигляду також неможливо.

Якщо всі симетричні четвірки складаються з набору $\{a, b, c, d\}$, принаймні два числа з якого рівні між собою (наприклад, $a = d$, і набір перетворюється у $\{a, b, c\}$), зведення квадрату до потрібного вигляду можливе завжди: з кожної впорядкованої симетричної четвірки вдасться отримати впорядковану четвірку $\{a, b, c, d\}$ за допомогою лише циклічних переставлень трійок чисел.

3. Гнізда

Описана в задачі структура підключень до Інтернету також відома як «Декартове дерево» (англійською *Cartesian tree*). Існує відомий алгоритм на основі динамічного програмування для побудови цього дерева за лінійний час. Опишемо його.

Будемо вважати гнізда відсортованими по горизонталі. Пронумеруємо гнізда зліва на право.

Нехай у нас є побудоване дерево для K–1 гнізда. Нехай r — корінь цього дерева. Нехай right[x] – номер гнізда до якого протягнули вправа від гнізда x, y[x] – висота (y-координата) гнізда x. Розглянемо таку послідовність:

$$a_1 = r, \quad a_2 = \text{right}[a_1], \quad a_3 = \text{right}[a_2], \quad \dots, \quad a_M = \text{right}[a_{M-1}].$$

Очевидно, що $y[a_M] < ... < y[a_3] < y[a_2] < y[a_1]$.

Спробуємо розширити дерево додавши K-те гніздо з координатами (x_0; y_0):

- якщо $y_0 < y[a_M]$, тоді очевидно, треба покласти right[a_M] = K. Таким чином ми отримаємо правильне декартове дерево для K перший гнізд;
- якщо $y[a_M] < y_0 < y[a_{M-1}]$, то для того, щоб підтримати структуру дерева, потрібно покласти right[a_{M-1}] = K, left[K] = a_M ;
- якщо $y[a_{M-1}] < y_0 < y[a_{M-2}]$, тоді потрібно покласти right[a_{M-2}] = K, left[K] = a_{M-1}...

Решта випадків – аналогічні. Окремо виділимо випадок $y[a_1] < y_0$, коли нова вершина повинна стати коренем дерева. Тобто, для підтримання структури потрібно покласти left[K] = a_1.

Псевдокод поданого алгоритму такий.

```
Для K := 2 to N
  // маємо побудоване дерево з перших K-1 вершин
  // пробуємо розширити це дерево додавши вершину (x0, y0)

  якщо y[root] < y0
    // Окремий випадок: K вища за теперішній корінь
    left[K] = root
    root = K // тепер новий корінь
  інакше
    v = K-1; - сама права вершина попереднього дерева

    // Піднімаємось вверх по дереву
    while y[v] < y0
      v = parent[v]

    // Тепер вершина v така, що: y[v] > y0 > y[ right[v] ]
    якщо для v визначена right[v], тоді
      left[K] = right[v]
    right[v] = K
    parent[K] = v
```

Оцінка складності

Поданий алгоритм працює за лінійний час. Це очевидно, враховуючи те, що parent[] від кожної вершини береться не більше одного разу. Отже в сумі за всі ітерації цикл while справдиться не більше N разів.

Деталі реалізації

Поданий псевдокод простіше буде запрограмувати, якщо додати нульову вершину, що вища за всі інші. Тоді не прийдеться розглядати випадок, коли вершина K стає коренем дерева.

4. Згортка

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

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

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

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

Але, мабуть, найбільш оптимальним є такий варіант:

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

Розглянемо розклад шуканого числа на прості множники:

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

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

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

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

$O(N^2)$.

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

5. Комарі

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

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

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

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

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

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

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

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

На першому кроці W містить точку з найменшою абсцисою, на другому x_0 дорівнює другій по величині абсцисі, і т. і. Таким чином, увесь час точки W можна накрити вертикальною смугою ширини w, яка весь час зсувається

праворуч. Позначимо через opt(W) оптимальний розв'язок задачі для кожної такої смуги. Розв'язком всієї задачі буде максимум серед усіх значень opt(W).

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

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

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

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

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

- якщо `v` — внутрішня вершина, то

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

  інакше

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

  тобто

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

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

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

  де T_v — шлях від кореня дерева до v.

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

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

UPDATE(v, a, b, delta)

```
# v ~ [l..r]
# додаємо delta в інтервалі [a..b], що є підінтервалом [l..r]
if [l..r] = [a..b] then
        C[v] := C[v] + delta
        M[v] := M[v] + delta
        return
m = (l+r) div 2
if [l..m] перетинається з [a..b] then
        UPDATE(left[v], a, min{b, m}, delta)
if [m+1..r] перетинається з [a..b] then
        UPDATE(right[v], max{m+1, a}, b, delta)
M[v] = max{M[left[v]], M[right[v]]} + C[v]
return
```

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

6. Пірати

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

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

від пірата №4). П’ятий пірат розділить гроші так: №1 — 1, №2 — 0, №3 — 1, №4 — 0, №5 — 98.

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

Номер пірата, що розподіляє гроші	Розподіл грошей
1	100
2	0 100
3	1 0 99
4	0 1 0 99
5	1 0 1 0 98

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

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

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

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

Алгоритм розв’язку можна описати так:

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

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

Реалізація цього алгоритму «в лоб» буде працювати не швидше ніж за час

$O(N^2)$. Для отримання швидшого розв'язку, потрібно відмовитись від моделювання поведінки піратів. Натомість, можна дослідити деякі властивості шуканого розподілу грошей.

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

Доведення (методом математичної індукції за N).

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

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

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

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

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

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

Складність роботи алгоритму є $O(N)$.

3. Авторські розв’язання

1. Сортувальник

```
{ Turbo Pascal }
PROGRAM Sort;
var ind:array[0..30000] of word;
    {ind[i] - це номер елемента пам'яті
     сортувальника з числом i}
    i,j,k:word; {Індекси для масива}
    n:word;  {Кількість елементів}
    n1:word; {Кількість ненульових
     елементів не на своєму місці}
    n2:word; {Кількість циклів, які
     складаються більше, ніж з одного
     елемента, та не містять 0}
BEGIN
  assign(input,'sort.in');
  reset(input);
  for i:=0 to 30000 do ind[i]:=65535;
  read(n);
  for i:=0 to n-1 do begin
    read(j);
    ind[j]:=i;
  end;
  j:=0;
  {Перетворимо масив ind на перестановку
   чисел від 0 до (n-1)}
  for i:=0 to 30000 do
    if (ind[i]&lt;&gt;65535) then begin
       ind[j]:=ind[i];
       j:=j+1;
    end;
  close(input);

  n1:=0;
  n2:=0;
  {Визначимо величини n1 та n2.}
  k:=0; {k - елемент поточного цикла}
  while (k&lt;n) do begin
    if (k&lt;&gt;0) then Inc(n2);
    {Розставимо елементи поточного
     цикла на місце, щоб не зустріти
     цей цикл знову}
    i:=ind[k];
    while (ind[k]&lt;&gt;k) do begin
      j:=ind[i];
      ind[i]:=i;
      if (i&lt;&gt;0) then Inc(n1);
      i:=j;
    end;
```

```
    {Знайдемо елемент нового істотного
     цикла, тобто цикла, що складається
     більше ніж з одного елемента та не
     містять 0}
    while (k&lt;n)and(ind[k]=k) do Inc(k);
  end;
  assign(output,'sort.out');
  rewrite(output);
  writeln(n1+n2);
  close(output);
END.
```

2. Квадрат

```
{ Free Pascal }
program square;

var     f,o: text;
a: array[1..200,1..200] of byte;
// Числа квадрату Рубика
states: array[1..100,1..100] of byte;
// Таблиця станів симетричних четвірок
moves: array[1..81000] of integer;
// Ходи
movescount: longint;
// Кількість ходів

t,i,j,k,n,l: longint;
sym: array[1..4] of byte;
// Числа кутової симетричної четвірки
numsdifferent: boolean;
// Чи всі числа кутової симетричної
// четвірки попарно різні

procedure swap(var a,b: byte);
// Обмін двох чисел місцями
var t: byte;
begin
t:=a;
a:=b;
b:=t;
end;

procedure swap3(var a,b,c: byte);
// Циклічний обмін трьох чисел місцями
var t: byte;
begin t:=a;  a:=b;  b:=c;  c:=t; end;

function
getstate(var a,b,c,d: byte): byte;
// Функція, що визначає стан заданої
// симетричної четвірки і разом з тим,
```

```
// чи всі симетричні четвірки однакові
var     s,i,j,cnt,samecnt: longint;
nums,nums2,same: array[1..4] of byte;

begin
s:=0;
nums[1]:=a;  nums[2]:=b;
nums[3]:=c;  nums[4]:=d;
for i:=1 to 3 do for j:=i+1 to 4 do
if nums[i]>nums[j] then begin
swap(nums[i],nums[j]);
inc(s);
// Сортуємо числа a, b, c, d та
// запам'ятовуємо кількість обмінів пар
// чисел, які довелося при цьому зробити
end;
if (nums[1]=sym[1]) and (nums[2]=sym[2])
and (nums[3]=sym[3]) and(nums[4]=sym[4])
then if numsdifferent then
getstate:=s mod 2 else begin
// Якщо симетрична четвірка збігається з
// кутовою, то якщо числа четвірки
// попарно різні, повертаємо як стан 0,
// якщо кількість обмінів при сортуванні
// парна і 1, якщо непарна, якщо деякі
// числа рівні між собою, продовжуємо
nums2[1]:=a;  nums2[2]:=b;
nums2[3]:=c;  nums2[4]:=d;
cnt:=100;
for i:=2 to 4 do if nums[i]=nums[i-1]
then begin
inc(cnt);
samecnt:=0;
for j:=1 to 4 do if nums2[j]=nums[i]
then begin
if samecnt=0 then nums2[j]:=cnt;
inc(samecnt);
same[samecnt]:=j;
end;
end;
s:=0;
for i:=1 to 4 do nums[i]:=nums2[i];
for i:=1 to 3 do for j:=i+1 to 4 do
if nums[i]>nums[j] then begin
swap(nums[i],nums[j]);
inc(s);
end;
if s mod 2=1 then
swap(nums2[same[1]],nums2[same[2]]);
a:=nums2[1];  b:=nums2[2];
c:=nums2[3];  d:=nums2[4];
getstate:=0;
```

```
// Заміняємо деякі числа на інші, більші
// за 100, так, щоб не залишилось двох
// однакових чисел і при цьому можна
// було циклічними змінами трійок
// перетворити четвірку на впорядковану.
// Наприклад, четвірка (10, 3, 10, 95)
// поміняється на (10, 3, 101, 95),
// а четвірка (1, 1, 2, 2) на
// (101, 1, 2, 102)
end else getstate:=2;
// Якщо симетрична четвірка не
// збігається із кутовою симетричною
// четвіркою, повертаємо стан 2
end;

procedure process;
// Процедура, що обробляє
// поточний зчитанний квадрат

var     i,j: longint;

begin
movescount:=0;

sym[1]:=a[1,1];
sym[2]:=a[2*n,1];
sym[3]:=a[1,2*n];
sym[4]:=a[2*n,2*n];

for i:=1 to 3 do for j:=i+1 to 4 do if
sym[i]>sym[j] then swap(sym[i],sym[j]);
if (sym[1]=sym[2]) or (sym[2]=sym[3])
or (sym[3]=sym[4]) then
numsdifferent:=false else
numsdifferent:=true;
// Сортуємо числа кутової симетричної
// четвірки і визначаємо, чи є серед
// них рівні

for i:=1 to n do for j:=1 to n do begin
states[i,j]:=
getstate(a[i,j],a[2*n+1-i,j],
a[i,2*n+1-j],a[2*n+1-i,2*n+1-j]);
if states[i,j]=2 then begin
writeln(o,'-1');  exit; end;
// Будуємо таблицю станів симетричних
// четвірок. Якщо числа деякої
// симетричної четвірки не збігаються
// із числами кутової, виводимо -1
end;

for i:=2 to n do for j:=2 to n do
```

```
if abs(states[i,1]-states[1,1])<>
abs(states[i,j]-states[1,j])
then begin writeln(o,'-1'); exit; end;
// Перевіряємо, чи можна звести
// таблицю станів до нульової

for j:=1 to n do
if states[1,j]=1 then begin
inc(movescount);
moves[movescount]:=j;
for i:=1 to n do
swap(a[i,j],a[2*n+1-i,j]);
end;
for i:=2 to n do
if abs(states[i,1]-states[1,1])=1
then begin
inc(movescount);
moves[movescount]:=-i;
for j:=1 to n do
swap(a[i,j],a[i,2*n+1-j]);
end;
// Зводимо таблицю станів симетричних
// четвірок до нульової, при цьому
// запам'ятовуючи зроблені ходи і
// змінюючи розташування чисел у
// симетричних четвірках

sym[1]:=a[1,1];
sym[2]:=a[2*n,1];
sym[3]:=a[1,2*n];
sym[4]:=a[2*n,2*n];
// Запам'ятовуємо нове розташування
// чисел у кутовій четвірці

for i:=1 to n do for j:=1 to n do begin
if a[2*n+1-i,j]=sym[1] then begin
swap3(a[i,j],a[2*n+1-i,j],a[i,2*n+1-j]);
inc(movescount,4);
moves[movescount-3]:=j;
moves[movescount-2]:=-i;
moves[movescount-1]:=j;
moves[movescount]:=-i;
end else if a[i,2*n+1-j]=sym[1]
then begin
swap3(a[i,j],a[i,2*n+1-j],a[2*n+1-i,j]);
inc(movescount,4);
moves[movescount-3]:=-i;
moves[movescount-2]:=j;
moves[movescount-1]:=-i;
moves[movescount]:=j;
end else if a[2*n+1-i,2*n+1-j]=sym[1]
then begin
```

```
swap3(a[i,j],a[2*n+1-i,2*n+1-j],
a[2*n+1-i,j]);
inc(movescount,4);
moves[movescount-3]:=j;
moves[movescount-2]:=-(2*n+1-i);
moves[movescount-1]:=j;
moves[movescount]:=-(2*n+1-i);
end;

if a[i,2*n+1-j]=sym[2] then begin
inc(movescount,4);
moves[movescount-3]:=-(2*n+1-i);
moves[movescount-2]:=2*n+1-j;
moves[movescount-1]:=-(2*n+1-i);
moves[movescount]:=2*n+1-j;
end else if a[2*n+1-i,2*n+1-j]=sym[2]
then begin
inc(movescount,4);
moves[movescount-3]:=2*n+1-j;
moves[movescount-2]:=-(2*n+1-i);
moves[movescount-1]:=2*n+1-j;
moves[movescount]:=-(2*n+1-i);
end;
end;
// Перебором варіантів за допомогою
// циклічних переставлень трійок чисел
// зводимо всі симетричні четвірки до
// однакового вигляду, при цьому
// запам'ятовуємо зроблені ходи

writeln(o,movescount);
for i:=1 to movescount do
if moves[i]>0 then
writeln(o,'+',moves[i]) else
writeln(o,moves[i]);
// Виводимо запам'ятовану
// послідовність ходів
end;

begin
assign(f,'square.in');
reset(f);
assign(o,'square.out');
rewrite(o);
readln(f,k);
for t:=1 to k do begin
readln(f,n);
for i:=1 to 2*n do
for j:=1 to 2*n do read(f,a[j,i]);
// Зчитування чергового квадрата
process;
// Обробка зчитаного квадрата
```

```
end;
close(o);
close(f);
end.
```

3. Гнізда

```
/* GCC */
#include <cstdio>
#include <cstdlib>

using namespace std;

const int MaxN = 256000;
const int INF = 1024000000;

int N;
int x[MaxN], y[MaxN], pLeft[MaxN], pRight[MaxN];
int stackTop, stack[MaxN];

typedef long long i64;

inline int top()
{
   return stack[stackTop];
}

inline void pop()
{
   stackTop--;
}

inline void push(int x)
{
   stackTop++;
   stack[ stackTop ] = x;
}

inline i64 dist(int u, int v)
{
   if ((u == -1) || (v == -1))
       return 0;

   return i64(abs(x[u]-x[v])) + abs(y[u]-y[v]);
}

int main()
{
   // Відкриваємо файл для зчитування
   FILE *fIn = fopen("nests.in", "rt");
   fscanf(fIn, "%d", &N);
```

```
   stackTop = -1;

   push(0);
   y[0] = INF;

   for (int i = 1; i <= N; i++) {
        fscanf(fIn, "%d %d", &x[i], &y[i]);
        pLeft[i] = pRight[i] = -1;

        while (y[ top() ] < y[i]) {
             pLeft[i] = top();
             pop();
        }

        pRight[ top() ] = i;
        push(i);
   }

   fclose(fIn);

   // Рахуємо відповідь і виводимо її у файл
   i64 length = 0;
   for (int i = 1; i <= N; i++)
       length += dist(i, pLeft[i]) + dist(i, pRight[i]);

   int high = length / 1000000000, low = length % 1000000000;
   FILE *fOut = fopen("nests.out", "wt");
       if (high)
           fprintf(fOut, "%d%09d\n", high, low); else
           fprintf(fOut, "%d\n", low);
   fclose(fOut);
   return 0;
}
```

4.Згортка

```
/* GCC */


#include <fstream>
std::ifstream fin("convol.in");
std::ofstream fout("convol.out");


#define FOR(i,a,b) for (int _n(b), i(a); i < _n; i++)
#define REP(i,n) FOR(i,0,n)
#define FORD(i,a,b) for(int i=(a),_b=(b);i>=_b;--i)
#define MSET(a, v) memset(a, v, sizeof(a))
#define MSET0(a) MSET(a, 0)

bool IsPrime(int x)
{
```

```
  FOR(i, 2, x)
    if (x % i == 0)
      return false;
  return true;
}

int CountInFac(int x, int k)
  //how many times k! is divisible by x
{
  int r = 0;
  for (int t = x; t <= k; t *= x)
    r += k / t;
  return r;
}

int n;

const int MAX_LEN = 1 << 11;

int a[MAX_LEN];

int Mul(int x, int power)
  //multiply current result by x^power
{
  REP(cp, power)
  {
    int d = 0;
    REP(i, MAX_LEN)
    {
      d += x * a[i];
      a[i] = d % 10;
      d /= 10;
    }
  }
}

int main()
{
  fin >> n;
  --n;

  MSET0(a);
  a[0] = 1;

  FOR(i, 2, 2 * n + 1)
    if (IsPrime(i))
      Mul(i, CountInFac(i, 2 * n) - 2 * CountInFac(i, n));

  for (int i = MAX_LEN - 1, show = 0; i >= 0; --i)
    if (show || a[i])
    {
      fout << a[i];
```

```
      show = 1;
    }

  return 0;
}
```

5. Комарі

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

#define REP(i,n) for(int i=0;i<(n);++i)
#define all(x) (x).begin(),(x).end()
#define X first
#define Y second

#define DIM 102400
deque<pair<int,int> > q;
int n,area,x[DIM],y[DIM];

#define DIM_2 409600
int mx[DIM_2],cr[DIM_2];

void inline init(){
   memset(mx,0,sizeof(mx));
   memset(cr,0,sizeof(cr));
}

// реалізує UPDATE
void change(int p1,int p2,int q,
             int l,int r,int val){
   if(p1==l && p2==r){
        cr[q] += val;
        mx[q] += val;
        return;
   }
   int m = (l+r)/2;
   if(p1<=m && p2<=m)
        change(p1,p2,q*2,l,m,val);
   else if(p1>m && p2>m)
        change(p1,p2,q*2+1,m+1,r,val);
   else{
        change(p1,m,q*2,l,m,val);
        change(m+1,p2,q*2+1,m+1,r,val);
   }
   mx[q]=max(mx[2*q],mx[2*q+1])+cr[q];
}
```

```
void inline add_range(int from,int to){
   change(from,to,1,0,n+5,1);
}

void inline sub_range(int from,int to){
   change(from,to,1,0,n+5,-1);
}

// реалізує MAX
int inline find_max(){
   return mx[1];
}

int yy[DIM]; // V
int uniq;

// реалізує upper(d)
inline int y2n(int d){
   int ans =
        lower_bound(yy,yy+uniq,d)-yy;
   if(ans<uniq && yy[ans]==d) ++ans;
   return ans;
}

int inline f(int w,int h){
   init();
   int p1 = 0,p2 = 0;
   // x[p1] - x_0 для поточної смуги
   while(p2<n && x[p2]<=x[0]+w){
        add_range(y2n(y[p2]),
                  y2n(y[p2]+h));
        ++p2;
   }
   int best = find_max();
   while(p2<n){
        int t = x[p1];
        while(p1<n && x[p1]==t){
             sub_range(y2n(y[p1]),
                       y2n(y[p1]+h));
             ++p1;
        }
        while(p2<n && x[p2]<=x[p1]+w){
             add_range(y2n(y[p2]),
                       y2n(y[p2]+h));
             ++p2;
        }
        best = max(best,find_max());
        if(best==n||best==(w+1)*(h+1))
             break;
   }
   return best;
```

```
}

int main()
{
   freopen("mosquito.in","r",stdin);
   freopen("mosquito.out","w",stdout);
   int w,h,u,v;
   scanf("%d",&n);
   vector<pair<int,int> > b(n);
   REP(i,n){
        scanf("%d %d",&u,&v);
        b[i].X = u;
        b[i].Y = v;
        yy[i] = v;
   }
   scanf("%d %d",&w,&h);
   sort(all(b));
   sort(yy,yy+n);
   uniq = unique(yy,yy+n)-yy;
   REP(i,n){
        x[i] = b[i].X;
        y[i] = b[i].Y;
   }
   printf("%d\n",max(f(w,h),f(h,w)));
   return 0;
}
```

6. Пірати

```
/* GCC */
#include<vector>
#include<fstream>
using namespace std;
#define sz size()
vector<int> get_r(int N, int M, int K){
   vector<int> A(N,0);
   for(int i = K; i<N&&M>0; i+=2){
        A[i] = 1;
        --M;
   }
   A[N-1]+=M;
   return A;
}
vector<int> get_R(int N, int M){
   if (M>=(N-1)/2) return get_r(N,M,1-N%2);
   int K = (M+1)*2;
   int C = 0;
   int v = 1-K%2;
   for (int i = K+1; i<=N; ++i){
        if (C+M>=(i-1)/2){
             C = 0;
             v^=1;
```

```
            }
            else
                ++C;
        }
        vector<int> A = get_r(N,M,v);
        for(int i = N-1; C>0; --i,--C)
            A[i] = -1;
        return A;
}
int main(){
        int N,M;
        ifstream in("pirat.in");
        ofstream out("pirat.out");
        in>>N>>M;
        vector<int> D = get_R(N,M);
        for(int i = 0; i<D.size(); ++i)
            out<<D[i]<<endl;
        in.close();
        out.close();
        return 0;
}
```