**ЗМІСТ**

[Теоретична частина 8](#_Toc116552911)

[Алгоритм Дейкстри – Інтуітивно 8](#_Toc116552912)

[Крок 1 9](#_Toc116552913)

[Крок 2 9](#_Toc116552914)

[Крок 3 10](#_Toc116552915)

[Кроки 4, 5 10](#_Toc116552916)

[Крок 6 10](#_Toc116552917)

[Крок 7 11](#_Toc116552918)

[Крок 8 11](#_Toc116552919)

[Крок 8 (з іншими вхідними даними) 12](#_Toc116552920)

[Крок 9 12](#_Toc116552921)

[Крок 10 13](#_Toc116552922)

[Кроки 11 — 15 13](#_Toc116552923)

[Наступні кроки 14](#_Toc116552924)

[Завершення виконання алгоритму 14](#_Toc116552925)

[Пошук з поверненням 14](#_Toc116552926)

[**Огляд** 15](#_Toc116552927)

[**Опис методу** 16](#_Toc116552928)

[**Псевдокод** 17](#_Toc116552929)

[Алгоритм Флойда — Воршелла 18](#_Toc116552930)

[**Знай свои данные — не плотные графы** 29](#_Toc116552931)

[**Знай свое оборудование — параллелизм** 32](#_Toc116552932)

[**Знай свою платформу — векторизация** 35](#_Toc116552933)

[**Знаю свою платформу и свое оборудование — векторизация и параллелизм!** 41](#_Toc116552934)

[Завдання для всіх варіантів 44](#_Toc116552935)

[Варіант 1 45](#_Toc116552936)

[Входные данные 45](#_Toc116552937)

[Выходные данные 45](#_Toc116552938)

[Пример 45](#_Toc116552939)

[Варіант 2 45](#_Toc116552940)

[Входные данные 45](#_Toc116552941)

[Выходные данные 46](#_Toc116552942)

[Пример 46](#_Toc116552943)

[Варіант 3 46](#_Toc116552944)

[Входные данные 47](#_Toc116552945)

[Выходные данные 47](#_Toc116552946)

[Примеры 47](#_Toc116552947)

[Варіант 4 47](#_Toc116552948)

[Входные данные 47](#_Toc116552949)

[Выходные данные 48](#_Toc116552950)

[Пример 48](#_Toc116552951)

[Варіант 5 48](#_Toc116552952)

[Входные данные 48](#_Toc116552953)

[Выходные данные 48](#_Toc116552954)

[Пример 48](#_Toc116552955)

[Варіант 6 49](#_Toc116552956)

[Входные данные 49](#_Toc116552957)

[Выходные данные 49](#_Toc116552958)

[Примеры 50](#_Toc116552959)

[Варіант 7 50](#_Toc116552960)

[Входные данные 50](#_Toc116552961)

[Выходные данные 50](#_Toc116552962)

[Пример 51](#_Toc116552963)

[Варіант 8 51](#_Toc116552964)

[Входные данные 51](#_Toc116552965)

[Выходные данные 52](#_Toc116552966)

[Примеры 52](#_Toc116552967)

[Варіант 9 52](#_Toc116552968)

[Входные данные 52](#_Toc116552969)

[Выходные данные 52](#_Toc116552970)

[Примеры 53](#_Toc116552971)

[Варіант 10 53](#_Toc116552972)

[Входные данные 53](#_Toc116552973)

[Выходные данные 53](#_Toc116552974)

[Пример 54](#_Toc116552975)

[Варіант 11 54](#_Toc116552976)

[Входные данные 54](#_Toc116552977)

[Выходные данные 54](#_Toc116552978)

[Примеры 54](#_Toc116552979)

[Варіант 12 55](#_Toc116552980)

[Входные данные 56](#_Toc116552981)

[Выходные данные 56](#_Toc116552982)

[Пример 56](#_Toc116552983)

[Варіант 13 56](#_Toc116552984)

[Входные данные 57](#_Toc116552985)

[Выходные данные 57](#_Toc116552986)

[Примеры 57](#_Toc116552987)

[Варіант 14 57](#_Toc116552988)

[Входные данные 58](#_Toc116552989)

[Выходные данные 59](#_Toc116552990)

[Пример 59](#_Toc116552991)

[Варіант 15 59](#_Toc116552992)

[Входные данные 60](#_Toc116552993)

[Выходные данные 60](#_Toc116552994)

[Пример 60](#_Toc116552995)

[Варіант 16 61](#_Toc116552996)

[Входные данные 61](#_Toc116552997)

[Выходные данные 61](#_Toc116552998)

[Пример 61](#_Toc116552999)

[Варіант 17 61](#_Toc116553000)

[Входные данные 62](#_Toc116553001)

[Выходные данные 63](#_Toc116553002)

[Пример 63](#_Toc116553003)

[Варіант 18 63](#_Toc116553004)

[Входные данные 63](#_Toc116553005)

[Выходные данные 64](#_Toc116553006)

[Пример 64](#_Toc116553007)

[Варіант 19 64](#_Toc116553008)

[Входные данные 64](#_Toc116553009)

[Выходные данные 64](#_Toc116553010)

[Пример 64](#_Toc116553011)

[Варіант 20 65](#_Toc116553012)

[Входные данные 65](#_Toc116553013)

[Выходные данные 65](#_Toc116553014)

[Примеры 66](#_Toc116553015)

[Варіант 21 66](#_Toc116553016)

[Входные данные 66](#_Toc116553017)

[Выходные данные 67](#_Toc116553018)

[Пример 67](#_Toc116553019)

[Варіант 22 67](#_Toc116553020)

[Входные данные 68](#_Toc116553021)

[Выходные данные 68](#_Toc116553022)

[Примеры 68](#_Toc116553023)

[Варіант 23 69](#_Toc116553024)

[Входные данные 69](#_Toc116553025)

[Выходные данные 69](#_Toc116553026)

[Примеры 69](#_Toc116553027)

[Варіант 24 70](#_Toc116553028)

[Входные данные 70](#_Toc116553029)

[Выходные данные 71](#_Toc116553030)

[Примеры 71](#_Toc116553031)

[Варіант 25 71](#_Toc116553032)

[Входные данные 71](#_Toc116553033)

[Выходные данные 71](#_Toc116553034)

[Пример 71](#_Toc116553035)

[Варіант 26 72](#_Toc116553036)

[Входные данные 72](#_Toc116553037)

[Выходные данные 72](#_Toc116553038)

[Пример 72](#_Toc116553039)

[Варіант 27 73](#_Toc116553040)

[Входные данные 73](#_Toc116553041)

[Выходные данные 73](#_Toc116553042)

[Примеры 74](#_Toc116553043)

[Варіант 28 74](#_Toc116553044)

[Входные данные 74](#_Toc116553045)

[Выходные данные 74](#_Toc116553046)

[Пример 75](#_Toc116553047)

[Варіант 29 75](#_Toc116553048)

[Входные данные 75](#_Toc116553049)

[Выходные данные 76](#_Toc116553050)

[Примеры 76](#_Toc116553051)

[Варіант 30 76](#_Toc116553052)

[Входные данные 77](#_Toc116553053)

[Выходные данные 77](#_Toc116553054)

[Пример 77](#_Toc116553055)

[Варіант 31 77](#_Toc116553056)

[Входные данные 78](#_Toc116553057)

[Выходные данные 78](#_Toc116553058)

[Пример 78](#_Toc116553059)

[Варіант 32 78](#_Toc116553060)

[Входные данные 79](#_Toc116553061)

[Выходные данные 79](#_Toc116553062)

[Примеры 80](#_Toc116553063)

[Варіант 33 80](#_Toc116553064)

[Входные данные 80](#_Toc116553065)

[Выходные данные 81](#_Toc116553066)

[Пример 81](#_Toc116553067)

[Варіант 34 81](#_Toc116553068)

[Входные данные 82](#_Toc116553069)

[Выходные данные 82](#_Toc116553070)

[Примеры 82](#_Toc116553071)

[Варіант 35 82](#_Toc116553072)

[Входные данные 83](#_Toc116553073)

[Выходные данные 83](#_Toc116553074)

[Пример 83](#_Toc116553075)

[Варіант 36 83](#_Toc116553076)

[Входные данные 83](#_Toc116553077)

[Выходные данные 84](#_Toc116553078)

[Пример 84](#_Toc116553079)

[Пояснение к примеру 84](#_Toc116553080)

[Варіант 37 84](#_Toc116553081)

[Входные данные 85](#_Toc116553082)

[Выходные данные 85](#_Toc116553083)

[Примеры 85](#_Toc116553084)

[Варіант 38 86](#_Toc116553085)

[Входные данные 86](#_Toc116553086)

[Выходные данные 86](#_Toc116553087)

[Примеры 87](#_Toc116553088)

[Варіант 39 87](#_Toc116553089)

[Входные данные 88](#_Toc116553090)

[Выходные данные 88](#_Toc116553091)

[Примеры 88](#_Toc116553092)

[Варіант 40 89](#_Toc116553093)

[Входные данные 89](#_Toc116553094)

[Выходные данные 89](#_Toc116553095)

[Примеры 90](#_Toc116553096)

[Варіант 41 90](#_Toc116553097)

[Входные данные 90](#_Toc116553098)

[Выходные данные 90](#_Toc116553099)

[Примеры 91](#_Toc116553100)

[Варіант 42 91](#_Toc116553101)

[Входные данные 91](#_Toc116553102)

[Выходные данные 91](#_Toc116553103)

[Пример 91](#_Toc116553104)

[Варіант 43 92](#_Toc116553105)

[Входные данные 92](#_Toc116553106)

[Выходные данные 92](#_Toc116553107)

[Пример 92](#_Toc116553108)

[Варіант 44 92](#_Toc116553109)

[Входные данные 93](#_Toc116553110)

[Выходные данные 93](#_Toc116553111)

[Пример 93](#_Toc116553112)

[Варіант 45 93](#_Toc116553113)

[Входные данные 93](#_Toc116553114)

[Выходные данные 94](#_Toc116553115)

[Примеры 94](#_Toc116553116)

[Варіант 46 94](#_Toc116553117)

[Варіант 47 95](#_Toc116553118)

[Варіант 48 96](#_Toc116553119)

[Варіант 49 96](#_Toc116553120)

[Варіант 50 97](#_Toc116553121)

[Варіант 51 98](#_Toc116553122)

[Варіант 52 99](#_Toc116553123)

[Варіант 53 101](#_Toc116553124)

[Варіант 54 102](#_Toc116553125)

[Варіант 55 103](#_Toc116553126)

[Варіант 56 103](#_Toc116553127)

[Варіант 57 104](#_Toc116553128)

[Варіант 58 106](#_Toc116553129)

[Варіант 59 107](#_Toc116553130)

[Варіант 60 108](#_Toc116553131)

[Варіант 61 110](#_Toc116553132)

[Варіант 62 111](#_Toc116553133)

[Варіант 63 112](#_Toc116553134)

[Варіант 64 113](#_Toc116553135)

[Варіант 65 114](#_Toc116553136)

[Варіант 66 115](#_Toc116553137)

[Варіант 67 116](#_Toc116553138)

[Варіант 68 118](#_Toc116553139)

[Варіант 69 120](#_Toc116553140)

[Варіант 70 120](#_Toc116553141)

[Варіант 71 121](#_Toc116553142)

[Варіант 72 122](#_Toc116553143)

[Варіант 73 123](#_Toc116553144)

[Варіант 74 125](#_Toc116553145)

[Варіант 75 127](#_Toc116553146)

[Варіант 76 128](#_Toc116553147)

[Варіант 77 128](#_Toc116553148)

[Варіант 78 130](#_Toc116553149)

[Варіант 79 131](#_Toc116553150)

[Варіант 80 132](#_Toc116553151)

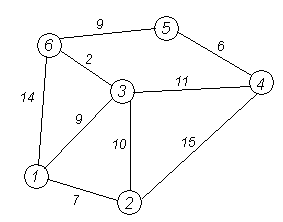
[Варіант 81 133](#_Toc116553152)

[Варіант 82 134](#_Toc116553153)

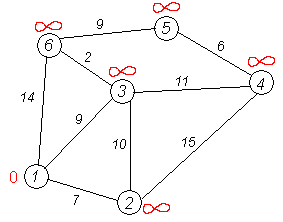
[Варіант 83 135](#_Toc116553154)

# Теоретична частина

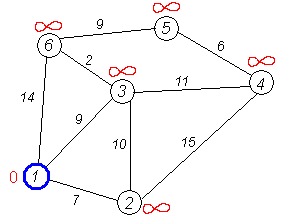
## Алгоритм Дейкстри – Інтуітивно

Зберігатимемо поточну мінімальну відстань до всіх вершин **V** (від даної вершини **a**) і на кожному кроці алгоритму намагатимемося зменшити цю відстань. Спочатку встановимо відстані до всіх вершин рівними нескінченості, а до вершини **а** — нулю. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph0.PNG) Розглянемо виконання алгоритму на прикладі. Хай потрібно знайти відстані від 1-ї вершини до всіх інших. Кружечками позначені вершини, лініями — шляхи між ними («дуги»). Над дугами позначена їх «ціна» — довжина шляху. Надписом над кружечком позначена поточна найкоротша відстань до вершини.

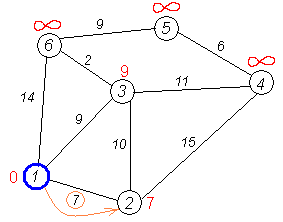
### Крок 1

Ініціалізація. Відстань до всіх вершин [графу](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Граф (математика)) **V** = {\displaystyle \infty }. Відстань до **а** = 0. Жодної вершини графу ще не опрацьовано.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph1.PNG)

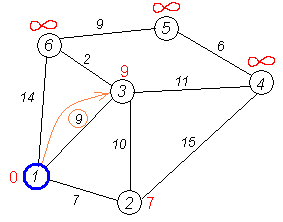
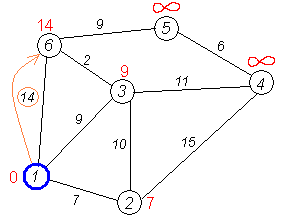
### Крок 2

Знаходимо таку вершину (із ще не опрацьованих), поточна найкоротша відстань до якої мінімальна. В нашому випадку це вершина 1. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через 1 менший за поточний мінімальний шлях в цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph2.PNG)

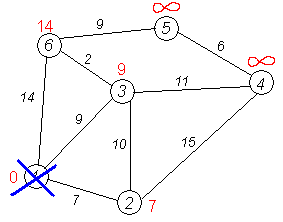
### Крок 3

Перший по порядку сусід 1-ї вершини — 2-а вершина. Шлях до неї через 1-у вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина дуги між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини, тому найкоротший шлях до 2-ї вершини дорівнює 7.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph3.PNG)

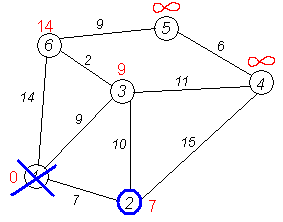
### Кроки 4, 5

Аналогічну операцію проробляємо з двома іншими сусідами 1-ї вершини — 3-ю та 6-ю.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph4.PNG) [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph5.PNG)

### Крок 6

Всі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів [Дейкстра](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0_%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80" \o "Дейкстра Едсгер)). Тому викреслимо її з [графу](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Граф (математика)), щоб відмітити цей факт. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph6.PNG)

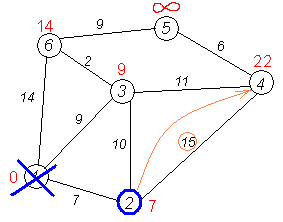
### Крок 7

Практично відбувається повернення до кроку 2. Знову знаходимо «найближчу» необроблену (невикреслену) вершину. Це вершина 2 з поточною найкоротшою відстанню до неї = 7. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph7.PNG) І знову намагаємося зменшити відстань до всіх сусідів 2-ї вершини, намагаючись пройти в них через 2-у. Сусідами 2-ї вершини є 1, 3, 4.

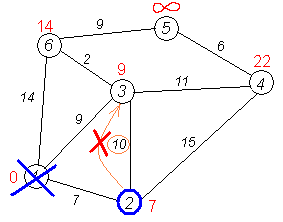
### Крок 8

Перший (по порядку) сусід вершини № 2 — 1-ша вершина. Але вона вже оброблена (або викреслена — див. крок 6). Тому з 1-ю вершиною нічого не робимо.

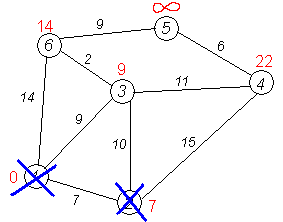
### Крок 8 (з іншими вхідними даними)

Інший сусід вершини 2 — вершина 4. Якщо йти в неї через 2-у, то шлях буде = найкоротша відстань до 2-ї + відстань між 2-ю і 4-ю вершинами = 7 + 15 = 22. Оскільки 22 < ∞, встановлюємо відстань до вершини № 4 рівним 22. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph8.PNG)

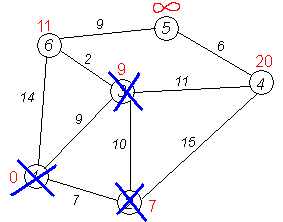
### Крок 9

Ще один сусід вершини 2 — вершина 3. Якщо йти в неї через 2-у, то шлях буде = 7 + 10 = 17. Але 17 більше за відстань, що вже запам'ятали раніше до вершини № 3 і дорівнює 9, тому поточну відстань до 3-ї вершини не міняємо. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph9.PNG)

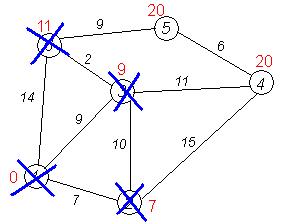
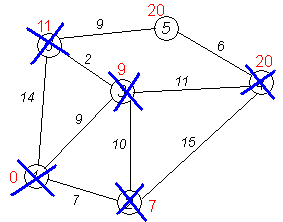
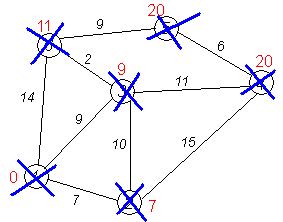
### Крок 10

Всі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і викреслюємо її з [графу](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Граф (математика)). [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph10.PNG)

### Кроки 11 — 15

По вже «відпрацьованій» схемі повторюємо кроки 2 — 6. Тепер «найближчою» виявляється вершина № 3. Після її «обробки» отримаємо такі результати:  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph11.PNG)

### Наступні кроки

Проробляємо те саме з вершинами, що залишилися (№ по порядку: 6, 4 і 5).  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph12.PNG) [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph13.PNG) [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph14.PNG)

### Завершення виконання алгоритму

Алгоритм закінчує роботу, коли викреслені всі вершини. Результат його роботи видно на останньому малюнку: найкоротший шлях від 1-ї вершини до 2-ї становить 7, до 3-ї — 9, до 4-ї — 20, до 5-ї — 20, до 6-ї — 11 умовних одиниць

## Пошук з поверненням

### **Огляд**

Класичний приклад використання пошуку з вертанням — це [задача про вісім ферзів](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BF%D1%80%D0%BE_%D0%B2%D1%96%D1%81%D1%96%D0%BC_%D1%84%D0%B5%D1%80%D0%B7%D1%96%D0%B2" \o "Задача про вісім ферзів), в який потрібно знайти розташування восьми ферзів на стандартній [шахівниці](https://uk.wikipedia.org/wiki/%D0%A8%D0%B0%D1%85%D0%B8" \o "Шахи) таким чином, щоб жоднен ферзь не атакував іншого. В звичайному підході пошуку з вертанням, неповні кандидати це розташування *k* ферзів в перших *k* рядах дошки, всі в різних рядах і стовпцях. Будь-який неповний розв'язок, що містить два ферзі, які атакують один одного має бути відкинутий, бо він не може бути доведеним до повного правильного розв'язку.

Пошук з вертанням може бути застосований тільки до задач, що дозволяють поняття «неповних кандидатів на розв'язок» і порівняно швидку перевірку на можливість доповнення кандидата до вірного розв'язку. Це корисно, наприклад, для знаходження значення у невпорядкованій таблиці. За можливості застосування, пошук з вертанням часто значно швидший ніж [повний перебір](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%BD%D0%B8%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D1%96%D1%80" \o "Повний перебір) всіх можливих кандидатів, через виключення великої кількості кандидатів однією перевіркою.

Пошук з вертанням — це важливе знаряддя для розв'язання проблеми відповідності обмеженням, таких як [кросворди](https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D0%B2%D0%BE%D1%80%D0%B4" \o "Кросворд), судоку та багато інших головоломок. Часто це найзручніший (якщо не найефективніший) підхід для [розбору](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7" \o "Синтаксичний аналіз), для [задачі пакування рюкзака](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BF%D0%B0%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D1%80%D1%8E%D0%BA%D0%B7%D0%B0%D0%BA%D0%B0" \o "Задача пакування рюкзака) та інших задач [комбінаторної оптимізації](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D1%96%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0_%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F" \o "Комбінаторна оптимізація). Це також базис для так званих мов [логічного програмування](https://uk.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Логічне програмування) таких як Icon, Planner і [Prolog](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BB%D0%BE%D0%B3_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Пролог (мова програмування)). Пошук з вертанням також використовується в рушії змін (diff) для програмного забезпечення [MediaWiki](https://uk.wikipedia.org/wiki/MediaWiki" \o "MediaWiki).

Пошук з вертанням покладається на подані користувачем процедури, які визначають задачу для розв'язання, природу неповних кандидатів і як вони доповнюються до повних кандидатів. Тому це швидше [метаевристика](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%B5%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0" \o "Метаевристика) ніж конкретний алгоритм — хоча, на відміну від інших метаевристик, вона гарантовано знаходить всі розв'язки скінченної проблеми за обмежений час.

Англомовний термін «backtrack» був винайдений американським математиком D. H. Lehmer в 1950-х. Піонерська мова опрацювання рядків [SNOBOL](https://uk.wikipedia.org/wiki/SNOBOL" \o "SNOBOL) (1962) можливо першою забезпечила вбудовані засоби загального пошуку з вертанням.

### **Опис методу**

Алгоритм пошуку з вертанням перераховує множину *неповних кандидатів* що, в принципі, можуть бути *доповнені* кількома шляхами для отримання всіх можливих розв'язків даної задачі. Доповнення будується покроково, послідовністю *кроків розширення кандидата.*

Концептуально, неповні кандидати є вузлами [деревоподібної структури](https://uk.wikipedia.org/w/index.php?title=%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%D0%BF%D0%BE%D0%B4%D1%96%D0%B1%D0%BD%D0%B0_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0&action=edit&redlink=1" \o "Деревоподібна структура (ще не написана)), *потенційне дерево пошуку.* Кожний неповний кандидат є батьком кандидатів відмінних від нього на один крок розширення; листям дерева є кандидати які не можуть бути розширені далі.

Алгоритм пошуку з вертанням обходить це дерево пошуку [рекурсивно](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D1%96%D1%8F" \o "Рекурсія), від кореня донизу, [пошуком в глибину](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D0%B2_%D0%B3%D0%BB%D0%B8%D0%B1%D0%B8%D0%BD%D1%83" \o "Пошук в глибину). В кожному вузлі *c*, алгоритм перевіряє чи може *c* бути доповнене до вірного розв'язку. Якщо ні, ціле піддерево з коренем в *c* пропускається (*обрізається*). Інакше, алгоритм (1) перевіряє чи є *c* вірним розв'язком, і якщо так повідомляє про це користувача; і (2) рекурсивно обходить усі піддерева *c*. Обидва тести і дочірні вузли кожного вузла визначаються за допомогою поданих користувачем процедур.

Внаслідок цього, *актуальне дерево пошуку* яке обходить алгоритм становить лише частину потенційного дерева. Загальна ціна алгоритму це кількість вузлів актуального дерева помножена на вартість отримання і обробки кожного вузла. Цей факт має бути врахованим коли обирається потенційне дерево і реалізується тест на обрізання.

### **Псевдокод**

Для застосування пошуку з вертанням для певного класу задач, потрібно мати дані *P* для конкреної задачі, що має бути розв'язана, і шість [підпрограм](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0" \o "Підпрограма), *корінь*, *відмова*, *прийняття*, *перший*, *наступний*, і *вихід*. Ці підпрограми мають отримувати дані *P* як параметр і мають бути такими:

1. *корінь(«P»)*: повертає неповний кандидат в корені дерева пошуку.
2. *відмова*(*P*,*c*): повертає *істину* тільки якщо неповний кандидат *c* невартий завершення.
3. *прийняття*(*P*,*c*): повертає *істину* якщо *c* є розв'язком *P*, і *хибу* якщо ні.
4. *перший*(*P*,*c*): продукує перше розширення кандидата *c*.
5. *наступний*(*P*,*s*): продукує інше наступне розширення кандидата, після розширення *s*.
6. *вихід*(*P*,*c*): прийняти розв'язок *c* з *P*, як підходяще для застосування.

Пошук з вертанням зводиться до виклику *bt*(*root*(*P*)), де *bt* наступна рекурсивна підпрограма:

**procedure** *bt*(*c*)

**якщо** *відмова*(*P*,*c*) **тоді** **вийти**

**якщо** *прийняття*(*P*,*c*) **тоді** *вихід*(*P*,*c*)

*s* ← *перший*(*P*,*c*)

**доки** *s* ≠ Λ **робити**

*bt*(*s*)

*s* ← *наступний*(*P*,*s*)

## Алгоритм Флойда — Воршелла

Но подумайте вот над чем. Вы решили сходить в кафе. Вам необходимо определить, как туда добраться, и вот вы просчитываете остановки общественного транспорта или изучаете улицы и переулки, по которым ехать на автомобиле. Вы прокладываете маршруты и выбираете из них самый короткий. Или вот, вы идете на работу, и решаете срезать через переулок, ведь так «быстрее». Но откуда взялось это «быстрее»? Не придавая этому особого значения, вы проложили маршруты и выбрали из них самый короткий: пролегающий через переулок.

В приведенных примерах «короткость» пути определяется или преодолеваемым расстоянием, или затрачиваемым интервалом времени. В примере со столом «короткость» пути может определяться количеством (или сложностью) необходимых действий: открыть ящик стола, открыть папку, достать лист бумаги – против взять лист из стопки на столе.

Если внимательно посмотреть, то каждый из приведенных примеров описывает поиск кратчайшего пути из одной исходной точки. Такой класс задач называется **SSSP (Single Source Shortest Path)** и базовым алгоритмом решения таких задач, является [алгоритм Дейкстры](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B), имеющий квадратичную сложность. Именно такие задачи мы решаем чаще всего.

Однако иногда нам необходимо знать больше, чем один или несколько кратчайших путей, пролегающих из одной точки. Представьте. Вы решили составить карту путей между работой, домом и театром. В этом случае вам необходимо проложить 6 маршрутов:

1. Работа — Дом
2. Дом — Работа (пути могут отличаться, например из-за наличия улиц с односторонним движением)
3. Работа — Театр
4. Театр — Работа
5. Дом — Театр
6. Театр — Дом

С увеличением интересующих нас мест количество пар маршрутов, будет стремительно возрастать, согласно формулам [комбинаторики](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%B8%D0%BA%D0%B0), если точнее, то формуле [размещения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%BC%D0%B5%D1%89%D0%B5%D0%BD%D0%B8%D0%B5):

A(k, n) = n! / (n - m)!

// где n - количество элементов,

//     k - количество элементов в размещение (в нашем случае k = 2)

Не трудно посчитать, что между 4-я местами существует 12 маршрутов, а между 10-ю их уже 90. И все это не учитывая факта, что между интересующими нас путями может не быть прямого пути, а это привносит в нашу схему промежуточные места (напр. остановки, перекрестки). Класс задач, описывающих поиск всех кратчайших путей между всеми вершинами графа (каждое место представляет собой вершину графа) называется **APSP (All Pairs Shortest Paths)** и базовым алгоритмом решения таких задач является [алгоритм Флойда-Уоршелла](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A4%D0%BB%D0%BE%D0%B9%D0%B4%D0%B0_%E2%80%94_%D0%A3%D0%BE%D1%80%D1%88%D0%B5%D0%BB%D0%BB%D0%B0), которые мы и рассмотрим.

Алгоритм Флойда-Уоршелла нахождения всех кратчайших путей между всеми вершинами графа был представлен Робертом Флойдом в работе [1] (см. секцию «Ссылки» в самом конце). В том же году, Питер Ингерман в работе [2] представил современную, программную формулировку алгоритма в виде трех вложенных циклов for, приблизительный псевдокод которой приведен ниже:

algorithm FloydWarshall**(**W**)** **do**

**for** k = 0 to N - 1 **do**

**for** i = 0 to N - 1 **do**

**for** j = 0 to N - 1 **do**

W**[**i,j**]** = min**(**W**[**i,j**]**, W**[**i,k**]** + W**[**k,j**])**

**end** **for**

**end** **for**

**end** **for**

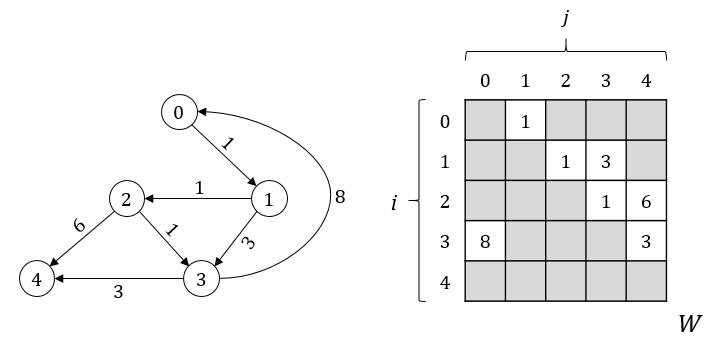
**end** algorithm

// где W - матрица весов размером N x N,

// min() - функция возвращающая минимальный из переданных аргументов.

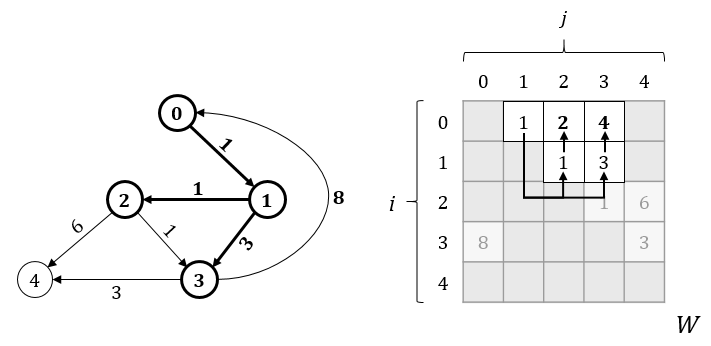
Если вам никогда не приходилось работать с графами, представленными в виде матриц, то приведенный псевдокод и реализованный им алгоритм, может показаться весьма запутанным. Так что вначале мы попробуем разобраться с тем, как можно представить граф в виде матрицы и как это представление можно использовать для нахождения кратчайших путей.

На рисунке ниже изображен [взвешенный, ориентированный граф](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) из 5-и вершин. Граф изображен как графически (слева), так и в виде матрицы весов (справа). Матрица весов представляет собой вариант [матрицы смежности](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8), где каждая ячейка матрицы содержит «вес» — расстояние из вершины i (номер строки) в вершину j (номер столбца). Важно отметить, что матрица весов не содержит «путь» (список вершин, через которые пролегает путь между вершинами) — только расстояние между ними.

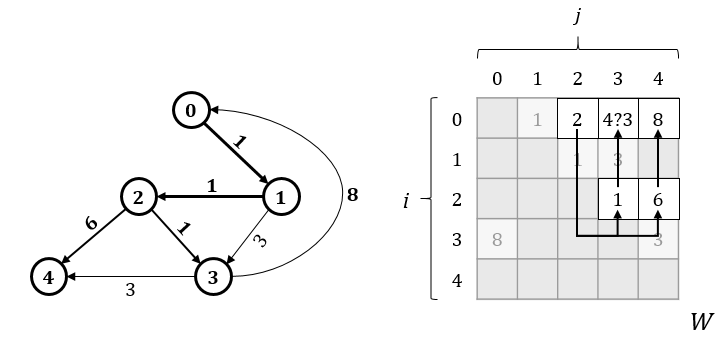


На рисунке все ячейки матрицы W содержат значения, равные весам ребер между [смежными вершинами](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8) графа. Поэтому, если изучить пути из вершины 0 (строка 0), то окажется, что их … нет. Однако, на графическом представлении графа отчетливо видно наличие таких путей, правда, проходящих через другие вершины графа — из вершины 0, через вершину 1, можно попасть в вершины 2 и 3. Причина этого отличия в том, что в изначальном состоянии, матрица весов содержит исключительно информацию о расстояниях между смежными вершинами графа.

Однако представленной в матрице информации на самом деле достаточно, чтобы **обнаружить** все остальные пути. Обратите внимание на ячейку W[0,1]. Значение в этой ячейке говорит о существовании пути 0→1:1 (из вершины 0 в вершину 1, с весом 1). Наличие этого пути означает, что мы можем просканировать все ячейки строки 1 матрицы весов (в которой содержатся все пути из вершины 1) и добавить эту информацию в строку 0, предварительно увеличив их на вес записанный в ячейке W[0,1] (на расстояние из вершины 0 в вершину 1):



Точно таким же образом, мы можем найти пути из вершины 0 в остальные вершины в графе. Во время поиска может оказаться, что в одну и ту же вершину, можно попасть разными путями и что самое главное — веса этих путей могут различаться. Пример подобной ситуации проиллюстрирован на рисунке ниже, на котором изображен поиск путей из вершины 0, через вершину 2, в остальные вершины графа.



На рисунке отчетливо видно, что известный нам путь 0→3 имеет вес 4 (помним, что матрица весов не содержит «самого пути» т.е. мы не знаем, через какие вершины проходит этот путь), в то время, как новый путь 0→2→3 имеет вес 3. В этот момент мы совершаем выбор и записываем в ячейку W[0,3] меньшее значение — 3.

Кажется, мы только что нашли кратчайший путь из вершины 0 в вершину 3, ну или хотя бы один из них

Именно так и работает алгоритм Флойда-Уоршелла. Посмотрите на псевдокод алгоритма еще раз:

algorithm FloydWarshall**(**W**)** **do**

**for** k = 0 to N - 1 **do**

**for** i = 0 to N - 1 **do**

**for** j = 0 to N - 1 **do**

W**[**i,j**]** = min**(**W**[**i,j**]**, W**[**i,k**]** + W**[**k,j**])**

**end** **for**

**end** **for**

**end** **for**

**end** algorithm

Внешний цикл for по k обходит все вершины графа. k представляет собой текущую вершину, через которую мы выполняем поиск. Внутренний цикл for по i обходит все вершины графа, из которых мы выполняем поиск путей, проходящих через вершину k. И наконец внутренний цикл for по j обходит все вершины графа, в которые мы выполняем поиск из вершины i через вершину k. И наконец, в теле цикла, мы сравниваем вес известного нам пути i→j записанного в матрице весов с весом пути i→k→j, записывая в ячейку минимальное из значений.

Теперь, когда мы разобрались в механике алгоритма, пришло время его реализации.

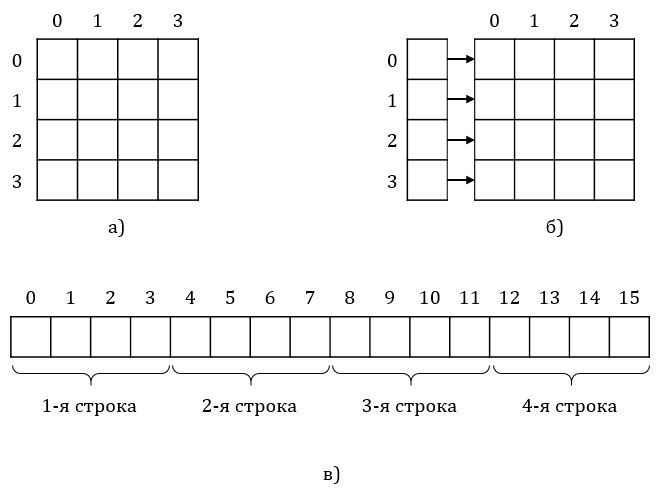
**Базовая реализация алгоритма Флойда-Уоршелла**

Прежде чем приступить к реализации первой (а их будет несколько) версии алгоритма, необходимо прояснить несколько технических моментов:

1. Все приведенные реализации алгоритма работают с матрицей весов, представленной в виде массива.
2. Все приведенные реализации алгоритма используют целочисленную арифметику. Отсутствие пути между вершинами представляется в матрице константой NO\_EDGE = (int.MaxValue / 2) - 1.

Теперь по прядку. Первый пункт.

Когда мы говорим о матрицах, мы оперируем такими понятиями как «строка» и «столбец». Из-за этого нам естественно думать о матрице как о «квадрате» или «прямоугольнике», как показано в подпункте а) на рисунке ниже.



Однако это вовсе не означает, что в программной реализации мы должны иметь «массив из массивов», как показано в подпункте б). Вместо этого мы можем представить матрицу в виде массива, как показано в подпункте в), где индекс каждой ячейки матрицы вычисляется по формуле:

i = row **\*** row\_size + col;

// где row - индекс строки, в которой расположена ячейка,

// col - индекс столбца, в котором расположена ячейка,

// row\_size - количество ячеек матрицы в одной строке.

Представление матрицы в виде массива является **необходимым предусловием** для эффективного выполнения алгоритма Флойда-Уоршелла. И обусловлена она тем, что такое представление повышает [локализацию доступа к данным](https://en.wikipedia.org/wiki/Locality_of_reference). Тема эффективного доступа к памяти невероятно объемная и является объектом многих научных исследований в области разработки алгоритмов. Её просто невозможно описать в нескольких абзацах, поэтому в этот раз я попрошу вас поверить мне на слово (а перед этим, разумеется, изучить тему самостоятельно: не стоит доверять всяким незнакомцам в интернете).

Насчет второго пункта.

Если внимательно посмотреть, на псевдокод алгоритма, можно заметить, что там нигде нет проверки, что путь между двумя вершинами существует. Вместо этого в псевдокоде используется функция min(), возвращающая минимальное значение. Причина в том, что если между двумя вершинами в графе нет пути, то ячейка матрицы весов должна содержать в себе значение бесконечности (а любое число, не являющееся бесконечностью, меньше бесконечности — если это конечно не JavaScript, там возможно все). При использовании целых чисел может появиться соблазн использовать максимальное допустимое значение — int.MaxValue. Однако в этом случае, мы столкнемся с переполнением, когда веса путей i→k и k→j будут равны int.MaxValue (при использовании константы (int.MaxValue / 2) - 1 переполнения не будет).

«Окей, но ведь можно просто предварительно проверять, существует ли пути i→k и k→j, например сравнивая их с 0 перед выполнением».

Это действительно возможно, однако к несчастью, результат такого рода сравнений со стороны центрального процессора является случайным и приведет к очень высокому показателю [branch misprediction](https://en.wikipedia.org/wiki/Branch_predictor) и как следствие — к значительной потере производительности. Поэтому, как и в предыдущий раз, в этом вопросе я попрошу вас поверить мне на слово.

Уф, с теорией закончили — теперь к реализации:

public **void** FloydWarshall\_00**(**int**[]** matrix, int sz**)**

**{**

**for** **(**var k = 0; k **<** sz; ++k**)**

**{**

**for** **(**var i = 0; i **<** sz; ++i**)**

**{**

**for** **(**var j = 0; j **<** sz; ++j**)**

**{**

var distance = matrix**[**i **\*** sz + k**]** + matrix**[**k **\*** sz + j**]**;

**if** **(**matrix**[**i **\*** sz + j**]** **>** distance**)**

**{**

matrix**[**i **\*** sz + j**]** = distance;

**}**

**}**

**}**

**}**

**}**

Приведенная реализация представляет собой практически полное отображение приведенного в самом начале псевдокода, за тем исключением, что вместо использования функции Math.Min() мы используем проверку if, обновляя значение в ячейке матрицы только при необходимости.

Пришло время запустить код на выполнение! Посмотрим, насколько быстро он работает!

В качестве платформы для запуска я буду использовать [Benchmark.NET](https://github.com/dotnet/BenchmarkDotNet) (версия 0.12.1). Экспериментальные графы представляют собой [ориентированные, ациклические](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) графы (размером 300, 600, 1200, 2400 и 4800 вершин). Количество ребер в графах приближается к 80% от возможного (максимальное количество ребер в ориентированном, ациклическом графе вычисляется по формуле: (v \* (v - 1)) / 2, где v — это количество вершин в графе).

Поехали!

Результаты выполнения на моем компьютере (ОС Windows 10.0.19042, процессор Intel Core i7-7700 CPU 3.60GHz (Kaby Lake) / 8 логических процессоров / 4 физических ядра) приведены в таблице ниже:

| **Method** | **Size** | **Mean** | **Error** | **StdDev** |
| --- | --- | --- | --- | --- |
| FloydWarshall\_00 | 300 | 27.525 ms | 0.1937 ms | 0.1617 ms |
| FloydWarshall\_00 | 600 | 217.897 ms | 1.6415 ms | 1.5355 ms |
| FloydWarshall\_00 | 1200 | 1,763.335 ms | 7.4561 ms | 6.2262 ms |
| FloydWarshall\_00 | 2400 | 14,533.335 ms | 63.3518 ms | 52.9016 ms |
| FloydWarshall\_00 | 4800 | 119,768.219 ms | 181.5514 ms | 160.9406 ms |

Из полученных результатов примечательно, что время выполнения невероятно возрастает с увеличением размера графа — так поиск всех кратчайших путей между всеми вершинами в графа из 300 вершин занимает 27 миллисекунд, в графе из 2400 вершин — 14.5 секунд, а в графе из 4800 — целых 119 секунд! Почти 2 минуты!

Смотря на реализацию алгоритма, трудно представить, что мы можем что-то изменить и сократить время выполнения алгоритма. Там ведь всего ТРИ цикла!

Однако, как это очень часто бывает, возможности скрываются в деталях

**Знай свои данные — не плотные графы**

В базовой формулировке алгоритм Флойда-Уоршелла является наиболее эффективным при использовании [плотных](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) или [полных](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) графов т. к. по своей механике во время выполнения алгоритм обходит каждую вершину графа через каждую вершину. Однако, используемые нами графы, хоть и являются «достаточно плотными» (помним про 80%), в первую очередь являются **ациклическими**. Простыми словами это означает, что если из вершины 1 есть путь в вершину 2, то из вершины 2 не существует пути в вершину 1. А это в свою очередь означает, наличие значительного количества вершин между которыми нет и не может быть пути. Мы можем это использовать и не проверять наличие путей из вершины i в вершины j, если отсутствует путь i→k:

public **void** FloydWarshall\_01**(**int**[]** matrix, int sz**)**

**{**

**for** **(**var k = 0; k **<** sz; ++k**)**

**{**

**for** **(**var i = 0; i **<** sz; ++i**)**

**{**

**if** **(**matrix**[**i **\*** sz + k**]** == NO\_EDGE**)**

**{**

continue;

**}**

**for** **(**var j = 0; j **<** sz; ++j**)**

**{**

var distance = matrix**[**i **\*** sz + k**]** + matrix**[**k **\*** sz + j**]**;

**if** **(**matrix**[**i **\*** sz + j**]** **>** distance**)**

**{**

matrix**[**i **\*** sz + j**]** = distance;

**}**

**}**

**}**

**}**

**}**

Результаты выполнения предыдущей (00) обновленной (01) реализации приведены в таблице ниже:

| **Method** | **Size** | **Mean** | **Error** | **StdDev** | **Ratio** |
| --- | --- | --- | --- | --- | --- |
| FloydWarshall\_00 | 300 | 27.525 ms | 0.1937 ms | 0.1617 ms | 1.00 |
| FloydWarshall\_01 | 300 | 12.399 ms | 0.0943 ms | 0.0882 ms | 0.45 |
| FloydWarshall\_00 | 600 | 217.897 ms | 1.6415 ms | 1.5355 ms | 1.00 |
| FloydWarshall\_01 | 600 | 99.122 ms | 0.8230 ms | 0.7698 ms | 0.45 |
| FloydWarshall\_00 | 1200 | 1,763.335 ms | 7.4561 ms | 6.2262 ms | 1.00 |
| FloydWarshall\_01 | 1200 | 766.675 ms | 6.1147 ms | 5.7197 ms | 0.43 |
| FloydWarshall\_00 | 2400 | 14,533.335 ms | 63.3518 ms | 52.9016 ms | 1.00 |
| FloydWarshall\_01 | 2400 | 6,507.878 ms | 28.2317 ms | 26.4079 ms | 0.45 |
| FloydWarshall\_00 | 4800 | 119,768.219 ms | 181.5514 ms | 160.9406 ms | 1.00 |
| FloydWarshall\_01 | 4800 | 55,590.374 ms | 414.6051 ms | 387.8218 ms | 0.46 |

Выглядит впечатляюще!

Мы сократили время выполнения алгоритма более чем вдвое! Разумеется, чем плотнее граф, тем меньше будет выигрыш, однако, если вам заранее известен класс данных, с которыми предполагается работа, то использование такого рода оптимизаций может значительно сократить время выполнения алгоритма.

Но можно ли сделать что-нибудь еще?

**Знай свое оборудование — параллелизм**

Мой компьютер оборудован процессором Intel Core i7-7700 CPU 3.60GHz (Kaby Lake) с 8-ю логическими процессорами ([HW](https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)" \t "_blank)) или 4-я физическими ядрами с технологий [Hyper Threading](https://en.wikipedia.org/wiki/Hyper-threading). Наличие нескольких ядер означает, что мы имеем в своем распоряжении «свободные рабочие руки», которые можно (и нужно) занять полезной работой. Однако, прежде чем создавать потоки, нам необходимо понять — какую именно часть работы алгоритма мы можем разделить.

Циклы всегда представляют собой очевидных кандидатов на распараллеливание т.к. зачастую, работа выполняемая на каждой итерации цикла не зависит от других итераций. Поэтому давайте рассмотрим каждый из имеющихся циклов в алгоритме на предмет зависимостей.

Начнем с цикла по k. На каждой итерации цикла алгоритм рассчитывает пути из каждой вершины в каждую, через вершину k, записывая обнаруженные пути в матрицу весов. По отдельности каждая из итераций выглядит независимой. Например, не имеет значения в каком порядке выполняются итерации k-го цикла: 0, 1, 2, 3 или 3, 1, 2, 0. Однако, основным условием является то, что все эти итерации должны выполняться последовательно (одна за другой). Только тогда выполнится основное условие алгоритма: обход графа из каждой вершины в каждую вершину через каждую вершину.

Теперь рассмотрим цикл по i. На каждой итерации мы ищем пути из вершины i в вершину k и затем из вершины k в вершину j. Если обратиться к матрице весов, то можно заметить, что на каждой итерации цикла мы считываем значение с k-й строки матрицы и обновляем значения только в i-й строке матрицы. Это означает, что итерации цикла по i не только могут быть выполнены в любом порядке (как и итерации цикла по k), но и не используют результат работы друг друга, а значит, могут выполняться параллельно!

Кажется, мы что-то нашли — реализуем это!

*Цикл по j, так же как и цикл по i может быть распараллелен. Однако, распараллеливание вложенного цикла является крайне неэффективным.*

public **void** FloydWarshall\_02**(**int**[]** matrix, int sz**)**

**{**

**for** **(**var k = 0; k **<** sz; ++k**)**

**{**

Parallel.For**(**0, sz, i =**>**

**{**

**if** **(**matrix**[**i **\*** sz + k**]** == NO\_EDGE**)**

**{**

**return**;

**}**

**for** **(**var j = 0; j **<** sz; ++j**)**

**{**

var distance = matrix**[**i **\*** sz + k**]** + matrix**[**k **\*** sz + j**]**;

**if** **(**matrix**[**i **\*** sz + j**]** **>** distance**)**

**{**

matrix**[**i **\*** sz + j**]** = distance;

**}**

**}**

**})**;

**}**

**}**

В нашей реализации распараллеливание выполняется при помощи класса [Parallel](https://docs.microsoft.com/dotnet/api/system.threading.tasks.parallel?view=net-5.0). Результаты выполнения алгоритмов 01 и 02 приведены в таблице ниже (колонка Ratio показывает значение относительно алгоритма 00):

| **Method** | **Size** | **Mean** | **Error** | **StdDev** | **Ratio** |
| --- | --- | --- | --- | --- | --- |
| FloydWarshall\_01 | 300 | 12.399 ms | 0.0943 ms | 0.0882 ms | 0.45 |
| FloydWarshall\_02 | 300 | 6.252 ms | 0.0211 ms | 0.0187 ms | 0.23 |
| FloydWarshall\_01 | 600 | 99.122 ms | 0.8230 ms | 0.7698 ms | 0.45 |
| FloydWarshall\_02 | 600 | 35.825 ms | 0.1003 ms | 0.0837 ms | 0.16 |
| FloydWarshall\_01 | 1200 | 766.675 ms | 6.1147 ms | 5.7197 ms | 0.43 |
| FloydWarshall\_02 | 1200 | 248.801 ms | 0.6040 ms | 0.5043 ms | 0.14 |
| FloydWarshall\_01 | 2400 | 6,507.878 ms | 28.2317 ms | 26.4079 ms | 0.45 |
| FloydWarshall\_02 | 2400 | 2,076.403 ms | 20.8320 ms | 19.4863 ms | 0.14 |
| FloydWarshall\_01 | 4800 | 55,590.374 ms | 414.6051 ms | 387.8218 ms | 0.46 |
| FloydWarshall\_02 | 4800 | 15,614.506 ms | 115.6996 ms | 102.5647 ms | 0.13 |

Из полученных результатов отчетливо видно — распараллеливание сократило время выполнения алгоритма от 2-х до 4-х раз! Однако важно помнить, что выполнение параллельной реализации потребляет практически все доступные на компьютере ресурсы процессора, что может привести к их серьезному дефициту у других приложений в системе.

Возможно вы догадались — это еще не все

**Знай свою платформу — векторизация**

[Векторизацией](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)) называют процесс преобразования программного кода, выполняющего в каждый момент времени одну операцию над одним элементом, в программный код, выполняющий одну операцию, но над несколькими элементами одновременно.

Звучит сложнее, чем оно есть на самом деле, поэтому давайте разберемся на примере:

var a = **new** **[]** **{** 5, 7, 8, 1 **}**;

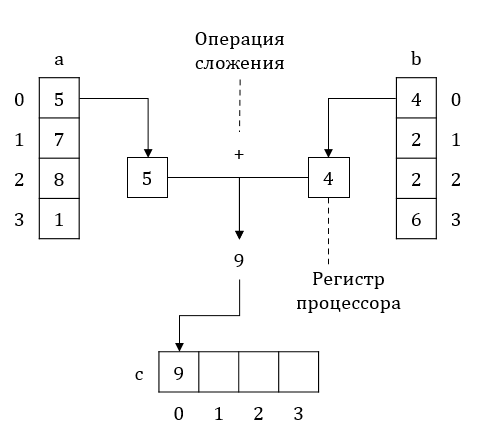
var b = **new** **[]** **{** 4, 2, 2, 6 **}**;

var c = **new** **[]** **{** 0, 0, 0, 0 **}**;

**for** **(**var i = 0; i **<** 4; ++i**)**

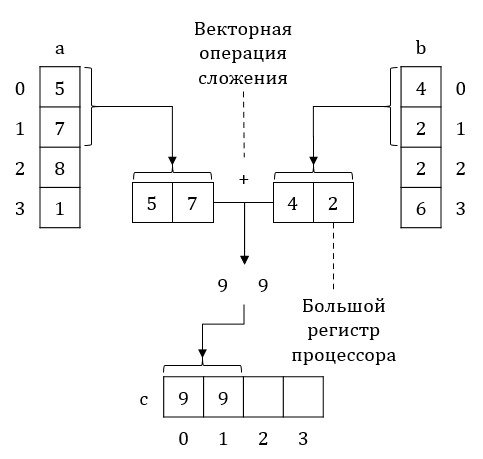
c**[**i**]** = a**[**i**]** + b**[**i**]**;

В сверх упрощённом виде итерация 0 цикла for выглядит следующим образом:



Первым шагом – значения элементов массивов a и b из памяти загружаются в регистры процессора (обратите внимание, что в один регистр помещается только один элемент массива). Как только значения оказались в регистрах, над ними выполняется операция сложения и ее результат записывается обратно в память в соответствующий элемент массива c. Подобным образом выполняются остальные итерации цикла, т. е. этот процесс повторяется 4 раза.

Использование векторизации означает использование специализированных регистров процессора (в которых размещаются несколько элементов сразу) и специализированных инструкций процессора позволяющий выполнять операцию одновременно над ними:



Таким образом (в приведенном примере) мы складываем элементы массива по два за одну операцию. Это означает, что теперь нам необходимо не 4-е, а всего 2-е итерации.

В .NET использование векторных инструкций и векторных регистров доступно через специальные типы — [Vector](https://docs.microsoft.com/dotnet/api/system.numerics.vector?view=net-5.0) и [Vector<T>](https://docs.microsoft.com/dotnet/api/system.numerics.vector-1?view=net-5.0) (а также, через типы из пространства имен [System.Runtime.Intrinsics](https://docs.microsoft.com/dotnet/api/system.runtime.intrinsics?view=net-5.0)). Реализация алгоритма с использованием типов Vector приведена ниже:

public **void** FloydWarshall\_03**(**int**[]** matrix, int sz**)**

**{**

**for** **(**var k = 0; k **<** sz; ++k**)**

**{**

**for** **(**var i = 0; i **<** sz; ++i**)**

**{**

**if** **(**matrix**[**i **\*** sz + k**]** == NO\_EDGE**)**

**{**

continue;

**}**

var ik\_vec = **new** Vector**<**int**>(**matrix**[**i **\*** sz + k**])**;

var j = 0;

**for** **(**; j **<** sz - Vector**<**int**>**.Count; j += Vector**<**int**>**.Count**)**

**{**

var ij\_vec = **new** Vector**<**int**>(**matrix, i **\*** sz + j**)**;

var ikj\_vec = **new** Vector**<**int**>(**matrix, k **\*** sz + j**)** + ik\_vec;

var lt\_vec = Vector.LessThan**(**ij\_vec, ikj\_vec**)**;

**if** **(**lt\_vec == **new** Vector**<**int**>(**-1**))**

**{**

continue;

**}**

var r\_vec = Vector.ConditionalSelect**(**lt\_vec, ij\_vec, ikj\_vec**)**;

r\_vec.CopyTo**(**matrix, i **\*** sz + j**)**;

**}**

**for** **(**; j **<** sz; ++j**)**

**{**

var distance = matrix**[**i **\*** sz + k**]** + matrix**[**k **\*** sz + j**]**;

**if** **(**matrix**[**i **\*** sz + j**]** **>** distance**)**

**{**

matrix**[**i **\*** sz + j**]** = distance;

**}**

**}**

**}**

**}**

**}**

Первым шагом var ik\_vec = new Vector<int>(matrix[i \* sz + k]) в векторизованной части программного кода мы создаем новый вектор, в котором каждый элемент равен весу пути i→k. В результате, если в векторе можно разместить 4-е значения типа int, а вес пути i→k равен 5, то созданный вектор будет иметь значение [5,5,5,5]. Затем, на каждой итерации мы одновременно рассматриваем пути из вершины i в вершины j,j + 1,...,j + Vector<int>.Count.

Использование Vector<int>.Count возвращает возможное количество элементов типа int размещаемых векторном регистре. В зависимости от модели процессора векторные регистры могут иметь разный размер.

Важное отличие векторной реализации в том, что мы работаем над несколькими элементами одновременно, и поэтому результат операции может быть не однозначен. Например, операция сравнения Vector.LessThan(ij\_vec,ikj\_vec) возвращает новый вектор, каждый элемент которого содержит результат сравнения соответствующих элементов из векторов ij\_vec и ikj\_vec (-1, если элемент из ij\_vec меньше элемента из ikj\_vec и 0, если наоборот). В свою очередь, операция Vector.ConditionalSelect(lt\_vec,ij\_vec,ikj\_vec) возвращает новый вектор, который содержит минимальные элементы из обоих векторов, т.е. если элемент вектора lt\_vec установлен в -1, то берется элемент из ij\_vec, иначе выбирается соответствующий элемент из ikj\_vec. И наконец, операция r\_vec.CopyTo(matrix,i \* sz + j) записывает значение вектора обратно в массив.

Разбиение реализации на 2 цикла (векторизованный и не векторизованный) необходимо из-за случаев, когда длина массива не кратна значению возвращаемому Vector<int>.Count. В этом случае мы не можем обработать все элементы массива, используя векторные инструкции, и нам нужно рассчитать оставшиеся пути, использовав обычные инструкции.

Результатом наших трудов стали следующие результаты (сравнение реализации 01, 02 и 03):

| **Method** | **Size** | **Mean** | **Error** | **StdDev** | **Ratio** |
| --- | --- | --- | --- | --- | --- |
| FloydWarshall\_01 | 300 | 12.399 ms | 0.0943 ms | 0.0882 ms | 0.45 |
| FloydWarshall\_02 | 300 | 6.252 ms | 0.0211 ms | 0.0187 ms | 0.23 |
| FloydWarshall\_03 | 300 | 3.056 ms | 0.0301 ms | 0.0281 ms | 0.11 |
| FloydWarshall\_01 | 600 | 99.122 ms | 0.8230 ms | 0.7698 ms | 0.45 |
| FloydWarshall\_02 | 600 | 35.825 ms | 0.1003 ms | 0.0837 ms | 0.16 |
| FloydWarshall\_03 | 600 | 24.378 ms | 0.4320 ms | 0.4041 ms | 0.11 |
| FloydWarshall\_01 | 1200 | 766.675 ms | 6.1147 ms | 5.7197 ms | 0.43 |
| FloydWarshall\_02 | 1200 | 248.801 ms | 0.6040 ms | 0.5043 ms | 0.14 |
| FloydWarshall\_03 | 1200 | 185.628 ms | 2.1240 ms | 1.9868 ms | 0.11 |
| FloydWarshall\_01 | 2400 | 6,507.878 ms | 28.2317 ms | 26.4079 ms | 0.45 |
| FloydWarshall\_02 | 2400 | 2,076.403 ms | 20.8320 ms | 19.4863 ms | 0.14 |
| FloydWarshall\_03 | 2400 | 2,568.676 ms | 31.7359 ms | 29.6858 ms | 0.18 |
| FloydWarshall\_01 | 4800 | 55,590.374 ms | 414.6051 ms | 387.8218 ms | 0.46 |
| FloydWarshall\_02 | 4800 | 15,614.506 ms | 115.6996 ms | 102.5647 ms | 0.13 |
| FloydWarshall\_03 | 4800 | 18,257.991 ms | 84.5978 ms | 79.1329 ms | 0.15 |

Из полученных результатов видно, что использование векторных инструкций в значительной степени сократило время выполнение алгоритма: от 3-х до 4-х раз по сравнению предыдущей не использующей параллелизм версией (01). Интересно, что векторизованная версия демонстрирует преимущество над распараллеленной версией на небольших размерах графов (до 2400 вершин).

И напоследок — совместим векторизацию и параллелизм!

*Если вас заинтересовал вопрос векторизации, то я рекомендую ознакомиться с циклом статей от*[*Dan Shechter*](https://bits.houmus.org/2020-01-28/this-goes-to-eleven-pt1)*, в которых он занимается векторизацией метода Array.Sort (результаты работы представленные в его статьях впоследствии нашли свое место в реализации сборщика мусора в*[*.NET 5*](https://github.com/dotnet/runtime/pull/37159)*).*

**Знаю свою платформу и свое оборудование — векторизация и параллелизм!**

Последняя реализация совмещает в себе использование параллелизма и векторизации и … она не отличается оригинальностью  По сути, мы просто вставляем тело цикла по i из реализации 03 в тело метода Parallel.For из реализации 02:

public **void** FloydWarshall\_04**(**int**[]** matrix, int sz**)**

**{**

**for** **(**var k = 0; k **<** sz; ++k**)**

**{**

Parallel.For**(**0, sz, i =**>**

**{**

**if** **(**matrix**[**i **\*** sz + k**]** == NO\_EDGE**)**

**{**

**return**;

**}**

var ik\_vec = **new** Vector**<**int**>(**matrix**[**i **\*** sz + k**])**;

var j = 0;

**for** **(**; j **<** sz - Vector**<**int**>**.Count; j += Vector**<**int**>**.Count**)**

**{**

var ij\_vec = **new** Vector**<**int**>(**matrix, i **\*** sz + j**)**;

var ikj\_vec = **new** Vector**<**int**>(**matrix, k **\*** sz + j**)** + ik\_vec;

var lt\_vec = Vector.LessThan**(**ij\_vec, ikj\_vec**)**;

**if** **(**lt\_vec == **new** Vector**<**int**>(**-1**))**

**{**

continue;

**}**

var r\_vec = Vector.ConditionalSelect**(**lt\_vec, ij\_vec, ikj\_vec**)**;

r\_vec.CopyTo**(**matrix, i **\*** sz + j**)**;

**}**

**for** **(**; j **<** sz; ++j**)**

**{**

var distance = matrix**[**i **\*** sz + k**]** + matrix**[**k **\*** sz + j**]**;

**if** **(**matrix**[**i **\*** sz + j**]** **>** distance**)**

**{**

matrix**[**i **\*** sz + j**]** = distance;

**}**

**}**

**})**;

**}**

**}**

Результаты выполнения всех рассмотренных нами реализаций алгоритма приведены в таблице ниже. Как и ожидалось, одновременное использование векторизации и параллелизма показывает наилучшие результаты на всех размерах графов, сокращая время выполнение до 25 раз (при размере графа в 1200 вершин) по сравнению с первой реализацией.

| **Method** | **Size** | **Mean** | **Error** | **StdDev** | **Ratio** |
| --- | --- | --- | --- | --- | --- |
| FloydWarshall\_00 | 300 | 27.525 ms | 0.1937 ms | 0.1617 ms | 1.00 |
| FloydWarshall\_01 | 300 | 12.399 ms | 0.0943 ms | 0.0882 ms | 0.45 |
| FloydWarshall\_02 | 300 | 6.252 ms | 0.0211 ms | 0.0187 ms | 0.23 |
| FloydWarshall\_03 | 300 | 3.056 ms | 0.0301 ms | 0.0281 ms | 0.11 |
| FloydWarshall\_00 | 600 | 217.897 ms | 1.6415 ms | 1.5355 ms | 1.00 |
| FloydWarshall\_01 | 600 | 99.122 ms | 0.8230 ms | 0.7698 ms | 0.45 |
| FloydWarshall\_02 | 600 | 35.825 ms | 0.1003 ms | 0.0837 ms | 0.16 |
| FloydWarshall\_03 | 600 | 24.378 ms | 0.4320 ms | 0.4041 ms | 0.11 |
| FloydWarshall\_00 | 1200 | 1,763.335 ms | 7.4561 ms | 6.2262 ms | 1.00 |
| FloydWarshall\_01 | 1200 | 766.675 ms | 6.1147 ms | 5.7197 ms | 0.43 |
| FloydWarshall\_02 | 1200 | 248.801 ms | 0.6040 ms | 0.5043 ms | 0.14 |
| FloydWarshall\_03 | 1200 | 185.628 ms | 2.1240 ms | 1.9868 ms | 0.11 |
| FloydWarshall\_00 | 2400 | 14,533.335 ms | 63.3518 ms | 52.9016 ms | 1.00 |
| FloydWarshall\_01 | 2400 | 6,507.878 ms | 28.2317 ms | 26.4079 ms | 0.45 |
| FloydWarshall\_02 | 2400 | 2,076.403 ms | 20.8320 ms | 19.4863 ms | 0.14 |
| FloydWarshall\_03 | 2400 | 2,568.676 ms | 31.7359 ms | 29.6858 ms | 0.18 |
| FloydWarshall\_00 | 4800 | 119,768.219 ms | 181.5514 ms | 160.9406 ms | 1.00 |
| FloydWarshall\_01 | 4800 | 55,590.374 ms | 414.6051 ms | 387.8218 ms | 0.46 |
| FloydWarshall\_02 | 4800 | 15,614.506 ms | 115.6996 ms | 102.5647 ms | 0.13 |
| FloydWarshall\_03 | 4800 | 18,257.991 ms | 84.5978 ms | 79.1329 ms | 0.15 |

# Завдання для всіх варіантів

1. Вихідний код алгоритму для завдання слід оформити у вигляді бібліотеки класів.
2. Розгорнути локальний репозитарій Nuget.
3. Отриману бібліотеку класів спакувати, як Nuget пакет та опублікувати на локальний репозитарій. Імя пакету має бути (латинською) – перша літера вашого імені та фамілія.
4. Консольний застосунок має отримувати необхідні функціїї через підключений Nuget пакет.
5. Всі необхідні скриптти, консольні команди додано у проект з описом послідовності виконання та будь якими іншими деталями що необхідно зробити для успішного запуску.

# Варіант 1

Вася Пупкин взял листочек в клетку и начал его резать по определённым линиям. На запасном листке такого же размера он закрасил клетки, по которым проходили линии. Василий Васильевич так увлёкся этим занятием, что запутался, сколько частей от листа у него осталось. Ваша задача найти это число.

Входные данные

Во входном файле INPUT.TXT в первой строке записаны N и M (0 < N, M ≤ 100) – размерность матрицы. Далее записана матрица из N строк, каждая из которых содержит M нулей и единиц. 0 обозначает не закрашенную клетку и 1 – закрашенную (линию разреза).

Выходные данные

В выходной файл OUTPUT.TXT следует вывести количество оставшихся частей листа.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 4 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 1 | 2 |

# Варіант 2

В неориентированном графе требуется найти длину кратчайшего пути между двумя вершинами.

Входные данные

Во входном файле INPUT.TXT записано сначала число N - количество вершин в графе (1 ≤ N ≤ 100). Затем записана матрица смежности (0 обозначает отсутствие ребра, 1 - наличие ребра). Затем записаны номера двух вершин - начальной и конечной.

Выходные данные

В выходной файл OUTPUT.TXT выведите длину кратчайшего пути. Если пути не существует, выведите одно число -1.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 3 5 | 3 |

# Варіант 3

В городском управлении милиции одного прибрежного города ведется расследование крупного дела, в котором могут быть замешаны сотрудники милиции. Было принято решение о тайной установке оборудования для просмотра информации, поступающей через Интернет. Под подозрение попадают два отдела, но добиться выделения денег на покупку двух комплектов оборудования не удалось. К счастью, внутренняя сеть управления имеет древовидную структуру, то есть каждый отдел имеет выход в Интернет через какой-либо другой отдел. Исключение составляет отдел по борьбе с компьютерными преступлениями, который имеет непосредственный доступ в Интернет по модемной линии.

Можно было бы установить оборудование для слежения прямо в этом отделе, но для предотвращения злоупотреблений лучше найти такое расположение, чтобы нарушалась секретность как можно меньшего количества лишних отделов.

Как наиболее опытному в подобных вопросах сотруднику, решение этой задачи поручили вам. Подчиненные уже пронумеровали все отделы натуральными числами, начиная с 1, первый номер присвоен отделу по борьбе с компьютерными преступлениями.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N (N ≤ 30000) – количество отделов. Во второй строке записаны номера двух отделов, за которыми необходимо установить слежение. На третьей строке находятся n-1 натуральных чисел, i-е из них не больше i и задает номер отдела, к которому подсоединен отдел i + 1.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – номер отдела, в котором следует установить следящее оборудование.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 3 4 1 1 3 | 3 |
| 2 | 8 3 6 1 1 2 4 5 1 1 | 1 |

# Варіант 4

В клубе N человек. Многие из них - друзья. Так же известно, что друзья друзей так же являются друзьями. Требуется выяснить, сколько всего друзей у конкретного человека в клубе.

Входные данные

В первой строке входного файла INPUT.TXT заданы два числа: N и S (1 ≤ N ≤ 100; 1 ≤ S ≤ N), где N - количество человек в клубе, а S – номер конкретного человека. В следующих N строках записано по N чисел - матрица смежности, состоящая из единиц и нулей. Причем единица, стоящая в i-й строке и j-м столбце гарантирует, что люди с номерами i и j – друзья, а 0 – выражает неопределенность.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество гарантированных друзей у человека с номером S, помня о транзитивности дружбы.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 1 0 1 0 1 0 1 0 1 0 | 2 |

# Варіант 5

На шахматной доске N×N в клетке (x1,y1) стоит голодный шахматный конь. Он хочет попасть в клетку (x2,y2), где растет вкусная шахматная трава. Какое наименьшее количество ходов он должен для этого сделать?

Входные данные

Входной файл INPUT.TXT содержит пять чисел: N, x1, y1, x2, y2 (5 ≤ N ≤ 20, 1 ≤ x1, y1, x2, y2 ≤ N). Левая верхняя клетка доски имеет координаты (1,1), правая нижняя - (N,N).

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести наименьшее число ходов коня.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 1 1 3 1 | 2 |

# Варіант 6

Алхимики средневековья владели знаниями о превращении различных химических веществ друг в друга. Это подтверждают и недавние исследования археологов.

В ходе археологических раскопок было обнаружено m глиняных табличек, каждая из которых была покрыта непонятными на первый взгляд символами. В результате расшифровки выяснилось, что каждая из табличек описывает одну алхимическую реакцию, которую умели проводить алхимики.

Результатом алхимической реакции является превращение одного вещества в другое. Задан набор алхимических реакций, описанных на найденных глиняных табличках, исходное вещество и требуемое вещество. Необходимо выяснить: возможно ли преобразовать исходное вещество в требуемое с помощью этого набора реакций, а в случае положительного ответа на этот вопрос — найти минимальное количество реакций, необходимое для осуществления такого преобразования.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число m (0 ≤ m ≤ 1000) – количество записей в книге. Каждая из последующих m строк описывает одну алхимическую реакцию и имеет формат вещество1 -> вещество2, где вещество1 – название исходного вещества, вещество2 – название продукта алхимической реакции. m+2-ая строка входного файла содержит название вещества, которое имеется исходно, m+3-ая – название вещества, которое требуется получить.

Во входном файле упоминается не более 100 различных веществ. Название каждого из веществ состоит из строчных и заглавных английских букв и имеет длину не более 20 символов. Строчные и заглавные буквы различаются.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное количество алхимических реакций, которое требуется для получения требуемого вещества из исходного, или -1, если требуемое вещество невозможно получить.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Aurum | 2 |
| 2 | 5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Osmium |  |

# Варіант 7

Неориентированный граф без петель и кратных ребер задан матрицей смежности. Требуется определить, является ли этот граф деревом.

Входные данные

Во входном файле INPUT.TXT записано сначала число N - количество вершин графа (от 1 до 100). Далее записана матрица смежности размером N×N, в которой 1 обозначает наличие ребра, 0 - его отсутствие. Матрица симметрична относительно главной диагонали.

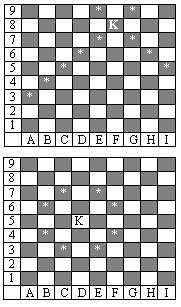
Выходные данные

В выходной файл OUTPUT.TXT выведите сообщение YES, если граф является деревом, и NO в противном случае.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 0 1 0 1 0 1 0 1 0 | YES |

# Варіант 8

Ваш друг Вася занимается разработкой компьютерной игры «Мифические шахматы». Он не укладывается в установленные сроки сдачи проекта.

Вася обратился к друзьям за помощью. Ему необходим модуль, вычисляющий оптимальные пути перемещения фигур из одной клетки в другую. Так как друзей у Васи много, то каждому досталась маленькая подзадача. Вам требуется написать программу, определяющую минимальное количество ходов, необходимое кентавру, чтобы добраться из одной клетки в другую.

В мифические шахматы играют на шахматной доске размером 9х9, угловые клетки которой окрашены в черный цвет. Кентавр – фигура мифических шахмат, объединяющая в себе свойства коня и слона. Когда кентавр стоит на белой клетке, он может ходить только как конь, а когда на черной – только как слон. На рисунках приведены варианты ходов для двух кентавров (буквой «K» отмечено местоположение кентавра, а звездочками – клетки, куда кентавр может сделать ход).

Входные данные

В первой строке входного файла INPUT.TXT содержатся координаты (большая английская буква и цифра) двух клеток доски для мифических шахмат, разделенные пробелом.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное число ходов, необходимое кентавру, чтобы добраться из первой клетки во вторую. Если добраться невозможно, то следует вывести число «-1» (без кавычек).

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | H6 E5 | 2 |
| 2 | A6 F6 | 3 |

# Варіант 9

В таблице из N строк и N столбцов некоторые клетки заняты шариками, другие свободны. Выбран шарик, который нужно переместить, и место, куда его нужно переместить. Выбранный шарик за один шаг перемещается в соседнюю по горизонтали или вертикали свободную клетку. Требуется выяснить, возможно ли переместить шарик из начальной клетки в заданную, и, если возможно, то найти путь из наименьшего количества шагов.

Входные данные

В первой строке входного файла INPUT.TXT находится число N, в следующих N строках - по N символов. Символом точки обозначена свободная клетка, английской заглавной O - шарик, @ - исходное положение шарика, который должен двигаться, английской заглавной X - конечное положение шарика. (2 ≤ N ≤ 40)

Выходные данные

В выходной файл OUTPUT.TXT выведите в первой строке Yes, если движение возможно, или No, если нет. Если движение возможно, то далее следует вывести N строк по N символов - как и на вводе, но букву X, а также все точки по пути следует заменить плюсами. Если решений несколько, выведите любое.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 ....X .OOOO ..... OOOO. @.... | Yes +++++ +OOOO +++++ OOOO+ @++++ |
| 2 | 5 ..X.. ..... OOOOO ..... ..@.. | No |

# Варіант 10

На банкет были приглашены N Очень Важных Персон (ОВП). Были поставлены 2 стола. Столы достаточно большие, чтобы все посетители банкета могли сесть за любой из них. Проблема заключается в том, что некоторые ОВП не ладят друг с другом и не могут сидеть за одним столом. Вас попросили определить, возможно ли всех ОВП рассадить за двумя столами.

Входные данные

В первой строке входного файла INPUT.TXT дано два целых числа: N и M (0 ≤ N,M ≤ 100), где N - количество ОВП, а M - количество пар ОВП, которые не могут сидеть за одним столом. В следующих M строках записано по 2 числа - пары ОВП, которые не могут сидеть за одним столом.

Выходные данные

Если способ рассадить ОВП существует, то в выходной файл OUTPUT.TXT выведите YES и NO в противном случае.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 2 1 2 1 3 | YES |

# Варіант 11

Дана шахматная доска, состоящая из N×N клеток, несколько из них вырезано. Провести ходом коня через невырезанные клетки путь минимальной длины из одной заданной клетки в другую.

Входные данные

В первой строке входного файла INPUT.TXT задано число N (2 ≤ N ≤ 50). В следующих N строках содержится по N символов. Символом # обозначена вырезанная клетка, точкой - невырезанная клетка, @ - заданные клетки (таких символов два), соответствующие началу и концу пути коня.

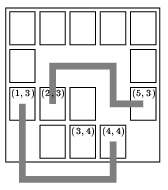
Выходные данные

Если путь построить невозможно, в выходной файл OUTPUT.TXT следует вывести "Impossible", в противном случае вывести такую же карту, как и на входе, но пометить все промежуточные положения коня символом @.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 ..... .@@.. ..... ..... ..... | ...@. .@@.. ....@ ..... ..... |
| 2 | 5 @..@. ..##. ..... ..... ..... | @..@. ..##. .@..@ ..@.. @.... |
| 3 | 5 @.... ..#.. .#... ..... ....@ | Impossible |

# Варіант 12

Вы являетесь одним из разработчиков новой компьютерной игры. Игра происходит на прямоугольной доске, состоящей из W×H клеток. Каждая клетка может либо содержать, либо не содержать фишку. Важной частью игры является проверка того, соединены ли две фишки путем, удовлетворяющим следующим свойствам:

Путь должен состоять из отрезков вертикальных и горизонтальных прямых.

Путь не должен пересекать других фишек. При этом часть пути может оказаться вне доски. Например:

Фишки с координатами (1,3) и (4,4) могут быть соединены. Фишки с координатами (2,3) и (5,3) тоже могут быть соединены. А вот фишки с координатами (2,3) и (3,4) соединить нельзя – любой соединяющий их путь пересекает другие фишки.

Вам необходимо написать программу, проверяющую, можно ли соединить две фишки путем, обладающим вышеуказанными свойствами, и, в случае положительного ответа, определяющую минимальную длину такого пути (считается, что путь имеет изломы, начало и конец только в центрах клеток (или «мнимых клеток», расположенных вне доски), а отрезок, соединяющий центры двух соседних клеток, имеет длину 1).

Входные данные

Первая строка входного файла INPUT.TXT содержит два натуральных числа: W – ширина доски, H – высота доски (1≤W,H≤75). Следующие H строк содержат описание доски: каждая строка состоит ровно из W символов: символ «X» (заглавная английская буква «экс») обозначает фишку, символ «.» (точка) обозначает пустое место. Все остальные строки содержат описания запросов: каждый запрос состоит из четырёх натуральных чисел, разделённых пробелами – X1, Y1, X2, Y2, причём 1≤X1,X2≤W, 1≤Y1,Y2≤H. Здесь (X1, Y1) и (X2, Y2) – координаты фишек, которые требуется соединить (левая верхняя клетка имеет координаты (1,1)). Гарантируется, что эти координаты не будут совпадать (кроме последнего запроса; см. далее). Последняя строка содержит запрос, состоящий из четырёх чисел 0; этот запрос обрабатывать не надо. Количество запросов не превосходит 20.

Выходные данные

Для каждого запроса в выходной файл OUTPUT.TXT необходимо вывести одно целое число на отдельной строке – длину кратчайшего пути, или 0, если такого пути не существует.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 4 XXXXX X...X XXX.X .XXX. 2 3 5 3 1 3 4 4 2 3 3 4 0 0 0 0 | 5 6 0 |

# Варіант 13

На плоскости расположено несколько прямоугольников. Каждый прямоугольник на плоскости задается координатами левого нижнего угла (X1, Y1) и правого верхнего угла (X2, Y2), при этом стороны прямоугольников параллельны осям координат. При наложении друг на друга прямоугольники образуют фигуры, отдельно расположенный прямоугольник - тоже фигура. Прямоугольники, соприкасающиеся только углами, не образуют фигуру. Если прямоугольники соприкасаются сторонами, то они тоже образуют фигуру. Требуется определить фигуру максимальной площади.

Входные данные

В первой строке входного файла INPUT.TXT записано количество прямоугольников N (1 ≤ N ≤ 25), далее идут N строк с координатами вершин прямоугольников X1, Y1, X2, Y2, разделенных пробелом. Координаты вершин - целые, неотрицательные числа, в диапазоне от 0 до 100 включительно.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – площадь найденной фигуры.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 2 15 15 25 25 10 10 20 20 | 175 |
| 2 | 5 1 1 6 2 2 1 3 5 2 4 7 5 6 2 7 5 8 4 10 7 | 14 |

# Варіант 14

Представьте, что вы состоите на службе во внешней разведке Межгалактического Альянса Республиканских Сил (МАРС). Одному из агентов разведки крупно не повезло, и он был захвачен на засекреченной космической базе. К счастью, внешней разведке МАРС удалось заполучить план этой базы. И вот теперь вам поручено разработать план побега.

База представляет собой прямоугольник размером NхM, со всех сторон окружённый стенами, и состоящий из квадратных отсеков единичной площади. База снабжена K выходами, до одного из которых агенту необходимо добраться. В некоторых отсеках базы находятся стены. Ваш агент может перемещаться из отсека в любой из четырех соседних с ним, если в том отсеке, куда он хочет переместиться, нет стены.

Кроме того, база снабжена системой гипертуннелей, способных перемещать агента из одного отсека базы (вход в гипертуннель) в другой (выход из гипертуннеля). Когда агент находится в отсеке, где есть вход в гипертуннель, он может (но не обязан) им воспользоваться.

Начальное положение вашего агента известно. Вам необходимо найти кратчайший путь побега (то есть путь, проходящий через минимальное количество отсеков).

Входные данные

В первой строке входного файла INPUT.TXT записаны числа N и M (2 ≤ N ≤ 100, 2 ≤ M ≤ 100), задающие размеры базы: N — количество строк в плане базы, M — количество столбцов. Во второй строке записаны начальные координаты агента XA,YA (1 ≤ XA ≤ N, 1 ≤ YA ≤ M). Первая координата задает номер строки, вторая — номер столбца. Строки нумеруются сверху вниз, столбцы слева направо. Далее следуют N строк по M чисел, задающих описание стен внутри базы: 1 соответствует стенке, 0 — её отсутствию. Далее в отдельной строке записано число H (0 ≤ H ≤ 1000) — количество гипертуннелей. В последующих H строках идут описания гипертуннелей. Каждый гипертуннель задается 4 числами: X1, Y1, X2, Y2 (1 ≤ X1,X2 ≤ N; 1 ≤ Y1,Y2 ≤ M) — координатами входа и выхода гипертуннеля. Никакие два гипертуннеля не имеют общего входа. После этого в отдельной строке следует число K (1 ≤ K ≤ 10) — количество выходов с базы. В последующих K строках идут описания выходов с базы. Каждый выход задается двумя координатами X и Y (1 ≤ X ≤ N; 1 ≤ Y ≤ M).

Гарантируется, что начальные координаты агента не совпадают ни с одним из выходов и он не стоит в отсеке, занятом стеной. Никакие входы и выходы гипертуннелей, а также выходы с базы не находятся в отсеках, занятых стенами. Никакой вход в гипертуннель не совпадает с выходом с базы.

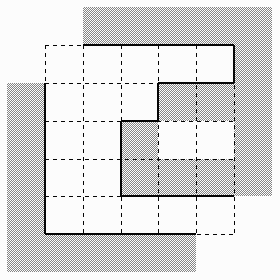
Выходные данные

Если побег невозможен, выведите в выходной файл OUTPUT.TXT "Impossible". В противном случае следует вывести количество отсеков в кратчайшем пути побега.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 5 2 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 4 3 1 1 4 1 2 4 | 4 |

# Варіант 15

Лабиринт представляет собой квадрат, состоящий из N×N сегментов. Каждый из сегментов может быть либо пустым, либо заполненным монолитной каменной стеной. Гарантируется, что левый верхний и правый нижний сегменты пусты. Лабиринт обнесён сверху, снизу, слева и справа стенами, оставляющими свободными только левый верхний и правый нижний углы. Директор лабиринта решил покрасить стены лабиринта, видимые изнутри (см. рисунок). Помогите ему рассчитать количество краски, необходимой для этого.

Входные данные

В первой строке входного файла INPUT.TXT находится число N, затем идут N строк по N символов: точка обозначает пустой сегмент, решётка - сегмент со стеной. Размер сегментов – 5х5 метров, высота стен – 5 метров. (3 ≤ N ≤ 50)

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - площадь видимой части внутренних стен лабиринта в квадратных метрах.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 ..... ...## ..#.. ..### ..... | 550 |

# Варіант 16

Группа солдат-новобранцев прибыла в армейскую часть N666. После знакомства с прапорщиком стало очевидно, что от работ на кухне по очистке картофеля спасти солдат может только чудо.

Прапорщик, будучи не в состоянии запомнить фамилии, пронумеровал новобранцев от 1 до N. После этого он велел им построиться по росту (начиная с самого высокого). С этой несложной задачей могут справиться даже совсем необученные новобранцы, да вот беда, прапорщик уверил себя, что знает про некоторых солдат, кто из них кого выше, и это далеко не всегда соответствует истине.

После трех дней обучения новобранцам удалось выяснить, что знает (а точнее, думает, что знает) прапорщик. Помогите им, используя эти знания, построиться так, чтобы товарищ прапорщик остался доволен.

Входные данные

Во входном файле INPUT.TXT сначала идут числа N и M (1 ≤ N ≤ 100, 1 ≤ M ≤ 5000) - количество солдат в роте и количество пар солдат, про которых прапорщик знает, кто из них выше. Далее идут эти пары чисел A и B по одной на строке (1 ≤ A,B ≤ N), что означает, что, по мнению прапорщика, солдат A выше, чем B.

Выходные данные

В выходной файл OUTPUT.TXT выведите "Yes" если можно построиться так, чтобы прапорщик остался доволен и "No" если нельзя.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 4 1 3 1 4 4 3 5 2 | Yes |

# Варіант 17

Одним из важных понятий, используемых в теории алгоритмов, является рекурсия. Неформально ее можно определить как использование в описании объекта самого себя. Если речь идет о процедуре, то в процессе исполнении эта процедура напрямую или косвенно (через другие процедуры) вызывает сама себя.

Рекурсия является очень «мощным» методом построения алгоритмов, но таит в себе некоторые опасности. Например, неаккуратно написанная рекурсивная процедура может войти в бесконечную рекурсию, то есть, никогда не закончить свое выполнение (на самом деле, выполнение закончится с переполнением стека).

Поскольку рекурсия может быть косвенной (процедура вызывает сама себя через другие процедуры), то задача определения того факта, является ли данная процедура рекурсивной, достаточно сложна. Попробуем решить более простую задачу.

Рассмотрим программу, состоящую из n процедур P1, P2, …, Pn. Пусть для каждой процедуры известны процедуры, которые она может вызывать. Процедура P называется потенциально рекурсивной, если существует такая последовательность процедур Q0, Q1, …, Qk, что Q0 = Qk = P и для i = 1…k процедура Qi-1 может вызвать процедуру Qi. В этом случае задача будет заключаться в определении для каждой из заданных процедур, является ли она потенциально рекурсивной.

Требуется написать программу, которая позволит решить названную задачу.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n — количество процедур в программе (1 ≤ n ≤ 100). Далее следуют n блоков, описывающих процедуры. После каждого блока следует строка, которая содержит 5 символов «\*».

Описание процедуры начинается со строки, содержащий ее идентификатор, состоящий только из маленьких букв английского алфавита и цифр. Идентификатор непуст, и его длина не превосходит 100 символов. Далее идет строка, содержащая число k (k ≤ n ) — количество процедур, которые могут быть вызваны описываемой процедурой. Последующие k строк содержат идентификаторы этих процедур — по одному идентификатору на строке.

Различные процедуры имеют различные идентификаторы. При этом ни одна процедура не может вызвать процедуру, которая не описана во входном файле.

Выходные данные

В выходной файл OUTPUT.TXT для каждой процедуры, присутствующей во входных данных, необходимо вывести слово YES, если она является потенциально рекурсивной, и слово NO – в противном случае, в том же порядке, в каком они перечислены во входных данных.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 p1 2 p1 p2 \*\*\*\*\* p2 1 p1 \*\*\*\*\* p3 1 p1 \*\*\*\*\* | YES YES NO |

# Варіант 18

Дан взвешенный граф. Определить, есть ли в нем цикл отрицательного веса.

Входные данные

Во входном файле INPUT.TXT в первой строке записано число N (1 ≤ N ≤ 100) - количество вершин графа. В следующих N строках находится по N чисел - матрица смежности графа. Веса ребер не превышают по модулю 10000. Если ребра нет, соответствующее значение равно 100000.

Выходные данные

В выходной файл OUTPUT.TXT выведите "YES", если цикл существует, или "NO" в противном случае.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 2 0 -1 -1 0 | YES |

# Варіант 19

Дан ориентированный взвешенный граф. Для него вам необходимо найти кратчайшее расстояние от вершины S до вершины F.

Входные данные

В первой строке входного файла INPUT.TXT записаны три числа: N, S и F (1 ≤ N ≤ 100; 1 ≤ S, F ≤ N), где N - количество вершин графа. В следующих N строках записаны по N чисел - матрица смежности графа, где число в i-ой строке j-ом столбце соответствует ребру из i в j: -1 означает отсутствие ребра между вершинами, а любое неотрицательное целое число (от 0 до 100) - наличие ребра данного веса. На главной диагонали матрицы всегда записаны нули.

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести искомое расстояние или -1, если пути между указанными вершинами не существует.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 2 1 0 1 1 4 0 1 2 1 0 | 3 |

# Варіант 20

Преподаватель читает курс лекций, в рамках которого обычно доказывается N различных теорем. Некоторые теоремы могут ссылаться в доказательстве друг на друга. Более точно, каждая теорема Ti зависит от некоторого набора из Ci других теорем; доказать ее можно лишь доказав не менее половины теорем из данного набора. При этом структура курса такова, что нет такой теоремы, от которой зависели бы две или более различных теоремы, а также нет цепочки теорем (Ti1,Ti2, . . . , Tis) такой, что Ti1 зависит от Ti2, Ti2 зависит от Ti3, …, Tis−1 зависит от Tis, а Tis – от Ti1.

Однако, в этом семестре в связи с обилием праздников, перекрывающихся с лекциями, может не удаться доказать все теоремы курса. Тем не менее, нужно доказать основную теорему курса – это центральный результат всей теории, и именно его, скорее всего, придется применять слушателям в других курсах в следующем семестре. Поэтому преподаватель хочет расположить теоремы в таком порядке, чтобы основную теорему курса удалось доказать как можно раньше. Затем, если останется время, он сможет вернуться к доказательству других, менее важных теорем.

Для простоты будем считать, что все теоремы доказываются за одинаковое время. Нужно доказать такое множество теорем и в таком порядке, чтобы основная теорема оказалась доказанной и чтобы общее время доказательства было минимально.

Входные данные

В первой строке входного файла INPUT.TXT записано число N (1 ≤ N ≤ 10 000) – количество теорем. Каждая из следующих N строк описывает теоремы, от которых зависит Ti−1, где i – номер этой строки во входном файле. Эти строки имеют вид Ai,1 Ai,2 ... Ai,Ci 0; здесь Ai,j – номер теоремы, от которой зависит Ti−1. Среди всех чисел Ai,j во входном файле нет двух одинаковых. Основная теорема имеет номер 1. Все числа во входном файле целые.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите K – минимальное количество теорем, которые потребуется доказать. В последующих K строках выведите номера этих теорем в порядке их доказательства, по одному числу в каждой. Если ответов с минимальным K несколько, можно вывести любой из них.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 2 2 0 0 | 2 2 1 |
| 2 | 6 2 3 6 0 4 0 0 0 0 5 0 | 4 4 3 2 1 |
| 3 | 3 0 1 0 2 0 | 1 1 |

# Варіант 21

Открыв глаза, Принц Персии обнаружил, что находится на верхнем уровне подземного лабиринта Джаффара. Лабиринт состоит из h уровней, расположенных строго друг под другом. Каждый уровень представляет собой прямоугольную площадку, разбитую на m х n участков. На некоторых участках стоят колонны, поддерживающие потолок, на такие участки Принц заходить не может.

Принц может перемещаться с одного участка на другой соседний свободный участок того же уровня, так же он может проломить пол под собой и оказаться уровнем ниже (на самом нижнем уровне пол проломить нельзя). Любое перемещение занимает у Принца 5 секунд.

На одном из участков нижнего уровня Принца ждет Принцесса. Помогите Принцу найти Принцессу, потратив на это как можно меньше времени.

Входные данные

В первой строке входного файла INPUT.TXT содержатся натуральные числа h, m и n — высота и горизонтальные размеры лабиринта (2 ≤ h, m, n ≤ 50). Далее во входном файле приведены h блоков, описывающих уровни лабиринта в порядке от верхнего к нижнему. Каждый блок содержит m строк, по n символов в каждой: «.» обозначает свободный участок, «о» обозначает участок с колонной, «1» обозначает свободный участок, в котором оказался Принц в начале своего путешествия, «2» обозначает свободный участок, на котором томится Принцесса. Символы «1» и «2» встречаются во входном файле ровно по одному разу: символ «1» — в описании самого верхнего уровня, а символ «2» — в описании самого нижнего. Соседние блоки разделены одной пустой строкой.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное время в секундах, необходимое Принцу, чтобы найти Принцессу. Поскольку добро всегда побеждает Зло, гарантируется, что Принц может это сделать.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 3 3 1.. oo. ...  ooo ..o .oo  ooo o.. o.2 | 60 |

# Варіант 22

Об этом еще никто не знает, но многие догадываются – мафия уже в городе. Поговаривают, что в планах главы мафиозного клана захват контроля над всем городом, однако поначалу он решил ограничиться захватом основных линий связи города.

В городе находятся n базовых телефонных станций, некоторые пары которых соединены двусторонними каналами связи. Для удобства, занумеруем базовые станции целыми числами от 1 до n, канал связи в этом случае задается парой чисел (u, v) – номерами станций, которые он соединяет.

Будем говорить, что канал связи (u, v) – контролируется мафией, если захвачена, либо станция u, либо станция v (либо обе).

Глава мафиозного клана хочет контролировать все каналы связи, захватив при этом как можно меньше базовых станций. Ваша задача помочь службе безопасности телефонной компании, составив возможный план захвата и определив количество таких планов.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа: n и m (2 ≤ n ≤ 18, 0 ≤ m). Каждая из последующих m строк описывает один канал связи и содержит по два целых числа: u и v (1 ≤ u, v ≤ n, u ≠ v) – номера базовых станций, соединенных этим каналом связи. Любая пара станций соединена не более, чем одним каналом.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите два числа: k и c – соответственно, минимальное количество базовых станций, которые необходимо захватить для того, чтобы контролировать все каналы связи, и число способов захватить такое количество станций так, чтобы контролировать все каналы связи.

Во второй строке выведите k чисел – номера базовых станций, соответствующих одному из способов захвата.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 3 1 2 2 3 3 1 | 2 3 1 2 |
| 2 | 5 4 1 2 1 3 1 4 1 5 | 1 1 1 |

# Варіант 23

Ане, как будущей чемпионке мира по программированию, поручили очень ответственное задание. Правительство вручает ей план постройки дорог между N городами. По плану все дороги односторонние, но между двумя городами может быть больше одной дороги, возможно, в разных направлениях. Ане необходимо вычислить минимальное такое K, что данный ей план является слабо K-связным.

Правительство называет план слабо K-связным, если выполнено следующее условие: для любых двух различных городов можно проехать от одного до другого, нарушая правила движения не более K раз. Нарушение правил - это проезд по существующей дороге в обратном направлении. Гарантируется, что между любыми двумя городами можно проехать, возможно, несколько раз нарушив правила.

Входные данные

В первой строке входного файла INPUT.TXT записаны два числа 2 ≤ N ≤ 300 и 1 ≤ M ≤ 105 - количество городов и дорог в плане. В последующих M строках даны по два числа - номера городов, в которых начинается и заканчивается соответствующая дорога.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное K, такое, что данный во входном файле план является слабо K-связным.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 2 1 2 1 3 | 1 |
| 2 | 4 4 2 4 1 3 4 1 3 2 | 0 |

# Варіант 24

В городе N в ближайшее время состоится этап чемпионата мира по автогонкам среди автомобилей класса Формула-0. Поскольку специальный автодром для этих соревнований организаторы построить не успели, было решено организовать трассу на улицах города.

В городе N есть n перекрестков, некоторые пары которых соединены дорогами, движение по которым возможно в обоих направлениях. При этом любые два перекрестка соединены не более чем одной дорогой, и есть возможность доехать по дорогам от любого перекрестка до любого другого.

Трасса, на которой будут проводиться соревнования, должна быть круговой (т.е. должна начинаться и заканчиваться на одном и том же перекрестке), при этом в процессе движения по ней никакой перекресток не должен встречаться более одного раза.

На предварительном этапе подготовки оргкомитетом был создан список всех дорог города. Теперь настало время его использовать. Первый вопрос, который необходимо решить, вопрос о существовании в городе требуемой круговой трассы (разумеется, если ответ будет отрицательным, организаторам придется в срочном порядке построить еще несколько дорог). Единственная проблема заключается в том, что у организаторов есть подозрение, что, поскольку список составлялся не очень внимательно, в нем некоторые дороги указаны больше одного раза.

Напишите программу, которая по заданному списку дорог города определит, возможна ли организация в городе требуемой круговой трассы.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа: n (1 ≤ n ≤ 103) – количество перекрестков в городе N и m (0 ≤ m ≤ 105) – количество дорог в составленном списке.

Последующие m строк описывают дороги. Каждая дорога описывается двумя числами: u и v (1 ≤ u, v ≤ n, u ≠ v) номерами перекрестков, которые она соединяет. Так как дороги двухсторонние, то пара чисел (u, v) и пара чисел (v, u) описывают одну и ту же дорогу.

Выходные данные

В выходной файл OUTPUT.TXT выведите слово YES, если в городе возможно организовать круговую трассу для соревнований, и слово NO в противном случае.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 4 1 2 2 3 3 1 3 2 | YES |
| 2 | 2 3 1 2 2 1 2 1 | NO |

# Варіант 25

Полный ориентированный взвешенный граф задан матрицей смежности. Постройте матрицу кратчайших путей между его вершинами. Гарантируется, что в графе нет циклов отрицательного веса.

Входные данные

В первой строке входного файла INPUT.TXT записано единственное число N (1 ≤ N ≤ 100) - количество вершин графа. В следующих N строках по N чисел - матрица смежности графа (j-ое число в i-ой строке соответствует весу ребра из вершины i в вершину j). Все числа по модулю не превышают 100. На главной диагонали матрицы - всегда нули.

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк по N чисел - матрицу кратчайших расстояний между парами вершин. j-ое число в i-ой строке должно быть равно весу кратчайшего пути из вершины i в вершину j.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 0 5 9 100 100 0 2 8 100 100 0 7 4 100 100 0 | 0 5 7 13 12 0 2 8 11 16 0 7 4 9 11 0 |

# Варіант 26

Дан ориентированный граф, в котором могут быть кратные ребра и петли. Каждое ребро имеет вес, выражающийся целым числом (возможно, отрицательным). Гарантируется, что циклы отрицательного веса отсутствуют.

Требуется посчитать длины кратчайших путей от вершины номер 1 до всех остальных вершин.

Входные данные

В первой строке входного файла INPUT.TXT записаны целые числа N и M - количество вершин и количество ребер графа (1 ≤ N ≤ 100, 0 ≤ M ≤ 10000). В каждой из последующих M строк записана тройка чисел, описывающих ребра: начало ребра, конец ребра и вес (вес - целое число от -100 до 100).

Выходные данные

В выходной файл OUTPUT.TXT выведите N чисел - расстояния от вершины номер 1 до всех вершин графа. Если пути до соответствующей вершины не существует, вместо длины пути выведите число 30000.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 5 1 2 10 2 3 10 1 3 100 3 1 -10 2 3 1 | 0 10 11 30000 |

# Варіант 27

В одном из парков одного большого города недавно был организован новый аттракцион Цветной лабиринт. Он состоит из n комнат, соединенных m двунаправленными коридорами. Каждый из коридоров покрашен в один из ста цветов, при этом от каждой комнаты отходит не более одного коридора каждого цвета. При этом две комнаты могут быть соединены любым количеством коридоров.

Человек, купивший билет на аттракцион, оказывается в комнате номер один. Кроме билета, он также получает описание пути, по которому он может выбраться из лабиринта. Это описание представляет собой последовательность цветов c1…ck. Пользоваться ей надо так: находясь в комнате, надо посмотреть на очередной цвет в этой последовательности, выбрать коридор такого цвета и пойти по нему. При этом если из комнаты нельзя пойти по коридору соответствующего цвета, то человеку приходится дальше самому выбирать, куда идти.

В последнее время в администрацию парка стали часто поступать жалобы от заблудившихся в лабиринте людей. В связи с этим, возникла необходимость написания программы, проверяющей корректность описания и пути, и, в случае ее корректности, сообщающей номер комнаты, в которую ведет путь.

Описание пути некорректно, если на пути, который оно описывает, возникает ситуация, когда из комнаты нельзя пойти по коридору соответствующего цвета.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа n (1 ≤ n ≤ 10000) и m (1 ≤ m ≤ 100000) - соответственно количество комнат и коридоров в лабиринте. Следующие m строк содержат описания коридоров. Каждое описание содержит три числа u (1 ≤ u ≤ n), v (1 ≤ v ≤ n), c (1 ≤ c ≤ 100) - соответственно номера комнат, соединенных этим коридором, и цвет коридора. Следующая, (m+2)-ая строка входного файла содержит длину описания пути - целое число k (0 ≤ k ≤ 100000). Последняя строка входного файла содержит k целых чисел, разделенных пробелами, - описание пути по лабиринту.

Выходные данные

В выходной файл OUTPUT.TXT выведите строку INCORRECT, если описание пути некорректно, иначе выведите номер комнаты, в которую ведет описанный путь. Помните, что путь начинается в комнате номер один.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 2 1 2 10 1 3 5 5 10 10 10 10 5 | 3 |
| 2 | 3 2 1 2 10 2 3 5 5 5 10 10 10 10 | INCORRECT |
| 3 | 3 2 1 2 10 1 3 5 4 10 10 10 5 | INCORRECT |

# Варіант 28

Дан ориентированный взвешенный граф. Вам необходимо найти пару вершин, кратчайшее расстояние от одной из которых до другой максимально среди всех пар вершин.

Входные данные

В первой строке входного файла INPUT.TXT записано единственное число N (1 ≤ N ≤ 100) - количество вершин графа. В следующих N строках по N чисел - матрица смежности графа, где -1 означает отсутствие ребра между вершинами, а любое неотрицательное число - присутствие ребра данного веса. Элементы матрицы - целые числа от -1 до 100. На главной диагонали матрицы - всегда нули. Гарантируется, что в графе есть хотя бы одно ребро.

Выходные данные

В выходной файл OUTPUT.TXT требуется вывести искомое максимальное кратчайшее расстояние.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 0 5 9 -1 -1 0 2 8 -1 -1 0 7 4 -1 -1 0 | 16 |

# Варіант 29

Любознательный школьник Петя очень любит программировать. Однажды он настолько увлекся этим делом, что уснул прямо за компьютером! Пете приснилось, что он попал в альтернативную реальность. Альтернативная реальность представляет собой прямоугольный лабиринт, который можно изобразить в виде таблицы размером R × C клеток. Чтобы не опоздать школу, Пете нужно найти выход из лабиринта. Начальная позиция Пети обозначается символом ‘S’, выход из альтернативной реальности – ‘E’. За один ход Петя перемещается в одну из четырех смежных клеток (влево, вправо, вниз, вверх). Если клетка занята стеной (символ ‘X’), то Петя пройти в нее не может. В некоторых клетках расположены двери с замками одного из четырех цветов (‘R’, ‘G’, ‘B’, ‘Y’). Для прохода в эту клетку, необходимо обладать ключом определенного цвета. Так как ключи многоразовые, то одним ключом можно открыть сколь угодно много соответствующих ему замков.

Властелин альтернативной реальности предлагает Пете купить ключи, чтобы пройти к выходу. У нашего героя совсем немного денег с собой, поэтому ему хочется потратить как можно меньше денег и при этом пройти к выходу. Помогите ему определить минимальную сумму денег, которую нужно потратить на ключи.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральные числа R и C (R,C ≤ 50). Вторая строка теста содержит 4 целых числа Pi, стоимости покупки ключей ‘R’, ‘G’, ‘B’ и ‘Y’ соответственно (Pi ≤ 100). Далее следуют R строк, в каждой из которых C символов, описывающих лабиринт. Каждый лабиринт содержит только один символ ‘S’ и один символ ‘E’.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальную сумму денег, необходимую для покупки набора ключей, позволяющего пройти к выходу. Если пути к выходу из альтернативной реальности не существует, и Петя обречен проспать уроки, выведите «Sleep».

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 7 1 1 1 1 XXXXXXX XS.X.EX XXXXXXX | Sleep |
| 2 | 6 6 1 5 3 1 XXXXXX XS.X.X X..R.X X.XXBX X.G.EX XXXXXX | 4 |

# Варіант 30

Всем известна увлекательная игра «Морской бой». Сейчас играть в морской бой можно не только с соседом по парте, но и с компьютером. Игра c компьютером ведется на прямоугольном поле произвольных размеров N×M, где N - количество строк, M - количество столбцов. Приближается чемпионат Мира по морскому бою. Планируется вести его трансляцию в режиме реального времени: демонстрировать карту с кораблями и выводить статистику: количество целых, подбитых и уничтоженных кораблей, находящихся на поле. Требуется написать программу для подсчета статистики.

Корабль на поле — это связанная фигура, стоящая из одной или нескольких рядом лежащих клеток, имеющих общую сторону. Корабли могут быть абсолютно любых форм и размеров!

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа N и M (1≤ N,M ≤ 103), разделённых пробелами - размеры игрового поля. Далее идут N строк по M символов - описание игрового поля.

Английская буква 'X' обозначает подбитую клетку корабля, 'S' - не подбитую клетку корабля, '-' – свободное водное пространство.

Выходные данные

В выходной файл OUTPUT.TXT выведите через пробел три числа:

* количество целых кораблей
* количество подбитых кораблей
* количество уничтоженных кораблей

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 8 ---SSS-- XX--S-X- X-S---S- | 2 1 1 |

# Варіант 31

Одна из команд-участниц олимпиады решила вернуться домой на электричках. При этом ребята хотят попасть домой как можно раньше. К сожалению, не все электрички идут от города, где проводится олимпиада, до станции, на которой живут ребята. И, что еще более обидно, не все электрички, которые идут мимо их станции, останавливаются на ней (равно как вообще, электрички останавливаются далеко не на всех станциях, мимо которых они идут).

Все станции на линии пронумерованы числами от 1 до N. При этом станция номер 1 находится в городе, где проводится олимпиада, и в момент времени 0 ребята приходят на станцию. Станция, на которую нужно попасть ребятам, имеет номер E.

Напишите программу, которая по данному расписанию движения электричек вычисляет минимальное время, когда ребята могут оказаться дома.

Входные данные

Во входном файле INPUT.TXT записаны сначала числа N (2 ≤ N ≤ 100) и E (2 ≤ E ≤ N). Затем записано число M (0 ≤ M ≤ 100), обозначающее число рейсов электричек. Далее идет описание M рейсов электричек. Описание каждого рейса электрички начинается с числа Ki (2 ≤ Ki ≤ N) — количества станций, на которых она останавливается, а далее следует Ki пар чисел, первое число каждой пары задает номер станции, второе — время, когда электричка останавливается на этой станции (время выражается целым числом из диапазона от 0 до 109). Станции внутри одного рейса упорядочены в порядке возрастания времени. В течение одного рейса электричка все время движется в одном направлении — либо от города, либо к городу.

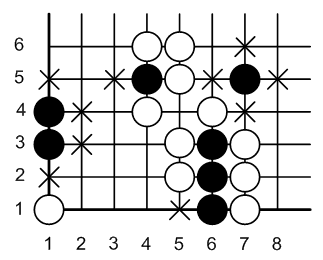
Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — минимальное время, когда ребята смогут оказаться на своей станции. Если существующими рейсами электричек они добраться не смогут, выведите –1.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 3 4 2 1 5 2 10 2 2 10 4 15 4 5 0 4 17 3 20 2 35 3 1 2 3 40 4 45 | 20 |

# Варіант 32

Совсем недавно Али-Баба узнал от своего брата Касима об удивительной игре Го. В Го играют на прямоугольной доске – гобане, расчерченном вертикальными и горизонтальными линиями. Все линии пронумерованы. В игре участвуют два игрока, которые по очереди выставляют на гобан камни – специальные круглые фишки. Каждый камень ставится на незанятую точку пересечения линий доски (пересечения называют пунктами). У одного игрока – черные камни, у другого – белые. Камни одного цвета, смежные по вертикали, либо по горизонтали (но не диагонали), объединяются в группу. Одиночный камень также считается группой.

Один из способов набрать очки в Го – захватить камни противника. Каждый камень может иметь от двух до четырех смежных с ним пунктов (по вертикали и горизонтали, но не по диагонали). Если такой пункт не занят камнем, то он называется «дамэ». Дамэ группы – это все дамэ камней, составляющих группу. Как только оппонент своими камнями закрывает все дамэ чужой группы, то эта группа считается захваченной и снимается с доски. Если у группы осталось лишь одно дамэ, то говорят, что эта группа находится в «атари» т.е. на один шаг от захвата соперником.

Дамэ черных камней на рисунке отмечены крестиком. Группа черных из камней (1, 3) и (1, 4) имеет 4 дамэ. Группа (6, 1), (6, 2) и (6, 3) имеет одно дамэ и находится в атари. Черный камень (4, 5) также находится в атари. Помогите Али-Бабе, который всегда играет черными, определить, какие его группы находятся в атари.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N – размерность игровой доски (6 ≤ N ≤ 19). Далее следует N строк по N символов каждая. Каждый символ описывает один пункт доски. «B» означает черный камень, «W» – белый, «.» означает пустой пункт. Все группы на доске имеют хотя бы одно дамэ.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – количество групп черных камней, находящихся в атари.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 9 ......... ......... ......... ...WW.... ...BW.B.. B..W.W... B...WBW.. ....WBW.. W....BW.. | 2 |
| 2 | 6 WB.WBB .B.W.B ..WW.W WWW..W ..W... BBW... | 1 |

# Варіант 33

Между некоторыми деревнями края Власюки ходят автобусы. Поскольку пассажиропотоки здесь не очень большие, то автобусы ходят всего несколько раз в день.

Марии Ивановне требуется добраться из деревни d в деревню v как можно быстрее (считается, что в момент времени 0 она находится в деревне d).

Входные данные

Во входном файле INPUT.TXT записано число N - общее число деревень (1 ≤ N ≤ 100), номера деревень d и v, затем количество автобусных рейсов R (0 ≤ R ≤ 10000). Затем идут описания автобусных рейсов. Каждый рейс задается номером деревни отправления, временем отправления, деревней назначения и временем прибытия (все времена - целые от 0 до 10000). Если в момент t пассажир приезжает в деревню, то уехать из нее он может в любой момент времени, начиная с t.

Выходные данные

В выходной файл OUTPUT.TXT вывести минимальное время, когда Мария Ивановна может оказаться в деревне v. Если она не сможет с помощью указанных автобусных рейсов добраться из d в v, вывести -1.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 1 3 4 1 0 2 5 1 1 2 3 2 3 3 5 1 1 3 10 | 5 |

# Варіант 34

В постиндустриальную эпоху основной ценностью является информация. Поэтому особо важен контроль над каналами передачи информации. В одной стране все каналы связи контролируются государством.

Перед ИТ-отделом одной достаточно крупной фирмы, занимающейся консалтингом в области инновационных технологий, была поставлена задача распространить некий файл по филиалам этой фирмы, находящимся в различных городах страны.

Каналы передачи информации в этой стране, как уже говорилось, контролируются государством, поэтому за передачу по ним информации приходится платить деньги. Ситуация также осложняется тем, что каналы однонаправленные, то есть информацию по ним можно передавать только в одном направлении.

Пусть, для удобства, города пронумерованы натуральными числами от 1 до n. Главный офис находится в городе номер 1, таким образом, необходимо найти такой набор каналов связи, по которым можно доставить файл от города номер 1 до любого другого, а среди всех таких наборов выбрать имеющий наименьшую суммарную стоимость.

Задан список каналов связи, которыми может воспользоваться фирма. Напишите программу, находящую требуемый набор каналов связи.

Входные данные

Первая строка входного файла INPUT.TXT содержит числа n и m - количество городов и количество каналов связи соответственно (1 ≤ n ≤ 22, 0 ≤ m ≤ 22). Последующие m содержат описания каналов связи. Каждое описание содержит три целых числа: u, v и c - соответственно номера городов, соединенных каналом и стоимость пересылки файла по этому каналу (1 ≤ u, v ≤ n, 0 ≤ c ≤ 1000). Ни один из каналов не соединяет город с самим собой, но между двумя городами может быть больше одного канала.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите стоимость пересылки файла и число каналов, обеспечивающих такую стоимость. Во второй строке выведите номера каналов, составляющих такой набор. Каналы нумеруются от 1 до m в том порядке, в котором они перечислены во входном файле.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 2 2 1 2 3 1 2 4 | 3 1 1 |
| 2 | 3 3 1 2 5 1 3 10 3 2 4 | 14 2 2 3 |

# Варіант 35

Тезею из лабиринта Минотавра помог выйти клубок ниток. Вы можете вместо клубка использовать персональный компьютер.

Требуется написать программу, которая вводит маршрут Тезея в лабиринте и находит кратчайший обратный путь, по которому Тезей сможет выйти из лабиринта, не заходя в тупики и не делая петель.

Входные данные

Входной файл INPUT.TXT содержит маршрут Тезея, который представлен строкой, состоящей из букв: N, S, W, E и длиной от 1 до 200.

Буквы означают:

N - один "шаг" на север,  
S - один "шаг" на юг,  
W - один "шаг" на запад,  
E - один "шаг" на восток.

Выходные данные

В выходной файл OUTPUT.TXT записывается аналогично входному файлу найденный обратный путь. Если маршрут неоднозначен, то следует выбирать согласно следующему приоритету: N, E, S, W.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | EENNESWSSWE | NWW |

# Варіант 36

В стране N городов, некоторые из которых соединены между собой дорогами. Для того, чтобы проехать по одной дороге требуется один бак бензина. В каждом городе бак бензина имеет разную стоимость. Вам требуется добраться из первого города в N-ый, потратив как можно меньшее количество денег.

Входные данные

Во входном файле INPUT.TXT записано сначала число N (1 ≤ N ≤ 100), затем идет N чисел, i-ое из которых задает стоимость бензина в i-ом городе (все числа целые из диапазона от 0 до 100). Далее идет число M - количество дорог в стране, далее идет описание самих дорог. Каждая дорога задается двумя числами - номерами городов, которые она соединяет. Все дороги двухсторонние (то есть по ним можно ездить как в одну, так и в другую сторону); между двумя городами всегда существует не более одной дороги; не существует дорог, ведущих из города в себя.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - суммарную стоимость маршрута или -1, если добраться невозможно.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 4 1 10 2 15 4 1 2 1 3 4 2 4 3 | 3 |

Пояснение к примеру

Оптимальное решение в примере: из 1-го города поехать в 3-й, а затем в 4-й. Горючее придется покупать в 1 и 3 городах.

# Варіант 37

Предприятие «Авто-2010» выпускает двигатели для известных во всем мире автомобилей. Двигатель состоит ровно из n деталей, пронумерованных от 1 до n, при этом деталь с номером i изготавливается за pi секунд. Специфика предприятия «Авто-2010» заключается в том, что там одновременно может изготавливаться лишь одна деталь двигателя. Для производства некоторых деталей необходимо иметь предварительно изготовленный набор других деталей.

Генеральный директор «Авто-2010» поставил перед предприятием амбициозную задачу – за наименьшее время изготовить деталь с номером 1, чтобы представить ее на выставке.

Требуется написать программу, которая по заданным зависимостям порядка производства между деталями найдет наименьшее время, за которое можно произвести деталь с номером 1.

Входные данные

Первая строка входного файла INPUT.TXT содержит число n (1 ≤ n ≤ 100000) – количество деталей двигателя. Вторая строка содержит n натуральных чисел p1, p2 … pn , определяющих время изготовления каждой детали в секундах. Время для изготовления каждой детали не превосходит 109 секунд.

Каждая из последующих n строк входного файла описывает характеристики производства деталей. Здесь i-ая строка содержит число деталей ki, которые требуются для производства детали с номером i, а также их номера. Сумма всех чисел ki не превосходит 200000.

Известно, что не существует циклических зависимостей в производстве деталей.

Выходные данные

В первой строке выходного файла OUTPUT.TXT должны содержаться два числа: минимальное время (в секундах), необходимое для скорейшего производства детали с номером 1 и число k деталей, которые необходимо для этого произвести. Во второй строке требуется вывести через пробел k чисел – номера деталей в том порядке, в котором следует их производить для скорейшего производства детали с номером 1.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 100 200 300 1 2 0 2 2 1 | 300 2 2 1 |
| 2 | 2 2 3 1 2 0 | 5 2 2 1 |
| 3 | 4 2 3 4 5 2 3 2 1 3 0 2 1 3 | 9 3 3 2 1 |

# Варіант 38

В средние века в замках Европы был популярен следующий вид казни: в лабиринт, в котором находился тигр, заводили раба. Рабу была известна карта лабиринта и его первоначальное расположение. Тигр обладал очень тонким обонянием, то есть он знал, где находится раб в любой момент времени и мог в кратчайшее время настигнуть раба и съесть, если мог.

Дана схема лабиринта в виде таблицы N×M. Вход в лабиринт находится в левой верхней клетке. В этом же месте находится раб в начальный момент времени. Выход из лабиринта находится в правой нижней клетке. Гарантируется, что от входа до выхода существует путь и что тигр находится в свободной клетке лабиринта. Также известно, что лабиринт ограничен сплошной стеной по периметру.

Необходимо определить длину кратчайшего пути раба до выхода, и сможет ли раб гарантированно выбраться из лабиринта живым, если за единицу времени как раб, так и тигр, могут переместиться в соседнюю по стороне клетку в произвольном свободном направлении (то есть туда, где нет стены).

Входные данные

Первая строка входного файла INPUT.TXT содержит два числа: N и M – длина и ширина лабиринта (4 ≤ N, M ≤ 1000). Далее следует N строк по M символов – описание лабиринта. Символ «#» означает стену, а символ «.» - свободное пространство, «T» - положение тигра в начальный момент времени.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите целое число – длину кратчайшего пути раба, во второй строке выведите «Yes», если раб гарантированно сможет добраться до выхода, и «No» в противном случае.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 8 10 ########## #.#...##.# #.#..###.# #.#.##...# #.......## #...###..# #....T#..# ########## | 12 No |
| 2 | 8 10 ########## #.#...##.# #.#..###.# #.#.##...# #.......## #..####..# #....T#..# ########## | 12 Yes |

# Варіант 39

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасать королеву приходится героическим мушкетерам. Атос, Портос, Арамис и д’Артаньян уже перехватили агентов кардинала и вернули украденное; осталось лишь передать подвески королеве Анне. Королева ждет мушкетеров в дворцовом саду. Дворцовый сад имеет форму прямоугольника и разбит на участки, представляющие собой небольшие садики, содержащие коллекции растений из разных климатических зон. К сожалению, на некоторых участках, в том числе на всех участках, расположенных на границах сада, уже притаились в засаде гвардейцы кардинала; на бой с ними времени у мушкетеров нет. Мушкетерам удалось добыть карту сада с отмеченными местами засад; теперь им предстоит выбрать наиболее оптимальные пути к королеве. Для надежности друзья разделили между собой спасенные подвески и проникли в сад поодиночке, поэтому начинают свой путь к королеве с разных участков сада. Двигаются герои по максимально короткой возможной траектории.

Марлезонский балет вот-вот начнется; королева не в состоянии ждать героев больше L минут; ровно в начале L+1-ой минуты королева покинет парк, и те мушкетеры, что не успеют к этому времени до нее добраться, не смогут передать ей подвески. На преодоление одного участка у мушкетеров уйдет ровно по минуте. С каждого участка мушкетеры могут перейти на 4 соседние. Требуется выяснить, сколько подвесок будет красоваться на платье королевы, когда она придет на бал.

Входные данные

Первая строка входного файла INPUT.TXT содержит целые числа N и M (1 ≤ N,M ≤ 20) – размеры сада. Далее идут N строк по M символов в каждом; символы '0' соответствуют участкам, на которых нет засады, символы '1' – участкам, на которых разместились гвардейцы. В N+2-ой строке теста записано три целых числа: координаты участка, на котором королева будет ждать мушкетёров (Qx, Qy) (1 < Qx < N, 1 < Qy < M) и время в минутах до начала балета (1 ≤ L ≤ 1000). В N+3-ей строки записаны через пробел целые числа координаты участка, с которого стартует Атос (Ax,Ay) (1 < Ax < N, 1 < Ay < M) и количество подвесок, хранящихся у него (1 ≤ Pa ≤ 1000). В N+4, N+5 и N+6-ой строках аналогично записаны стартовые координаты и количество подвесок у Портоса, Арамиса и д’Артаньяна.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество подвесок, которое королева успеет получить у мушкетеров до начала балета.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 5 11111 10001 10001 10001 11111 4 4 10 2 2 1 2 3 2 3 2 3 3 3 4 | 10 |
| 2 | 5 5 11111 10001 10111 10101 11111 4 4 10 2 2 1 2 2 2 2 2 3 2 2 4 | 0 |

# Варіант 40

В таблице из N строк и N столбцов некоторые клетки заняты шариками, другие свободны. Выбран шарик, который нужно переместить, и место, куда его нужно переместить. Выбранный шарик за один шаг перемещается в соседнюю по горизонтали или вертикали свободную клетку. Требуется выяснить, возможно ли переместить шарик из начальной клетки в заданную, и, если возможно, то найти путь из наименьшего количества шагов.

Входные данные

В первой строке входного файла INPUT.TXT находится число N, в следующих N строках - по N символов. Символом точки обозначена свободная клетка, английской заглавной O - шарик, @ - исходное положение шарика, который должен двигаться, английской заглавной X - конечное положение шарика. (2 ≤ N ≤ 250)

Выходные данные

В выходной файл OUTPUT.TXT выведите в первой строке «Y», если движение возможно, или «N», если нет. Если движение возможно, то далее следует вывести N строк по N символов - как и на вводе, но буква X, а также все точки по пути следует заменить плюсами. Если решений несколько, выведите любое.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 5 ....X .OOOO ..... OOOO. @.... | Y +++++ +OOOO +++++ OOOO+ @++++ |
| 2 | 5 ..X.. ..... OOOOO ..... ..@.. | N |
| 3 | 5 ...X. ..... O.OOO ..... ....@ | Y ..++. .++.. O+OOO .++++ ....@ |

# Варіант 41

Бригаде строителей поручили уложить квадратной плиткой пол на кухне в виде шахматного узора. Но строители работали не очень слаженно, и когда весь пол уже был уложен, оказалось, что в некоторых местах плитки одинакового цвета граничат друг с другом.

По заданному замощению определите, какое минимальное число строителей могло укладывать плитку.

Входные данные

Входной файл INPUT.TXT содержит восемь строк, состоящих из восьми символов W и B – полученное замощение. Символ W обозначает плитку белого цвета, а символ B – чёрную.

Выходные данные

В выходной файл OUTPUT.TXT нужно вывести одно число - искомое число строителей.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | WBWBWBBW BWBBWBWB WBWWBWBW WBWWBWWB BWBBWBWB WBWBWWBW BWBWBBWB WBWBWWBW | 4 |

# Варіант 42

Произошло радиоактивное заражение местности. Составлена карта зараженности. Она представляет собой прямоугольную таблицу N×M, в клетках которой записана зараженность соответствующего участка.

Требуется написать программу, которая найдет путь из левой верхней клетки таблицы в правую нижнюю клетку с минимальной суммарной дозой радиации.

Входные данные

Входной файл INPUT.TXT содержит в первой строке числа N и M, а в следующих N строках – по M чисел – карта зараженности местности. Числа в строках разделяются одним пробелом. 1 ≤ N ≤ 30, 1 ≤ M ≤ 30, зараженность участка – целое число от 0 до 100.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно число – суммарную долю радиации.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 5 2 100 0 100 100 1 100 0 0 0 1 0 3 100 2 | 9 |

# Варіант 43

Вам дана табличка, состоящая из N строк и M столбцов. В каждой клетке таблицы стоит либо 0, либо 1. Расстоянием между клетками (x1,y1) и (x2,y2) называется |x1-x2|+|y1-y2|. Вам нужно построить другую таблицу, в которой в каждой клетке стоит расстояние от данной до ближайшей клетки, содержащей 1 (в начальной таблице). Гарантируется, что хотя бы одна 1 в таблице есть.

Входные данные

В первой строке входного файла INPUT.TXT содержатся два натуральных числа, не превосходящих 100 - N и M. Далее идут N строк по M чисел - элементы таблицы.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать N строк по M чисел - элементы искомой таблицы.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 2 3 0 0 1 1 0 0 | 1 1 0 0 1 1 |

# Варіант 44

Дана матрица A размером N×N, заполненная неотрицательными целыми числами. Расстояние между двумя элементами Ai j и Ap q определено как |i - p| + |j - q|. Требуется заменить каждый нулевой элемент матрицы ближайшим ненулевым. Если есть две или больше ближайших ненулевых ячейки, нуль должен быть оставлен.

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число N (N ≤ 200). Затем идут N строк по N чисел, разделённых пробелами и представляющих собой матрицу. Элементы матрицы не превосходят значения 106.

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк по N чисел, разделённых пробелами, - модифицированную матрицу.

Пример

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 3 0 0 0 1 0 2 0 3 0 | 1 0 2 1 0 2 0 3 0 |

# Варіант 45

Имеется три ведра, емкости которых известны и не равны. Самое большое ведро полное, остальные пусты. Требуется добиться, чтобы в самом большом ведре был заданный объем воды. За один шаг вода переливается из одного ведра в другое до тех пор, пока либо не закончится вода в ведре-источнике, либо не наполнится доверху вода в ведре-получателе.

Школьник Василий, чтобы занять себя, пытается решать эту задачу с разными входными данными, но не всегда находит решение. И даже если решение найдено, он хочет знать, является ли найденное решение оптимальным, а именно, используется ли минимальное количество шагов. Требуется написать программу, которая поможет Василию проверить его решение.

Входные данные

Во входном файле INPUT.TXT записаны 4 числа: емкости ведер B1, B2, B3 (1000 ≥ B1 > B2 > B3 > 0) и требуемое количество воды T в первом ведре (B1 > T > 0).

Выходные данные

В выходной файл OUTPUT.TXT выведите либо минимальное количество переливаний, либо если задача не имеет решения, то слово IMPOSSIBLE.

Примеры

|  |  |  |
| --- | --- | --- |
| **№** | **INPUT.TXT** | **OUTPUT.TXT** |
| 1 | 10 8 4 4 | 3 |
| 2 | 10 8 4 5 | IMPOSSIBLE |

# Варіант 46

Мэр города Гадюкино решил проверить состояние дорог после только что проведенного капитального ремонта. Для этого он хочет проехать по каждой дороге в обоих направлениях. Помогите мэру составить кратчайший маршрут, проходящий по каждой дороге в каждом направлении хотя бы один раз.

В городе Гадюкино n перекрестков и m дорог, каждая из которых соединяет два различных перекрестка. Между двумя перекрестками может быть не более одной дороги. Известно, что по дорогам от каждого перекрестка можно доехать до любого другого.

Входные данные

Входной файл INPUT.TXT содержит целые числа n и m (1 ≤ n ≤ 104, 1 ≤ m ≤ 105), и далее m пар целых чисел ai и bi - номера перекрестков, которые соединяет i-я дорога.

Выходные данные

В выходной файл OUTPUT.TXT выведите число s - минимальную длину пути и далее s+1 число - номера перекрестков в том порядке, в котором их нужно проезжать.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 3 1 2 2 3 1 3 | 6 1 2 3 1 3 2 1 |

# Варіант 47

Мальчику Севе очень нравится цвет индиго (это такой темно-синий цвет). Сева всячески старается окружить себя вещами этого цвета.

Скоро Новый год, и Сева решил изготовить гирлянду с лампочками, чтобы украсить ей свою комнату. Он купил n патронов для лампочек и соединил (n−1)-им проводом так, что в гирлянде не образовалось ни одного замкнутого контура. Его гирлянда, таким образом, с точки зрения математики, представляет собой дерево. Для того чтобы гирлянда была готова, осталось совсем немного надо только добавить провод для подключения гирлянды в электрическую сеть и вкрутить разноцветные лампочки в патроны.

У Севы есть лампочки трех разных цветов: синего, фиолетового и индиго. При этом, в некоторые патроны из соображений красоты нельзя устанавливать лампочки определенных цветов. Также, из соображений красоты, в гирлянде не должно быть двух лампочек одного цвета, напрямую соединенных проводом. Разумеется, Сева хочет, чтобы в гирлянде было как можно больше лампочек его любимого цвета.

Помогите Севе. Напишите программу, которая найдет максимальное количество лампочек цвета индиго, которые можно установить в собранную Севой гирлянду.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n количество патронов для лампочек в Севиной гирлянде (1 ≤ n ≤ 50000). Последующие (n−1) строка содержат каждая по два числа: ui и vi (1 ≤ ui, vi ≤ n) – номера патронов, соединенных соответствующим проводом.

Далее следуют n строк с описанием патронов. Каждая из них не пуста и состоит не более, чем из трех различных символов «I», «B» или «V». i-ая из этих строк описывает i-ый патрон, а именно: если она содержит символ «I», то в i-ый патрон можно устанавливать лампочку цвета индиго, «B» - можно устанавливать лампочку синего цвета, «V» - можно устанавливать лампочку фиолетового цвета.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число: ответ на задачу. Если же при наложенных ограничениях Севе вообще не удастся собрать гирлянду, выведите число −1.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 1 2 2 3 IB IV IB | 2 |

# Варіант 48

От вас требуется определить вес минимального остовного дерева для неориентированного взвешенного связного графа.

Входные данные

В первой строке входного файла INPUT.TXT находятся числа N и M (1 ≤ N ≤ 100; 1 ≤ M ≤ 6000), где N - количество вершин в графе, а M - количество рёбер. В каждой из последующих M строк записано по тройке чисел A, B, C, где A и B - номера вершин, соединённых ребром, а C - вес ребра (натуральное число, не превышающее 30000).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - искомый вес.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 3 1 2 1 2 3 2 3 1 3 | 3 |

# Варіант 49

В государстве алхимиков есть N населённых пунктов, пронумерованных числами от 1 до N, и M дорог. Населённые пункты бывают двух типов: деревни и города. Кроме того, в государстве есть одна столица (она может располагаться как в городе, так и в деревне). Каждая дорога соединяет два населённых пункта, и для проезда по ней требуется Ti минут. В столице было решено провести 1-ю государственную командную олимпиаду по алхимии. Для этого во все города из столицы были отправлены гонцы (по одному на город) с информацией про олимпиаду.

Напишите программу, которая посчитает, в каком порядке и через какое время каждый из гонцов доберётся до своего города. Считается, что гонец во время пути нигде не задерживается.

Входные данные

Во входном файле INPUT.TXT сначала записаны 3 числа N, M, K — количество населенных пунктов, количество дорог и количество городов (2 ≤ N ≤ 1000, 1 ≤ M ≤ 10000, 1 ≤ K ≤ N). Далее записан номер столицы C (1 ≤ C ≤ N). Следующие K чисел задают номера городов. Далее следуют M троек чисел Si, Ei, Ti, описывающих дороги: Si и Ei — номера населенных пунктов, которые соединяет данная дорога, а Ti — время для проезда по ней (1 ≤ Ti ≤ 100).

Гарантируется, что до каждого города из столицы можно добраться по дорогам (возможно, через другие населенные пункты) и то, что любые два населенных пункта соединены не более, чем одной дорогой.

Выходные данные

В выходной файл OUTPUT.TXT выведите K пар чисел: для каждого города должен быть выведен его номер и минимальное время, когда гонец может в нем оказаться (время измеряется в минутах с того момента, как гонцы выехали из столицы). Пары в выходном файле должны быть упорядочены по времени прибытия гонца, а в случае совпадения времени следует сортировать эти пары по номерам городов.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 5 4 5 1 1 2 3 4 5 1 2 1 2 3 10 3 4 100 4 5 100 | 1 0 2 1 3 11 4 111 5 211 |
| 2 | 5 5 3 1 2 4 5 2 1 1 2 3 10 3 4 100 4 5 100 1 5 1 | 2 1 5 1 4 101 |

# Варіант 50

Во многих видах спортивных соревнований имеются различные ритуалы, направленные для примирения конкурирующих команд или игроков. Это может быть рукопожатие, поклон или даже разбрызгивание шампанского. ACM (Alliance of Chess Masters) собирается создать свой собственный ритуал, шахматную миниигру, в которой участвует два игрока в союзе друг с другом (а не как обычно, друг против друга). Игра проходит на шахматной доске размером 3×3, у каждого из игроков имеется два шахматных коня, которых они должны переместить так, чтобы добраться от одной позиции до другой (игроки могут делать шаги в любом порядке, не обязательно по очереди). При этом два коня не могут занимать одну и ту же клетку.

Стартовое и завершающее положения определены судьей. Оказывается, что некоторые такие задачи являются более трудными, чем другие, а некоторые могут быть даже неразрешимыми - поэтому, некоторые игроки иногда неспособны закончить ритуал. Ваша задача состоит в том, чтобы написать программу, которая по заданным положениям начальной и конечной расстановки, сможет определить возможность успешного окончания игры, а в случае успеха сможет так же определить сложность задачи – минимально возможное количество ходов, необходимых для разрешения данной задачи.

Входные данные

Входной файл INPUT.TXT содержит 3 строки по 7 символов в каждой. Первые 3 символа каждой строки описывают соответствующую строку шахматной строки для начальной позиции, затем идет пробел и замыкающие 3 символа, описывающие аналогичным образом строку конечной позиции. Белый конь обозначается символом «W», а черный – символом «B», пустые клетки помечаются «.» (точка).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – сложность задачи. Если задача не имеет решения, то следует вывести -1.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | WBB ..W W.. ..W ... .BB | 4 |
| 2 | ..B ..B W.B ..B W.. WW. | -1 |

# Варіант 51

Внутри адронного коллайдера образовалось N разновидностей новых частиц в количестве A1, … ,AN единиц каждая. Большая часть новых частиц, однако, успевает прореагировать между собой раньше, чем эти частицы сможет уловить детектор, поэтому физикам очень важно предсказать конечный итог реакции между частицами.

Для простоты будем считать, что в реакции участвуют 2 частицы, с одним из следующих результатов, в зависимости от типа:

Первая частица уничтожает вторую

Частицы отскакивают друг от друга без какого-либо вреда

Необходимо определить все возможные исходы эксперимента.

Входные данные

В первой строке входного файла INPUT.TXT задано количество видов частиц N (1 ≤ N ≤ 4). В следующей строке записаны N чисел, определяющие начальное количество частиц каждого типа Ai (1 ≤ Ai ≤ 2). Следующие N строк формируют матрицу B[N][N] из чисел 0 и 1. Ненулевое значение ячейки Bij указывает, что частица типа i при столкновении уничтожает частицу типа j. Если Bij не ноль и Bji не ноль, то в результате взаимодействия уничтожается либо частица i, либо частица j, но не обе сразу.

Выходные данные

В первой строке выходного файла OUTPUT.TXT следует вывести количество возможных исходов K. В каждой из последующих K строк должно содержаться описание исхода эксперимента в формате исходного файла.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 1 1 2 0 0 1 1 0 0 1 1 1 | 3 0 1 0 0 0 1 1 0 0 |
| 2 | 1 2 0 | 1 2 |

# Варіант 52

Сема — любитель игр на шахматной доске. Больше всего он любит играть в шашки. Сема решил написать программу, которая будет играть в шашки. По его задумке, она будет показывать, какие шашки можно взять на данном ходу. Ваша задача — реализовать эту функцию.

Напомним, что взять можно только по диагонали одним из четырех способов:

A picture containing star, night sky

Description automatically generated

После того, как белая шашка перемещается на пустое поле, черная шашка снимается с доски и считается взятой. При этом, если после перемещения белой шашки, у нее вновь появляется возможность взять, то она продолжает свой ход. Аналогичны правила и для черных шашек.

Отметим, что в рассматриваемом варианте игры в шашки отсутствует понятие «дамка», то есть возможности шашки по взятию не зависят от того, доходила она до последней горизонтали или нет.

Входные данные

В первых восьми строках входного файла INPUT.TXT записаны по восемь символов из множества {«.», «B», «W»}, которые обозначают пустое поле, черную шашку и белую шашку соответственно. Во входном файле не более 12 шашек каждого цвета. Все шашки расположены либо на черных, либо на белых полях.

Выходные данные

В выходной файл OUTPUT.TXT выдайте поля, на которых стоят шашки, которые можно взять, если ходят белые или черные. Используйте формат вывода аналогичный примерам.

Отсортируйте поля сначала по первой координате (измеряется по вертикали), а при равенстве первых – по второй (измеряется по горизонтали). Учитывайте, что первый символ первой строки входного файла соответствует полю (1, 1), последний символ первой строки – полю (1, 8), первый символ последней строки – полю (8, 1), последний символ последней строки – (8, 8).

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | .W.W.W.W W.W.W.W. .W.W.W.W ........ ........ B.B.B.B. .B.B.B.B B.B.B.B. | White: 0 Black: 0 |
| 2 | .W.W.W.W W.W.W.W. .....W.. W.W...W. .B.B.... B...B.B. .B.B.B.B B.B...B. | White: 3 (5, 2), (5, 4), (7, 4) Black: 1 (4, 3) |

Пояснение

В первом примере никакая шашка не может брать.

Во втором примере белая шашка, стоящая на (4, 1) может взять сначала черную шашку на поле (5, 2) и, переместившись на поле (6, 3), взять черную шашку, стоящую на поле (5, 4). Также с поля (6, 3) можно взять еще одну черную шашку, стоящую на поле (7, 4), при этом придется двигаться другим путем. Черные же могут взять лишь белую шашку, которая стоит на поле (4, 3).

# Варіант 53

Карту местности условно разбили на квадраты, и посчитали среднюю высоту над уровнем моря для каждого квадрата.

Когда идет дождь, вода равномерно выпадает на все квадраты. Если один из четырех соседних с данным квадратом квадратов имеет меньшую высоту над уровнем моря, то вода с текущего квадрата стекает туда (и, если есть возможность, то дальше), если же все соседние квадраты имеют большую высоту, то вода скапливается в этом квадрате.

Разрешается в некоторых квадратах построить водостоки. Когда на каком-то квадрате строят водосток, то вся вода, которая раньше скапливалась в этом квадрате, будет утекать в водосток.

Если есть группа квадратов, имеющих одинаковую высоту и образующих связную область, то если хотя бы рядом с одним из этих квадратов есть квадрат, имеющий меньшую высоту, то вся вода утекает туда, если же такого квадрата нет, то вода стоит во всех этих квадратах. При этом достаточно построить водосток в любом из этих квадратов, и вся вода с них будет утекать в этот водосток.

Требуется определить, какое минимальное количество водостоков нужно построить, чтобы после дождя вся вода утекала в водостоки.

Входные данные

Во входном файле INPUT.TXT записаны сначала числа N и M, задающие размеры карты — натуральные числа, не превышающие 100. Далее, идет N строк, по M чисел в каждой, задающих высоту квадратов карты над уровнем моря. Высота задается натуральным числом, не превышающим 10000. Считается, что квадраты, расположенные за пределами карты, имеют высоту 10001 (то есть вода никогда не утекает за пределы карты).

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное количество водостоков, которое необходимо построить.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 5 1 2 3 1 10 1 4 3 10 10 1 5 5 5 5 6 6 6 6 6 | 2 |

# Варіант 54

Для исследования поверхности Марса ученым необходимо разработать систему оповещения, которая смогла бы передавать информацию по цепному принципу между городами, которые планируется там построить.

При этом в каждом городе необходимо построить радиостанцию таким образом, чтобы была связь между всеми городами. При этом все такие станции должны передавать сигнал друг другу на равном расстоянии R. Таким образом, будет возможна передачи информации из одного города в другой только тогда, когда расстояние между ними не более R.

По заданным координатам городов, в целях экономии энергии радиостанций, Вам следует определить минимальное значение R, при котором информация сможет быть доставлена из любого города во все остальные.

Входные данные

В первой строке входного файла INPUT.TXT сначала записано натуральное число N – количество городов (N ≤ 1000). Далее идет N строк, содержащих вещественные координаты (Xi, Yi) соответствующего города. (-10000 ≤ Xi, Yi ≤ 10000). Предполагается, что все города находятся на плоскости.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно вещественное число – наименьший радиус радиостанций. Число следует вывести с двумя знаками после запятой, без лидирующих нулей, в формате, приведенном в примерах.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 0 0 2 0 0 2 2 3 | 2.24 |
| 2 | 3 2 0 0 2 4 2 | 2.83 |

# Варіант 55

На стандартной шахматной доске (8х8) живут 2 шахматных коня: красный и зеленый. Обычно они беззаботно скачут по просторам доски, пощипывая шахматную травку, но сегодня особенный день: у зеленого коня день рождения. зеленый конь решил отпраздновать это событие вместе с красным. Но для осуществления этого прекрасного плана им нужно оказаться на одной клетке. Заметим, что красный и зеленый шахматные кони сильно отличаются от черного с белым: они ходят не по очереди, а одновременно, и если оказываются на одной клетке, никто никого не съедает. Сколько ходов им потребуется, чтобы насладиться праздником?

Входные данные

Во входном файле INPUT.TXT содержатся координаты коней, записанные по стандартным шахматным правилам (т.е. двумя символами - маленькая английская буква (от a до h) и цифра (от 1 до 8), задающие столбец и строку соответственно).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать наименьшее необходимое количество ходов, либо -1, если кони не могут встретиться.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | a1 a3 | 1 |

# Варіант 56

Все участники олимпиад знают, что во время соревнования на счету каждая секунда. Иногда даже время, которое в суете затрачивается на переключение между окнами может оказаться критичным. В таких соревнованиях, как Java Challenge, количество окон может быть довольно большим (Java Challenge - это соревнование, проходящее в рамках финала чемпионата мира по программированию среди студентов. Оно состоит в разработке искусственного интеллекта для управления виртуальным роботом).

В данной задаче мы будем считать, что этот процесс выполняется следующим способом. В системе хранится циклический список открытых окон. При нажатии определенной комбинации клавиш k раз можно перейти в этом списке на k позиций в одну или в другую сторону. Кроме того, окна, относящиеся к каждому приложению так же организованы в циклический список. По этому списку также можно перемещаться в обе стороны, для перемещения на k позиций так же требуется k нажатий клавиш. При этом после своей активизации окно перемещаются в позицию перед первым элементом общего списка окон. Напишите программу, которая для каждого из окон будет определять минимальное количество нажатий клавиш, которое нужно затратить для его активизации. До начала выполнения операции активным является первое окно.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n (1 ≤ n ≤ 50000) - количество открытых окон. Следующие n строк описывают окна в том порядке, в котором они идут в списке. Для каждого из окон задается номер приложения, которому соответствует это окно, и его номер в циклическом списке окон этого приложения.

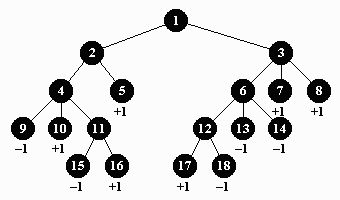
Выходные данные

На единственной строке выходного файла OUTPUT.TXT для каждого окна выведите минимальное количество нажатий клавиш, которое надо затратить для его активации.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 9 3 2 2 2 2 3 1 3 2 1 3 1 4 1 1 2 1 1 | 0 1 2 2 2 1 3 2 1 |

# Варіант 57



Игра для двух игроков определяется её деревом. Соперники делают ходы по очереди. Первый игрок начинает игру. Игра кончается или вничью, или победой одного из игроков. Листья дерева этой игры могут иметь значения, равные одному из трёх чисел: +1 - победа первого игрока, -1 - победа второго игрока, 0 - ничья. Ваша задача - определить, кто выиграет, если оба противника следуют правильной стратегии.

Входные данные

Узлы дерева пронумерованы последовательными целыми числами. Корень дерева всегда имеет номер 1. Первая строка входного файла INPUT.TXT содержит натуральное N - число узлов в дереве игры (N ≤ 1000). Следующая N - 1 строка описывает узлы - одна строка для каждого узла (за исключением первого). Вторая строка содержит описание второго узла дерева, третья - третьего узла и т.д. Если узел является листом, первый символ строки - L, затем идёт пробел, затем номер родительского узла, ещё пробел и результат игры (+1 - победа первого игрока, -1 - победа второго, 0 - ничья). Если узел внутренний, то строка содержит N - первый символ, затем пробел и номер родительского узла.

Выходные данные

В выходной файл OUTPUT.TXT выведите +1, если выигрывает первый игрок, -1, если второй, и 0 - в случае ничейного исхода.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 7 N 1 N 1 L 2 -1 L 2 +1 L 3 +1 L 3 +1 | +1 |
| 2 | 7 N 1 N 1 L 2 -1 L 2 +1 L 3 +1 L 3 0 | 0 |
| 3 | 18 N 1 N 1 N 2 L 2 +1 N 3 L 3 +1 L 3 +1 L 4 -1 L 4 +1 N 4 N 6 L 6 -1 L 6 -1 L 11 -1 L 11 +1 L 12 +1 L 12 -1 | +1 |

# Варіант 58

Недавно Билл устроился на работу полицейским. Теперь ему предстоит каждый вечер обходить свой участок, который представляет собой прямоугольник, состоящий из N×M кварталов. Каждый квартал имеет вид квадрата размером 100 х 100 метров, кварталы отделены друг от друга прямыми улицами.

Таким образом, через участок Билла проходит N+1 улица, идущая с запада на восток, и M+1 улица, идущая с севера на юг. Перекрестки разбивают улицы на (N+1)\*M + (M+1)\*N отрезков, каждый из которых имеет длину 100 метров.

Совершая обход, Билл выходит из полицейского управления, расположенного около юго-западного угла его участка, обходит свой участок и возвращается в управление. Во время обхода Билл должен пройти по каждому отрезку улицы на территории своего участка как минимум один раз. Известно, что во время обхода Билл проходит отрезок длиной 100 метров за одну минуту. Выясните, какое минимальное число минут потребуется Биллу, чтобы совершить обход.

Входные данные

Входной файл INPUT.TXT содержит натуральные числа N и M, не превышающие 10 000.

Выходные данные

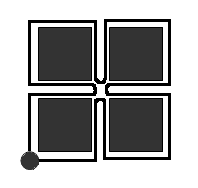
В выходной файл OUTPUT.TXT выведите минимальное время, за которое Билл может совершить обход.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 1 1 | 4 |
| 2 | 2 2 | 16 |
| 3 | 4 3 | 38 |

Пояснение

Один из возможных оптимальных путей для Билла во втором примере показан на рисунке:



# Варіант 59

В спортзале размером N×M метров построили современный аттракцион под названием "Левый лабиринт". Для этого на полу спортзала с интервалом в 1 метр начертили линии, параллельные стенам спортзала. Таким образом, спортзал разбили на N×M клеток. Дальше некоторые из этих клеток покрасили в черный цвет. Аттракцион заключается в том, что участника ставят в некоторой клетке спортзала и просят как можно быстрее добежать до некоторой другой клетки. При этом накладываются следующие условия:

Участнику запрещено ходить по черным клеткам.

Придя в какую-то клетку, участник может пойти либо прямо, либо налево, либо направо (если в соответствующем направлении клетка не покрашена в черный цвет): ходить назад, а также ходить по диагонали запрещается.

За все время пути участнику разрешается повернуть направо (то есть пойти из текущей клетки направо относительно того, откуда он пришел в данную клетку) не более K раз.

В начальной клетке участник может встать лицом в ту сторону, в какую ему захочется. С какой стороны участник прибежит в конечную клетку также не важно.

Известно, что на то, чтобы перебежать из клетки в соседнюю, участник тратит ровно 1 секунду. Требуется вычислить минимальное время, за которое участник сможет достичь конечной клетки.

Входные данные

Во входном файле INPUT.TXT сначала записано число K — количество разрешенных поворотов направо (целое число из диапазона от 0 до 5), затем записаны числа N и M, задающие размеры спортзала — натуральные числа, не превышающие 20. Далее записано N строк по M чисел в каждой. Число 0 обозначает неокрашенную клетку, число 1 — покрашенную, число 2 — клетку, откуда стартует участник и число 3 — клетку, куда нужно добежать (клетки, помеченные 2 и 3 являются неокрашенными). В лабиринте всегда есть ровно одна начальная клетка и ровно одна клетка, в которую нужно попасть.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное время, за которое можно добраться в конечную клетку. Если попасть в конечную клетку нельзя, выведите –1.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 1 5 5 2 0 0 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0 1 3 1 0 0 1 | 12 |
| 2 | 0 1 3 2 1 3 | -1 |

# Варіант 60

Мальчик Миша собирается участвовать в школьных соревнованиях по гонкам с препятствиями по версии «Формула 001». Но к любым соревнованиям необходимо готовиться. Младший брат Миши, Ральф, не так силен в гонках с препятствиями, как его старший брат, но зато он обладает незаурядной фантазией. Он решил помочь брату и придумал игру, играя в которую, можно существенно увеличить свой опыт в вождении гоночного автомобиля.

Игра состоит в следующем. Пусть у нас есть бесконечное игровое поле, покрытое бесконечной квадратной сеткой. Некоторое множество узлов этой сетки отмечено. Назовем это множество S. В игре участвуют несколько игроков. У каждого игрока есть машина – фигура, которая может находиться только в узлах из множества S. В начале игры все машины находятся в различных начальных узлах. Ход каждого игрока состоит в перемещении машины в некоторый узел из множества S, возможно, тот же самый. Цель игры – добраться до определенного, финишного узла первым.

Ход происходит по следующим правилам. Пусть на предыдущем шаге машина была перемещена на вектор (X, Y) (если это первый шаг, то X=0 и Y=0). Тогда за один текущий ход машину можно передвинуть на один из следующих векторов: (X−1, Y−1), (X−1, Y), (X−1, Y+1), (X, Y−1), (X, Y), (X, Y+1), (X+1, Y−1), (X+1, Y) и (X+1, Y+1).

Конечно, на какой-либо из этих векторов машину переместить можно только при том условии, что после этого она попадет в узел из множества S. Если ход сделать нельзя, то игрок считается проигравшим и выбывает из игры.

Некоторое время поиграв в эту игру, Миша и Ральф занялись ее анализом. В данный момент они хотят узнать, за какое наименьшее число ходов из стартового узла возможно попасть в финишный. Сами они эту задачу решить не смогли и обратились за помощью к Вам. Помогите им!

Входные данные

В первой строке входного файла INPUT.TXT находится N – число элементов множества S. 2 ≤ N ≤ 1000. В последующих N строках находятся координаты узлов из этого множества – целые числа Xi, Yi (−109 ≤ Xi, Yi ≤ 109).

Никакие два узла во входном файле не совпадают. Занумеруем эти узлы, начиная с 1, в порядке их описания во входном файле. Стартовым узлом будет являться узел с номером 1, финишным узел с номером N .

Выходные данные

В выходной файл OUTPUT.TXT выведите -1, если добраться до финишного узла невозможно. Иначе, в первой строке выведите минимальное требуемое число ходов K . Во второй выведите K+1 число – номера посещенных узлов в порядке посещения. Первым узлом должен быть узел с номером 1, последним – узел с номером N .

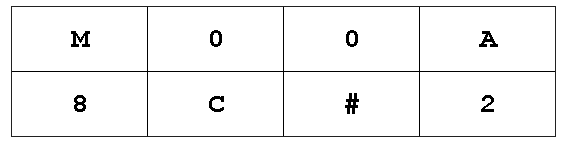
Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 0 0 0 1 0 2 | 2 1 2 3 |
| 2 | 3 0 0 0 2 0 3 | -1 |

# Варіант 61

Всем известна игра «Пятнашки», где надо выстроить изначально неупорядоченную последовательность чисел, перемещая фишки с нанесёнными числами от 1 до 15 в квадрате 4×4. На основе данной игры была разработана другая – поле в ней лишь 4×2 клетки, на поле 7 фишек, но на фишках изображены буквы английского алфавита и арабские цифры (на каждой фишке – один символ, но на разных фишках могут быть одинаковые символы). Цель игры прежняя – упорядочить в соответствии с образцом стартовую расстановку фишек за минимальное количество ходов.

Свободная клетка обозначается специальным символом «#» и используется для перемещения фишек по полю. Перемещать фишки на свободную клетку разрешается из соседних клеток, имеющих общую грань со свободной. Например, на рисунке более правый символ «0» можно переместить вниз на свободную клетку, тогда «0» будет в нижней клетке, а пустой станет верхняя клетка, либо в свободную клетку переместить букву «C» или цифру «2».



Входные данные

Входной файл INPUT.TXT содержит четыре строки: две первые строки содержат стартовую комбинацию символов, следующие две - образец. Каждая строка содержит 4 символа (английский алфавит и арабские цифры), пустая клетка обозначается символом «#» (решетка).

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное количество перемещений, необходимых для получения искомой комбинации. Если нужную комбинацию получить нельзя, выведите число -1.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | ACM8 002# ACM# 2008 | 17 |
| 2 | rogp mar# prog ram# | 26 |

# Варіант 62

Возможно, некоторым из вас знакома игра Zuma о приключениях лягушки. В данной задаче правила похожи и довольно просты: в каменном жёлобе находится ряд разноцветных шаров; пушка, расположившаяся рядом с жёлобом, имеет некоторый запас разноцветных шаров и периодически закидывает их в желоб. Заброшенные шары встраиваются в ряд. Если после выстрела в желобе образуется непрерывная последовательность из трех или более шаров одного цвета, включающая заброшенный шар, то они исчезают, а соседние шары сдвигаются, смыкая ряд. Если после исчезновения шаров в месте стыка присутствуют соседние шары (как слева, так и справа), образующие непрерывную последовательность из трех или более шаров одного цвета, то они также исчезают, и так далее. Цель игры – уничтожить все шары.

|  |  |  |
| --- | --- | --- |
| Этап | Рисунок | Пояснение |
| 1 | A picture containing text  Description automatically generated | Выстреливается новый шар «B», в позицию после шара №1 |
| 2 | A picture containing text, watch  Description automatically generated | После выстрела новый шар образует с соседними последовательность цвета «B», в позициях 2-5. Длина последовательности ≥3, поэтому шары 2-5 исчезнут |
| 3 | A picture containing text  Description automatically generated | Оставшиеся шары займут позиции 1-3, и поскольку новая последовательность цвета «А» длины ≥3, она тоже исчезнет |

Пронумеруем шары слева направо, начиная с единицы. Выстрел шара в позицию n означает, что он появится правее шара с номером n и окажется в позиции n+1. Номера шаров, расположенных правее прилетевшего шара, увеличиваются на единицу. Приземление шара левее всего ряда обозначается позицией с номером 0. После исчезновения некоторых шаров, шары в желобе нумеруются заново слева направо, начиная с единицы.

Требуется написать программу, определяющую оптимальную стратегию стрельбы. Оптимальной стратегией называется та, при которой наименьшее количество выстрелов приводит к исчезновению всех шаров.

Входные данные

Входной файл INPUT.TXT содержит описание ряда шаров, цвет каждого шара описывается заглавной буквой английского алфавита (A..Z). Известно, что длина ряда не превышает 14 шаров, а для уничтожения ряда требуется не более 10 выстрелов, если следовать оптимальной стратегии.

Выходные данные

В выходной файл OUTPUT.TXT выведите строку: сначала минимальное количество выстрелов, затем через пробел пары буква-число: цвет шара и позицию выстрела. Выстрелы в ответе должны быть перечислены в порядке их следования в игре. В случае наличия нескольких оптимальных стратегий выберите любую.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | ABBBAA | 1 B1 |
| 2 | ACMNEERC | 10 A0 A0 C0 M2 M2 N2 N2 E2 R2 R2 |
| 3 | BAAA | 3 B0 B0 A0 |

# Варіант 63

В одной замечательной стране живут замечательные люди. По исследованиям замечательного правительства, большинство граждан на выходных садятся в машину, выбирают циклический маршрут между некоторыми городами и деревнями без повторяющихся населенных пунктов и катаются по этому маршруту, пока не надоест. Некоторые, правда, катаются по своему городу и никуда не выезжают.

Так как правительство заботится о своих гражданах, оно хочет сделать их выходные максимально красочными. По этой причине недавно было принято решение покрасить каждую дорогу между населенными пунктами в какой-нибудь цвет. Причем так, чтобы ни на каком "выходном" маршруте не было дорог одинакового цвета. Но так как цветов могло понадобиться довольно много, правительство решило минимизировать количество различных цветов. Вам предстоит помочь этому замечательному государству в осуществлении его планов.

Входные данные

В первой строке находятся два числа: 1 ≤ n ≤ 50 000 - количество городов и деревень в стране и 1 ≤ m ≤ 100 000 - количество дорог. В m последующих строках находится по два числа – номера населенных пунктов, концов дороги. Ни одна дорога не ведет из города в себя, и между двумя населенными пунктами не может быть более одной дороги. Все дороги двусторонние.

Выходные данные

В первой строке выведите минимальное количество цветов. В последующих m строках выведите по три числа: два конца дороги в любом порядке и ее цвет. Дороги разрешается выводить в произвольном порядке.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 2 1 1 2 | 1 2 1 1 |
| 2 | 3 2 1 2 2 3 | 1 2 3 1 1 2 1 |
| 3 | 3 3 1 2 2 3 3 1 | 3 1 2 1 3 2 2 3 1 3 |

# Варіант 64

В парке флоры и фауны затеяли масштабное переустройство. Организаторы запланировали расширение площади парка, увеличение количества экзотических животных и строительство новых вольеров. После утверждения плана строители и зоологи принялись за работу.

Зоологи со своей задачей справились: привезли новых жирафов, долгожданных слонов, игуан c карибских островов и многих других животных и птиц. А вот строители не успели достроить новые вольеры, поэтому привезенных животных было решено временно разместить в клетках.

Однако и эта задача оказалось непростой, так как клеток может не хватить для привезенных животных. А в одну клетку можно поместить только совместимых животных. Зоологи составили таблицу совместимости животных, представив ее в виде матрицы A={aij} размером N×N. Если животные с номерами i и j совместимы, то aij=0, а если - нет, то aij=1. Необходимо определить минимальное количество клеток для безопасного размещения животных, когда во всех клетках находятся только совместимые между собой животные. При этом в клетке может находиться одно, два и более животных.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N - количество животных (1 ≤ N ≤ 18). Далее идет N строк по N чисел в каждой – матрица совместимости животных.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – минимальное количество клеток, необходимое для безопасного размещения животных.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 5 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 | 5 |
| 2 | 1 0 | 1 |

# Варіант 65

Давным давно в одном королевстве правил мудрый король. В том королевстве было n городов, любые два из которых были соединены дорогой. С целью безопасности дорожного движения, по каждой дороге разрешалось перемещаться только в одну сторону.

Король любил свое королевство. Каждый год он k раз осуществлял осмотр королевства. Каждый осмотр начинался в столице, затем король, перемещаясь по дорогам, посещал некоторые города, и, наконец, прибывал в курортный город на берегу моря, где он отдыхал после нелегкой работы. Таким образом король посещал каждый город в королевстве ровно один раз в течении года (за исключением столицы, где каждый осмотр начинался, и морского курорта, где каждый осмотр заканчивался).

Но годы уходили, и король становится стар. И ему было все сложнее делать эти k осмотров. Так что однажды он позвал своего министра транспорта и приказал ему сделать новую программу для осмотра королевства. Осмотр должен начинаться в столице и заканчиваться на морском курорте. Однако теперь весь этот путь должен посещать все города, которые ранее посещались в процессе k осмотров. И более того, если ранее некоторые города A и B посещались в процессе одного осмотра и причем A посещался до B, то новый осмотр также должен посещать A до B.

Помогите королю сделать осмотр королевства менее утомительным, разработайте новый маршрут для осмотра.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа: n и k (3 ≤ n ≤ 400, 1 ≤ k ≤ 100). Следующие n строк описывают дороги. Каждая строка содержит n символов, i-й символ i-й из этих строк равен '.'. Для всех j≠i j-й символ i-й строки равен '+', если дорога идет из j-го города в i-й, либо '-', если дорога идет из i-го города в j-й. Столица имеет номер 1, морской курорт имеет номер n.

Следующие k строк содержат описание маршрутов, по которым король традиционно осуществлял осмотр. Каждый маршрут описывается на одной строке. Описание содержит номера городов в том порядке, в котором король посещал города в процессе соответствующего осмотра. Гарантируется, что каждый осмотр начинается в столице, заканчивается в морском курорте, и следует по всем дорогам в корректном направлении. Каждый город, за исключением столицы и морского курорта, посещается ровно в одном маршруте. Ни один маршрут не идет непосредственно из столицы в морской курорт. Ни один маршрут не посещает в качестве промежуточного города столицу или морской курорт.

Выходные данные

В выходной файл OUTPUT.TXT выведите n чисел – порядок, в котором следует посещать города. Если выполнить требования короля невозможно, выведите −1.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 7 2 .------ +.+---- +-.-+++ +++.--- ++-+.-- ++-++.- ++-+++. 1 3 4 7 1 2 5 6 7 | 1 3 2 4 5 6 7 |

# Варіант 66

Как известно, при распространении радиоволн возникает интерференция, поэтому если рядом расположены две радиопередающие станции, вещающие на одной и той же частоте, то качество радиопередач резко снижается.

Радиостанция «Радио Информатика» планирует транслировать свои программы в стране Флатландия. Министерство связи Флатландии выдало радиостанции лицензию на вещание на двух различных частотах.

Владельцы радиостанции имеют возможность транслировать свои радиопрограммы с использованием n радиовышек, расположенных в различных точках страны. Для осуществления трансляции на каждой радиовышке требуется установить специальный передатчик – трансмиттер. Каждый передатчик можно настроить на одну из двух частот, выделенных радиостанции. Кроме частоты вещания, передатчик характеризуется также своей мощностью. Чем мощнее передатчик, тем на большее расстояние он распространяет радиоволны. Для простоты, предположим, что передатчик мощности R распространяет радиоволны на расстояние, равное R километрам.

Все передатчики, установленные на вышках, должны, согласно инструкции министерства, иметь одну и ту же мощность. Чтобы программы радиостанции могли приниматься на как можно большей территории, мощность передатчиков должна быть как можно большей. С другой стороны, необходимо, чтобы прием передач был качественным на всей территории Флатландии. Прием передач считается качественным, если не существует такого участка ненулевой площади, на который радиоволны радиостанции «Радио Информатика» приходят на одной частоте одновременно с двух вышек.

Требуется написать программу, которая определяет, какую максимальную мощность можно было установить на всех передатчиках, позволяющую выбрать на каждом передатчике такую одну из двух частот передачи, чтобы прием был качественным на всей территории Флатландии.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – количество вышек (3 ≤ N ≤ 1200). Последующие N строк содержат по два целых числа — координаты вышек. Координаты заданы в километрах и не превышают 104 по модулю. Все точки, в которых расположены вышки, различны. Все числа в строках разделены пробелом.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выводится вещественное число — искомая мощность передатчиков. Во второй строке выводятся N чисел, где i-е число должно быть равно 1, если соответствующий передатчик должен вещать на первой частоте, и 2, если на второй. Ответ должен быть выведен с точностью, не меньшей 10–8.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 0 0 0 1 1 0 1 1 | 0.70710678118654752 1 2 2 1 |

# Варіант 67

Современный мир немыслим без Интернета и электронной почты. Для того, чтобы людям было проще ориентироваться в потоке поступающих писем, были созданы специальные программы - почтовые клиенты. Фирма TIRLABS занимается разработкой почтового клиента The Bar!.

Недавно программисты компании завершили разработку очередной, 366239-ой, версии этого почтового клиента. Менеджеры по продажам и рекламе уже готовы вовсю рекламировать и продавать эту новую программу. Однако, генеральный директор компании TIRLABS считает, что любая программа должна быть подвергнута всестороннему тестированию, прежде чем она будет продаваться. «Да и не работающую программу, скорее всего, никто не купит!» - сказал он.

Одним из видов тестирования сложных программ является так называемое стресс-тестирование. При нем программа тестируется в экстремальных условиях, часто даже в таких, на какие она не рассчитана. Для тестирования The Bar! ver. 366239 был выбран такой метод: программа запускается на n компьютерах, стоящих в одной комнате, после чего с компьютеров друг на друга посылается m писем. При этом никакие два события (отправление или прием письма) не происходят одновременно, а сеть настолько надежна, что все письма доходят до адресата. Адресат у каждого письма при этом только один.

Почтовый клиент The Bar! в процессе работы ведет протокол, в который заносятся идентификаторы отправленных и полученных писем в том порядке, в котором они были обработаны почтовым клиентом. При этом при оценке результатов тестирования в расчет принимаются только события, отраженные в этом протоколе.

Программа считается правильно работающей по результатам тестирования, если всем событиям, указанным в протоколах, можно сопоставить моменты времени таким образом, что никакие два события не происходят одновременно, и каждое из писем отправляется до того, как получается. При этом, разумеется, внутри каждого из протоколов порядок событий должен остаться прежним.

Даны протоколы работы почтового клиента на каждом из компьютеров. Напишите программу, проверяющую, можно ли по результатам этого тестирования признать программу правильно работающей.

Входные данные

Входной файл INPUT.TXT содержит несколько наборов входных данных. Первая строка содержит t - число наборов входных данных. Оставшиеся строки входного файла содержат эти наборы.

Описание каждого набора начинается со строки, содержащей два целых числа n (1 ≤ n ≤ 50000) и m (1 ≤ m ≤ 100000) - количество компьютеров и отправленных писем соответственно. Далее следуют n строк, i-ая из которых содержит протокол работы почтового клиента на i-ом компьютере. Протокол работы состоит из целого числа ki(0 ≤ ki ≤ 2m) и ki чисел ai,j , описывающих события. Если ai,j > 0, то j-ым по счету событием на i-ом компьютере была посылка письма с идентификатором ai,j , если же ai,j < 0, то j-ым по счету событием на i-ом компьютере было получение письма с идентификатором -ai,j. Нулю ai,j равно быть не может.

Идентификатор письма - это целое число от 1 до 106. Внутри одного набора входных данных все письма имеют различные идентификаторы. Гарантируется, что все письма, которые были отправлены, были кем-то приняты, то есть сумма всех ki в одном наборе входных данных равна 2m.

Сумма чисел n по всем наборам входных данных не превосходит 50000, сумма чисел m по всем наборам входных данных не превосходит 100000.

Выходные данные

В выходной файл OUTPUT.TXT для каждого набора входных данных выведите ровно одну строку. Эта строка должна содержать слово YES, если программу можно считать правильно работающей по результатам тестирования, и NO - в противном случае.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 2 2 2 2 1 -2 2 2 -1 2 2 2 -2 1 2 -1 2 | YES NO |
| 2 | 1 2 3 2 1 -1 4 239 -239 366 -366 | YES |

# Варіант 68

На поверхности Юпитера приземлился летающий робот, его задачей является исследование поверхности планеты. Поверхность Юпитера представляет собой прямоугольное поле из N×M клеток, каждая из которых представляет лаву или твердую поверхность различной высоты. Для снятия проб грунта роботу необходимо переместиться из клетки с координатами (1,1) в клетку с координатами (X,Y). Для передвижения робот может использовать два типа действий - это переезд и перелет.

Переезд - перемещение в одну из четырех соседних клеток, имеющих общую грань с заданной, при этом высота клеток не должна отличаться больше чем на единицу.

Перелет из одной клетки в другую позволяет преодолевать препятствия, но для него нужна энергия, которая расходуется из специальных аккумуляторов, количество которых на борту ограничено и равно K. Дальность перелета ограничена мощностью одного аккумулятора и составляет D единиц. Перелет возможен только по направлениям, параллельным границам поля. Во время перелета луноход не может менять направление, но при этом может менять высоту, облетая препятствия. На каждое перемещение на одну клетку вдоль выбранного направления, вверх или вниз на одну единицу по высоте, тратится ровно одна единица энергии. После перелета аккумулятор утилизируется и больше использоваться не может, оставшаяся в нем энергия пропадает.

Одним действием назовем один переезд или один перелет. Ваша задача для заданной поверхности найти наименьшее количество действий, необходимых для достижения заданной клетки.

Входные данные

Первая строка входного файла INPUT.TXT содержит размеры поля N, M (1 ≤ N,M ≤ 100), разделенные пробелом. Во второй строке идут координаты клетки, куда необходимо найти путь X,Y (1 ≤ X ≤ N, 1 ≤ Y ≤ M). На третьей строке через пробел указаны целые K и D (0 ≤ K ≤ 10, 0 ≤ D ≤ 100). Далее в M строках идут по N чисел через пробел – высотные отметки участка Юпитера, высота каждой клетки - целое число, лежащее в диапазоне от 0 до 10000 включительно. Высота 0 (ноль) обозначает лаву, на которой останавливаться нельзя, но можно пролететь над ней.

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число - минимальное количество действий, необходимых для достижения заданной клетки. Если добраться до заданной клетки нельзя, то необходимо вывести в строке слово IMPOSSIBLE.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 5 4 5 3 1 6 2 1 4 2 1 1 2 4 2 1 4 4 6 2 1 2 2 2 2 1 | 5 |
| 2 | 4 4 4 2 1 3 2 0 0 3 3 0 4 3 4 0 5 2 4 5 5 1 | 3 |
| 3 | 3 3 3 3 10 10 1 0 0 1 0 0 0 1 1 | IMPOSSIBLE |

# Варіант 69

В стране Умландии построили аттракцион "Лабиринт знаний". Лабиринт представляет собой N комнат, занумерованных от 1 до N, между некоторыми из которых есть двери. Когда человек проходит через дверь, показатель его знаний изменяется на определенную величину, фиксированную для данной двери. Вход в лабиринт находится в комнате 1, выход - в комнате N. Каждый ученик проходит лабиринт ровно один раз и попадает в ту или иную учебную группу в зависимости от количества набранных знаний (при входе в лабиринт этот показатель равен нулю). Ваша задача показать наилучший результат.

Входные данные

Первая строка входного файла INPUT.TXT содержит целые числа N (1 ≤ N ≤ 2000) - количество комнат и M (0 ≤ M ≤ 10000) - количество дверей. В каждой из следующих M строк содержится описание двери - номера комнат, из которой она ведет и в которую она ведет (через дверь можно ходить только в одном направлении), а также целое число, которое прибавляется к количеству знаний при прохождении через дверь (это число по модулю не превышает 10000). Двери могут вести из комнаты в нее саму, между двумя комнатами может быть более одной двери.

Выходные данные

В выходной файл OUTPUT.TXT выведите ":)" - если можно получить неограниченно большой запас знаний, ":(" - если лабиринт пройти нельзя, и максимальное количество набранных знаний в противном случае.

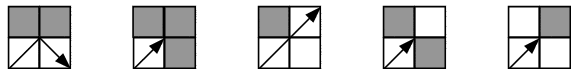
Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 2 2 1 2 5 1 2 -5 | 5 |

# Варіант 70

Темное царство представляет собой лабиринт N×M, некоторые клетки которого окружены зеркальными стенами, а остальные — пустые. Весь лабиринт также окружен зеркальной стеной. В одной из пустых клеток лабиринта поставили светофор, который испускает лучи в 4 направлениях: под 45 градусов относительно стен лабиринта. Требуется изобразить траекторию этих лучей.

Когда луч приходит в угол, через который проходят зеркальные стены, дальше он идет так, как показано на рисунках (серым цветом показаны клетки, которые окружены зеркальными стенами). Аналогичным образом луч ведет себя, когда приходит на границу лабиринта.



Входные данные

В первой строке входного файла INPUT.TXT записаны два натуральных числа N и M — число строк и столбцов в лабиринте (каждое из чисел не меньше 1 и не больше 100). В следующих N строках записано ровно по M символов в каждой — карта лабиринта. Символ \* (звездочка) обозначает клетку, окруженную зеркальными стенками, . (точка) — пустую клетку, символ X (заглавная английская буква X) — клетку, в которой расположен светофор (такая клетка ровно одна).

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк по M символов в каждой — изображение лабиринта с траекториями лучей. Здесь, как и раньше, \* (звездочка) должна обозначать клетки, окруженные зеркальными стенами, . (точка) — пустые клетки, через которые лучи света не проходят, / (слеш) — клетки, через которые луч света проходит из левого нижнего угла в правый верхний (или обратно — из правого верхнего в левый нижний), \ (обратный слеш) — клетки, через которые луч проходит из левого верхнего угла в правый нижний (или обратно), а символ X (заглавная английская буква X) — клетки, через которые лучи проходят по обеим диагоналям.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 5 6 ..\*... ...... .....\* \*X...\* ....\*. | ./\*./\ /..X./ \./.X\* \*X.//\* /\X/\*. |
| 2 | 3 3 ... .X. ... | \./ .X. /.\ |

# Варіант 71

В области L находится n городов. Некоторые пары городов соединены проселочной дорогой с двусторонним движением. Начавшись в каком-то городе, дорога не может закончиться в нем же. В этом году состояние дорог позволило отделению ГИБДД области L провести гонки под лозунгом «Скажем НЕТ нарушениям скоростного режима». Было решено, что круговая трасса должна состоять из четырех дорог, но не может проходить через один город два раза. Естественно, свернуть с одной дороги на другую можно только в городе. Организаторы уже должны приступить к составлению отчета, и для этого требуется посчитать количество различных трасс.

Входные данные

В первой строке входного файла INPUT.TXT записаны количество городов n (1 ≤ n ≤ 300) и количество дорог m. В каждой из следующих m строк содержится два различных числа — номера городов, соединенных соответствующей дорогой.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – количество круговых трасс из четырех дорог, которые могут составить организаторы.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 6 1 2 2 3 3 4 4 1 1 3 2 4 | 3 |

# Варіант 72

Всем известно, что в позапрошлом веке ковбои занимались перегоном скота. Перегон скота всегда считался опасным делом. Ковбой Джон, готовясь к очередному перегону, изучал план местности. Так как местность гористая, то добраться из одного поселения в другое можно только по дорогам, возможно через другие поселения. Главной опасностью на пути были бандиты, проживающие в разных населенных пунктах, и организующие группировки для нападения на ковбоев. Чтобы их разобщить, Джон разработал гениальный план полной изоляции поселений друг от друга.

Посоветовавшись с напарниками, Джон пришел к выводу, что временно дороги можно вывести из строя, устроив камнепад. Под покровом ночи можно выехать из одного населения в другое, остановиться где-то посреди дороги и устроить камнепад так, чтобы по этой дороге нельзя больше проехать никому. Камни падают позади повозки, поэтому обратной дороги нет. Но зато можно ехать в другой населенный пункт, и если из него существуют дороги, то и их можно вывести из строя.

Сам Джон этого сделать не может, только он знает тайные тропы и должен перегонять стадо. Поэтому он решил использовать наемников. Наемники есть в любом поселении и в любом количестве, однако, за каждого из них приходится платить не малую сумму, поэтому Джон хочет потратить как можно меньше денег. Помогите Джону определить минимальное число наемников, которые смогут привести в непригодность абсолютно все дороги и изолировать все поселения.

Входные данные

В первой строке входного файла INPUT.TXT находятся два целых неотрицательных числа N (0 < N < 1000) – количество поселений и M (0 ≤ M ≤ 100 000) – количество дорог, их соединяющих. Далее следует M строк, содержащих описание дорог. В i-й строке находятся два натуральных числа V и U (1 ≤ V, U ≤ N) – номера поселений, которые соединяет i-я дорога. Между двумя различными поселениями существует не более одной дороги, но может существовать несколько маршрутов. Нет дорог, которые образуют петлю, исходящую из поселения и ведущую в него же, не заходя в другие поселения. Не гарантируется, что существует маршрут между любой парой поселений. Маршрутом называется такая последовательность поселений s1-s2- … -sn, что любые два последовательных поселения si и si+1 соединены дорогой.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное количество наемников, необходимое для изоляции всех поселений.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 1 0 | 0 |
| 2 | 2 1 1 2 | 1 |
| 3 | 6 6 1 2 2 3 1 3 4 5 5 6 4 6 | 2 |

# Варіант 73

В скором времени на телеэкраны одной страны выйдет новое шоу «Двое в лабиринте». Его сюжет будет состоять в том, что два участника будут помещены в лабиринт. Их целью является поиск выхода из данного лабиринта. Первый, кто найдет выход, получит крупный денежный приз.

Однако, прежде чем шоу выйдет на экраны, лабиринты должны быть сертифицированы Государственным Бюро по Сертификации Лабиринтов. В своей работе бюро использует специальные машины, называемые обходчиками лабиринтов.

Поскольку в силу специфики работы этих машин для каждого лабиринта приходится строить нового обходчика, Вам поручено провести компьютерное моделирование обхода лабиринта обходчиком.

Лабиринт состоит из n комнат, соединенных m коридорами. На концах коридора имеются две двери, одна из которых открывается только из коридора, в вторая только из комнаты, из которой коридор выходит, таким образом, движение по коридору разрешено только в одну сторону. Кроме этого, каждый из коридоров покрашен в один из k цветов (это сделано для того, чтобы немного облегчить участникам нахождение выхода из лабиринта). Цвет коридора указан на соответствующей ему двери в комнате, из которой он выходит. При этом из комнаты могут выходить несколько коридоров одного цвета.

Обходчик лабиринтов работает по программе, которая состоит из L инструкций. Каждая инструкция – это номер цвета (число от 1 до k). Обход лабиринта начинается в комнате номер s и совершается следующим образом: обходчик поочередно считывает инструкции и на каждом шаге выбирает один из коридоров, покрашенных в цвет, указанный в этой инструкции. Если такого коридора не находится, то обходчик «зависает».

Так как на каждом шаге у обходчика может быть не один вариант выбора коридора, то комната, в которой он окажется после выполнения программы может определяться неоднозначно.

Ваша задача состоит в том, чтобы по описанию лабиринта и программе для обходчика определить, в каких комнатах обходчик может оказаться после выполнения соответствующей программы.

Входные данные

Первая строка входного файла INPUT.TXT содержит три целых числа: n, m, k (1 ≤ n, k ≤ 1000, 0 ≤ m ≤ 10000). Далее идут m строк, описывающих коридоры. Описание каждого коридора состоит из трех целых чисел: u, v, c (1 ≤ u, v ≤ n, 1 ≤ c ≤ k). Их значения таковы: u - номер комнаты, из которой выходит коридор, v - номер комнаты, в которую ведет коридор, c цвет этого коридора. Коридор может вести из комнаты в саму себя, между двумя комнатами может существовать несколько коридоров (более того, несколько коридоров одного цвета).

(m+2)-ая строка входного файла содержит целое число L (1 ≤ L ≤ 1000). (m+3)-ая строка содержит L целых чисел от 1 до k – программы для обходчика лабиринта.

Последняя строка входного файла содержит целое число s (1 ≤ s ≤ n).

Выходные данные

В выходной файл OUTPUT.TXT следует вывести слово Hangs в случае, если обходчик «зависает» независимо от того, какие коридоры он выбирает при существовании нескольких коридоров одного цвета.

Иначе, выведите на первой строке выходного файла слово OK, во второй – количество r комнат, в которых обходчик может оказаться после выполнения программы. В третьей строке выходного файла в этом случае выведите номера этих комнат в возрастающем порядке.

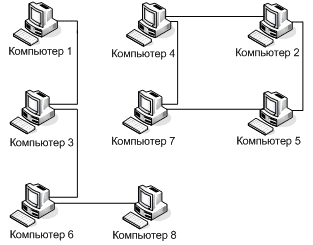
Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 6 2 1 2 1 1 2 2 1 3 1 1 3 2 3 4 2 3 3 2 2 1 2 1 | OK 2 3 4 |
| 2 | 4 6 2 1 2 1 1 2 2 1 3 1 1 3 2 3 4 1 3 3 1 2 1 2 1 | Hangs |

# Варіант 74

В одной из самых крупных IT-компаний уже много лет действует локальная сеть, в которую входят N компьютеров, последовательно пронумерованных целыми числами от 1 до N. Для соединения компьютеров используется провод специального типа. С помощью одного провода можно напрямую соединить ровно два компьютера, причем соединение позволяет передавать данные в двух направлениях. Каждый компьютер имеет два порта, поэтому он может быть соединен напрямую не более чем с двумя другими компьютерами. Сеть организована таким образом, что если два компьютера не соединены напрямую проводом, то передача данных может осуществляться через промежуточные компьютеры. Путем передачи данных между компьютерами с номерами V1 и VK называется такая последовательность компьютеров V1, V2, … , VK-1, VK, что для любого 2 ≤ i ≤ K существует прямое соединение между компьютерами Vi-1 и Vi. Длиной пути передачи данных считается количество используемых промежуточных компьютеров. Минимальным путем передачи данных между компьютерами Vi и Vj является такой путь передачи данных, длина которого минимальна из возможных.

Для приведенного ниже примера длина минимального пути передачи данных между компьютерами 1 и 8 равна 2, а между компьютерами 2 и 7 равна 1, между компьютерами 2 и 4 равна 0.



Если не существует ни одного пути передачи данных между какими-либо двумя компьютерами, то в данном случае длина минимального пути передачи данных между этими компьютерами считается равной нулю. Для приведенного выше примера между компьютерами 1 и 4 не существует пути передачи данных, поэтому длина минимального пути передачи данных равна 0.

Для выяснения эффективности локальной сети необходимо определить коэффициент связности, значение которого равно сумме длин минимальных путей передачи данных между всеми парами компьютеров. Пары, отличающиеся порядком следования элементов считаются различными, то есть пара (1, 2) и (2, 1) считаются различными.

Вашей задачей является определить значение коэффициента связности для заданной локальной сети.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа N и M – число компьютеров и число прямых соединений соответственно (2 ≤ N ≤ 105, 1 ≤ M ≤ 106). Числа разделены одиночным пробелом.

Каждая из следующих M строк содержит описание одного прямого соединения. Каждое прямое соединение описывается при помощи двух целых чисел Xi и Yi (1 ≤ Xi, Yi ≤ N; Xi ≠ Yi), разделенных пробелом – номера соответствующих компьютеров, соединенных напрямую. Известно, что Xi ≠ Xj или Yi ≠ Yj, если i ≠ j. Строки в файле нумеруются последовательно, начиная с единицы.

Выходные данные

Единственная строка выходного файла OUTPUT.TXT должна содержать одно целое число – коэффициент связности локальной сети.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 2 1 1 2 | 0 |
| 2 | 4 4 1 2 2 3 4 3 1 4 | 4 |
| 3 | 8 7 1 3 3 6 4 2 7 4 2 5 5 7 6 8 | 12 |

# Варіант 75

ан ориентированный взвешенный граф. По его матрице смежности нужно для каждой пары вершин определить: существует кратчайший путь между ними или нет.

Кратчайший путь может не существовать по двум причинам: либо нет ни одного пути, либо есть путь сколь угодно маленького веса.

Входные данные

В первой строке входного файла INPUT.TXT записано единственное число N (1 ≤ N ≤ 100) - количество вершин графа. В следующих N строках по N целых чисел - матрица смежности графа (j-ое число в i-ой строке соответствует весу ребра из вершины i в вершину j), в которой число 0 обозначает отсутствие ребра, а любое другое число - наличие ребра соответствующего веса. Все числа по модулю не превышают 100.

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк по N чисел: j-ое число в i-ой строке должно быть равно 0, если путь из i в j не существует, 1 - если существует кратчайший путь, и 2 - если существует путь сколь угодно маленького веса.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 5 0 1 2 0 0 1 0 3 0 0 2 3 0 0 0 0 0 0 0 -1 0 0 0 -1 0 | 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 2 2 0 0 0 2 2 |

# Варіант 76

Дано N прямоугольников со сторонами, параллельными осям координат. Требуется определить, на сколько частей эти прямоугольники разбивают плоскость (внутри частей не должно быть границ прямоугольников).

Входные данные

В первой строке входного файла INPUT.TXT содержится число прямоугольников N (1 ≤ N ≤ 100). Далее идут N строк, содержащих по 4 числа x1, y1, x2, y2 - координаты двух противоположных углов прямоугольника. Координаты представляют собой целые числа и по абсолютной величине не превосходят 10 000.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - количество частей, на которые разбивается плоскость.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 10 20 50 30 40 10 50 25 40 25 80 30 | 6 |

# Варіант 77

Женя недавно купил себе новую соковыжималку. Теперь по утрам он и его братья и сестры пьют свежевыжатый фруктовый сок. А это, между прочим, очень полезно! Недавно они поняли, что можно пить сок, выжатый не только из одного вида фруктов, как, например, апельсиновый, но и различные смеси, например, виноградно-яблочный.

В Жениной семье все очень любят сок, поэтому могут утром выпить не один стакан, причем разных видов сока. Например, его сестра Катя очень любит грейпфрутовый и апельсиновый соки. Женя, как наиболее технически грамотный человек, каждое утро занимается приготовлением соков.

Опишем подробнее, как работает соковыжималка. В нее загружаются фрукты, они проходят отжим в центрифуге, обезвоженная мякоть сбрасывается в отдельный резервуар, а сок попадает в специальную емкость.

Основная проблема состоит в том, что эту емкость иногда приходится мыть. Например, если после приготовления апельсинового сока, необходимо приготовить яблочный, то емкость надо мыть, иначе получится апельсиново-яблочный сок. Более формально, пусть сок A состоит из компонентов a1, ... , an, а сок B - из компонентов b1, ... , bm. Сок B можно готовить после сока A, если любой из компонентов сока A является компонентом сока B. В противном случае емкость для сока надо помыть.

Женя не очень любит мыть посуду, поэтому хочет мыть емкость как можно меньшее число раз. Помогите ему.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – количество различных соков, которые требуется приготовить (N ≤ 300). Каждая из последующих N строк описывает один из соков. Описание сока состоит из числа k его компонентов (1 ≤ k ≤ 300) и списка этих компонентов. Каждый из компонентов сока описывается словом длиной до 30 символов из строчных и прописных букв английского алфавита. Прописные и строчные буквы различаются. Различные компоненты имеют различные названия.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное количество раз, которое Жене придется помыть емкость для сока. Учитывайте при этом, что емкость для сока надо помыть и после приготовления последней порции сока.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 1 Apple 2 Apple Orange 1 Orange 2 Orange Pineapple | 2 |
| 2 | 3 1 Apple 1 Orange 1 Mango | 3 |

# Варіант 78

Начинается зима, но в городе К на дорогах все еще продолжается ремонт. После того, как очередная дорога была отремонтирована, департамент транспорта пришел к выводу, что дешевле обслуживать дороги с односторонним движением.

Сейчас все дороги в городе К двусторонние. Дорога состоит из двух полос – встречного и попутного направления. Требуется превратить максимальное количество дорог в односторонние, оставив одну из двух полос так, чтобы сообщение между точками города не нарушилось. Это означает, что если из точки i можно проехать в точку j (прямо или через промежуточные точки), то после введения одностороннего движения эта возможность должна остаться.

Карта города задаётся матрицей смежности M, заполненной нулями и единицами. Размерность матрицы N (1 ≤ N ≤ 100) – число точек города. Если M[i, j] = 1, то существует полоса для перемещения из точки i в точку j, не проходящая через другие точки, иначе M[i, j] = 0 (M[i, i] = 0 для любого i). Матрица M симметрична относительно главной диагонали, т.е. M[i, j] = M[j, i], что означает двустороннюю дорогу (полоса встречного и попутного направления).

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число N – размерность матрицы M, следующие N строк, каждая по N символов (нули или единицы), разделенные пробелами – элементы матрицы M.

Выходные данные

В выходной файл OUTPUT.TXT выведите максимально возможное число двусторонних дорог, которые можно превратить в односторонние.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 4 0 0 1 1 0 0 0 0 1 0 0 1 1 0 1 0 | 3 |
| 2 | 7 0 1 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 | 6 |

# Варіант 79

Кубик Рубика – популярная головоломка в форме куба, состоящая из множества мелких кубиков. Каждая видимая сторона такого кубика окрашена в определенный цвет. Повороты сторон кубика позволяют переупорядочить цветные квадраты множеством различных способов. Целью игры служит поиск последовательности поворотов сторон куба таким образом, чтобы он вернулся в первоначальное состояние, где каждая из граней состоит из квадратов одного цвета.

Доказано, что число всех достижимых различных состояний традиционного 6-цветного кубика Рубика 3×3×3 равно 43 252 003 274 489 856 000, а оптимальная последовательность ходов, необходимых для сборки кубика Рубика из любого состояния не превышает 20. Алгоритм, собирающий кубик Рубика за минимальное число ходов, традиционно называется «алгоритмом Бога», а 20 – числом Бога.

Рассмотрим более простую модель кубика Рубика 2×2×2, в которой используется всего 3 цвета: красный (R), синий (B) и зеленый (G). При этом противоположные грани окрашены в одинаковый цвет. Под ходом будем понимать вращение одной из граней на угол 90º. Поскольку всего 6 граней (передняя (F), левая (L), задняя (B), правая (R), верхняя (U) и нижняя (D)), то всего может быть 12 различных ходов (по часовой и против часовой стрелки для каждой грани). Ходы обозначаются буквами соответствующих граней, если ход происходит против часовой стрелки, то после буквы ставится апостроф (ASCII 39).

По заданной конфигурации трехцветного кубика Рубика 2×2×2 требуется найти оптимальный алгоритм (алгоритм Бога), решающий головоломку.

Входные данные

Входной файл INPUT.TXT содержит две строки, определяющие конфигурацию кубика Рубика. Первая строка содержит информацию о цветах верхнего слоя для каждой грани – пары символов, разделенные пробелом. Вторая строка аналогичным образом описывает нижние слои. Порядок граней (слева направо): передняя, левая, задняя, правая, верхняя, нижняя.

Выходные данные

В выходной файл OUTPUT.TXT выведите оптимальное решение головоломки для заданной конфигурации кубика Рубика. Если решений несколько, выведите любое. Если задана начальная конфигурация (кубик Рубика собран), то следует вывести «Solved».

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | RR GG RR GG BB BB RR GG RR GG BB BB | Solved |
| 2 | RR BG GR GB GR BB GG BR GR RB BB GR | F'R |
| 3 | BR GR GR GR BB RR BG RG RG BB GB BG | LULU'LB'DBR |

# Варіант 80

A picture containing application

Description automatically generated

Костя нашел головоломку, в которую его дедушка играл в детстве - доску n\_m клеток, на некоторых из которых прикреплены вращающиеся стрелки (Костя предполагает, что раньше стрелки были на всех клетках). К сожалению, дедушка уже не помнит, что нужно было сделать в этой головоломке, поэтому Костя придумывает правила сам.

Он заметил следующую особенность. Направим каждую стрелку на одну из соседних клеток, на которой тоже есть стрелка (оказалось, что одиноко стоящих стрелок на доске нет). Теперь на доске появились циклы, идя по которым, мы всегда будем идти по направлению стрелок. Например, при расстановке, которая отражена на рисунке, есть четыре цикла.

Костя придумал сразу два задания: «поставить стрелки так, чтобы число циклов было минимально», и «поставить стрелки так, чтобы число циклов было максимально». Пока Костя ищет решение вручную, напишите программу, находящую требуемые расстановки.

Входные данные

Входной файл INPUT.TXT содержит Первая строка входного файла содержит числа n и m (1 ≤ n, m ≤ 100). Следующие n строк по m символов содержат описание доски: символ «?» соответствует клетке со стрелкой, а символ «.» - пустой клетке.

Выходные данные

В выходной файл OUTPUT.TXT выведите сначала расстановку, в которой число циклов минимально, а затем расстановку, в которой число циклов максимально. Разделите расстановки одной пустой строкой.

Расстановку стрелок выводите, заменив в описании доски символы «?» на символы, соответствующие направлениям стрелок. Используйте символы «R», «L», «U» и «D» для обозначения направлений вправо, влево, вверх и вниз соответственно.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 3 4 ???? ???? .??. | RRRD UUDL .UL.  RLDD RLUU .RL. |

# Варіант 81

Однажды в древнее государство Оссия приехал ученый японец Хисикоши. Он обнаружил, что в этом государстве есть одна большая проблема – дороги, а точнее их отсутствие. Еще он обнаружил другую проблему – графов (а было их N), которые, мягко говоря, не отличались особенным уровнем интеллекта. Хисикоши решил помочь построить дороги. С рабочей силой проблем не было, поэтому он решил подойти к этому мероприятию с выдумкой. Он узнал имена всех графов и записал их по-японски (а в Японии, как известно, пользуются иероглифами). Он решил, что надо провести дорогу между замками двух графов (и ввести на ней одностороннее движение от первого ко второму – чтобы интереснее было), если последняя буква имени первого графа совпадает с первой буквой имени второго. Однако, оказалось, что не от каждого графа можно доехать до всех других. Тогда он решил поселить еще несколько графов и дать им имена так, чтобы теперь можно было бы доехать от любого графа до любого другого. Какое наименьшее количество графов ему надо добавить?

Будем считать, что японский иероглиф однозначно кодируется тремя английскими буквами. И каждой трехбуквенной комбинации соответствует свой иероглиф.

Входные данные

В первой строке входного файла INPUT.TXT находится целое число N – количество графов (1 ≤ N ≤ 100 000). В следующих N строках написаны их имена. При этом гарантируется, что длина каждого имени (в английских символах) делится на 3. Имена написаны большими английскими буквами. Каждое имя содержит от одного до десяти иероглифов.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное целое число – минимальное количество графов, которые должен поселить Хисикоши.

Примеры

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 2 AAABBB BBBCCC | 1 |
| 2 | 2 AAABBB CCCDDD | 2 |
| 3 | 3 AAABAA ABAAAB AABBAA | 2 |

# Варіант 82

В королевстве Далеком n городов, соединенных m двусторонними дорогами. Некоторые дороги вымощены камнем, а другие представляют собой лишь обычные проселочные дороги. Столица королевства расположена в городе номер 1. Дороги устроены таким образом, что можно из любого города добраться до любого другого, перемещаясь исключительно по вымощенным камнем дорогам, причем количество каменных дорог минимальное возможное. Проселочные же дороги были построены таким образом, что если каменная дорога заблокирована или уничтожена, то все равно можно добраться из любого города до любого другого по дорогам.

Обозначим количество каменных дорог, по которым требуется проехать, чтобы попасть из города u в город v как s(u, v). Дороги подчиняются следующему правилу: если два города u и v соединены дорогой (неважно, каменной или проселочной), то либо s(1, u) + s(u, v) = s(1, v), либо s(1, v) + s(v, u) = s(1, u).

Король соседнего королевства планирует напасть на Далекое. В качестве начала операции предполагается уничтожить некоторые дороги. Расчеты показали, что финансирования, выделенного министерству атаки, достаточно, чтобы уничтожить ровно одну каменную и одну проселочную дорогу. Король хотел бы уничтожить такую пару дорог, чтобы после этого хотя бы для каких-нибудь двух городов стало невозможно добраться из одного города в другой.

Теперь он просит министра атаки посчитать количество возможных диверсионных планов. Однако министр атаки обучен только атаковать, считать для него слишком сложно. Помогите ему!

Входные данные

Первая строка входного файла INPUT.TXT содержит n и m – количество городов и дорог, соответственно (3 ≤ n ≤ 20 000, m ≤ 100 000). Следующие m строк описывают дороги, каждая строка содержит три целых числа – номера городов, соединенных соответствующей дорогой, и 1, если соответствующая дорога вымощена камнем, или 0, если соответствующая дорога – проселочная. Никакие два города не соединены более чем одной дорогой, никакая дорога не соединяет город сам с собой.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – количество способов организовать диверсию.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 6 7 1 2 1 2 3 1 1 4 0 3 4 1 4 5 1 3 6 0 5 6 1 | 4 |

# Варіант 83

Леша работает в службе доставки сушеных грибов. Некоторые клиенты жалуются, что грибы доставляются слишком долго. Леша хочет проверить, не является ли это следствием неэффективной системы доставки.

Система доставки устроена следующим образом. Существуют n распределительных пунктов, между некоторыми из которых есть односторонние каналы доставки. Грибы могут быть перевезены из пункта ai в пункт bi за время ti. С основной базы грибы поступают на пункт номер 1 и далее доставляются до нужного пункта по каналам доставки. Система каналов довольна сложна, так что есть несколько путей, по которым можно доставить грибы до некоторых пунктов. При этом никто не следит за тем, чтобы выбранный путь был самым коротким.

Леша хочет узнать для каждого пункта, какое максимальное время до него могут идти грибы от первого пункта при условии, что на промежуточных пунктах грибы не задерживаются.

Входные данные

В первой строке входного файла INPUT.TXT содержатся числа n и m - количество распределительных пунктов и количество каналов доставки (2 ≤ n ≤ 100, 1 ≤ m ≤ 10000). Далее следуют m троек ai, bi, ti (ai ≠ bi, 1 ≤ ti ≤ 100).

Выходные данные

В выходной файл OUTPUT.TXT выведите n-1 число – максимальное время доставки грибов до второго, третьего, …, n-го пункта. Если грибы до пункта могут идти сколь угодно долго, выведите вместо соответствующего числа -1.

Пример

|  |  |  |
| --- | --- | --- |
| № | INPUT.TXT | OUTPUT.TXT |
| 1 | 6 7 1 2 2 2 3 3 1 3 1 3 4 1 4 5 2 5 3 1 3 6 1 | 2 5 6 8 -1 |