1. Граф

Опишемо спочатку *структуру даних* для списків суміжності вершин.

Черга — структура даних, що працює за принципом «хто раніше прийшов, той раніше пішов». У черги є голова та хвіст. Основні операції з чергою такі:

- «Поставити в чергу» — додати елемент у «хвіст» черги. Довжину черги при цьому буде збільшено на одиницю.
- «Отримати з черги» — повернути значення елемента з голови та видалити його з черги, встановлюючи голову черги на наступний за видаленим елемент. Довжину черги при цьому буде зменшено на одиницю.

Можливо реалізувати чергу за допомогою масиву `a[1...n]`, в якому зберігають дані, та двох додаткових змінних `head` і `tail`, у яких зберігають індекси відповідно «голови» та «хвоста» черги. Основні операції можна записати кількома рядками:

«поставити в чергу»	«отримати з черги»

`if tail=n then tail:=1`
`  else tail:=tail+1;`
`a[tail]:=x;`
	`x:=a[head];`
	`if head=n then head:=1`
	`  else head:=head+1;`

Але у випадку багатьох черг різної довжини чергу доречно реалізувати з використанням динамічного розподілу пам'яті (або скористатися вже готовою структурою даних `queue` у C++). Крім черг для списків суміжності вершин, одну додаткову чергу `edges` потрібно використати для незаблокованих ребер (означення див. далі).

Опишемо *жадібний алгоритм* розв'язання задачі: послідовно виписувати ті ребра, які в таблиці суміжності не заблоковано жодним іншим (ще не виписаним) ребром, і видаляти їх із таблиці.

Незаблокованими вважати ребра (*A*, *B*) з такими властивостями:

- першою у списку суміжності вершини *A* іде на даний момент вершина *B;*
- першою у списку суміжності вершини *B* іде на даний момент вершина *A*.

Початковим набором незаблокованих ребер чергу `edges` можна заповнити просто під час зчитування вхідних даних або за допомогою додаткового лінійного проходу. Після цього чергу потрібно пройти «від голови до хвоста» і на кожному кроці:

- вивести й видалити з черги поточне ребро (*A*, *B*);
- видалити це саме ребро з таблиці суміжності — видалити перші вершини з черг вершин *A* і *B*;
- проаналізувати нові перші вершини в чергах *A* та *B* і в разі необхідності додати одне або два нових ребра до черги незаблокованих ребер.

Процес завершується тоді, коли в черзі більше немає ребер.

Як можна зрозуміти, час виконання алгоритму є пропорційним до кількості чисел у вхідному файлі.

2. Дороги

У термінах теорії графів дану задачу називають пошуком мінімального кістякового дерева — зв'язного графа без циклів, що містить усі вершини початкового графа і має найменшу суму ваг ребер серед усіх таких графів. Одним із способів розв'язання цієї задачі є жадібний *алгоритм Прима* (Prim)*:*

1. Утворити дерево T_1, що містить:
   - одне ребро, що має *найменшу вагу* серед ребер початкового графа;
   - дві вершини — кінці цього ребра найменшої ваги

   та надати значення $k = 1$.
2. Якщо існують вершини початкового графа G зовні останнього побудованого дерева T_k з ребрами $e_1, e_2, e_3, \ldots, e_k$, то зробити таке:
   - вибрати ребро e_{k+1} з найменшою вагою серед тих, у яких одна вершина належить до T_k, а інша вершина не належить;
   - утворити дерево T_{k+1} долученням до T_k вибраного ребра e_{k+1} і його вершини, яка не належить до T_k;
   - збільшити значення k на 1;
   - перейти на початок виконання пункту 2.
3. Якщо всі вершини початкового графа G належать до дерева T_k, то припинити побудову мінімального кістякового дерева.

Теорема. *Алгоритм Прима породжує мінімальне кістякове дерево.*

Доведення (від супротивного). Позначимо через $e_1, e_2, e_3, \ldots, e_n$ ребра дерева T_n у тому порядку, як їх вибрано згідно з алгоритмом Прима. Нехай k — найбільше таке ціле невід'ємне число, при якому $e_1, e_2, e_3, \ldots, e_k$ — ребра деякого мінімального кістякового дерева T'. Припустимо, що $k < n$. Приєднаємо ребро e_{k+1} до мінімального кістякового дерева T'. В утвореному таким чином графі є цикл. Цикл містить ребро e, відмінне від e_{k+1}, що також має лише одну вершину в дереві T_k. Утворимо дерево T'', вилучивши з T' ребро e і долучивши ребро e_{k+1}. Згідно з алгоритмом Прима, вага e_{k+1} не перевищує вагу e, тому T'' є мінімальним каркасним деревом, що суперечить означенню k. Отже, $k = n$, T_n — мінімальне каркасне дерево.

Алгоритм Прима використано в авторському розв'язанні для калібрування системи тестування. На думку автора завдання, найімовірніше очікувати використання саме цього алгоритму учасниками олімпіади. Приклад вихідного файлу може наштовхнути на думку використати саме цей жадібний алгоритм.

Не менше відомий *алгоритм Крускала* (Kruskal):

1. Надати множині E значення величини $\emptyset$ — порожньої множини.
2. Визначити, які з ребер початкового графа, що не належать до E, при долученні до E не утворюють циклів.
3. Якщо такі ребра є, то:
   - серед цих ребер визначити «найлегше» — з найменшою вагою;
   - долучити це ребро до множини E;
   - перейти до виконання пункту 2.
4. Якщо таких ребер немає, то припинити побудову мінімального кістякового дерева E.

Учасникам відбірково-тренувальних зборів радимо звернути увагу на *алгоритм Борувки,* що він гарантує максимальну кількість балів. Він полягає у

послідовному долученні ребер до лісу, що містить усі вершини, а ребра утворюють окремі дерева, доки ліс не перетвориться на одне дерево:

1. Нехай спочатку T — порожня множина ребер. Інакше кажучи, починаємо з лісу, до якого кожна вершина входить як окреме дерево (без ребер).
2. Поки T не є деревом (кількість ребер у T менше ніж $V - 1$, де V — кількість вершин у графі), робити таке:
   - для кожної компоненти зв'язності, що є деревом поточного лісу, знайти одне найлегше ребро, що пов'язує цю складову з будь-якою іншою складовою зв'язності. Якщо таких ребер кілька, вибрати довільне, але лише одне;
   - долучити усі знайдені ребра до множини T.

Отримана у результаті ребер T є мінімальним кістяковим деревом початкового графа. Уперше цей алгоритм опубліковано 1926 року Отакаром Борувкою як метод пошуку оптимальної електричної мережі у Моравії.

У програмі найголовніше організувати облік належності вершин до складових (дерев). Для цього в авторському розв’язанні використано лінійні масиви:

- c — утворений записами з такими полями:
  - p — вказівник на попередній елемент опису складової;
  - f — вказівник на перший елемент опису складової;
  - k — номер ребра, що належить до складової або яке потрібно долучити;
  - m — вага ребра з номером k;
- p — вказує на елемент c, що завершує опис складової, що містить дану вершину;
- v — вказує на елемент c, що завершує опис даної складової.

При цьому для опису динамічної структури не потрібно використовувати динамічний розподіл пам’яті.