Лабораторна работа №5

# Тема: Алгоритм Крускала-Прима. Алгоритм Дейкстри.

**Мета:** опрацювати алгоритми знаходження кістякового дерева найбільшої на найменшої ваги; знаходження шляху найменьшої ваги між вершинами, відновлення маршруту за індексами з алгоритму Дейкстри.

## ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ 6.

Завдання 1. Побудувати мінімальне остовне дерево використовуючи алгоритми Крускала-Прима.

Завдання 2. Побудувати максимальне остовне дерево використовуючи алгоритми Крускала-Прима.

Завдання 3. Розставити індекси в вершинах та знайти найкоротший шлях з вершини Vz до вершини Vo використовуючи алгоритм Дейкстри.

Завдання 4. Знайти найкоротший шлях з вершини Vz до вершини Vo

## АЛГОРИТМ КРУСКАЛА

Алгоритм Крускала — алгоритм побудови мінімального кістякового дерева зваженого неорієнтовного графа. Алгоритм було вперше описано Джозефом Крускалом 1956 року.

Візьмемо зважений зв'язний граф *G=(V,E),* де *V* — множина вершин, *E* — множина ребер, для кожного з яких задано вагу (довжину). Тоді ациклічна множина ребер, що поєднують усі вершини графа і чия загальна вага мінімальна, називається мінімальним каркасним (або кістяковим) деревом.

Алгоритм Крускала починається з побудови виродженого лісу, що містить *V* дерев, кожне з яких складається з однієї вершини. Далі виконуються операції об'єднання двох дерев, для чого використовуються найкоротші можливі ребра, поки не утвориться єдине дерево. Це дерево і буде мінімальним кістяковим деревом.

**Оцінка складності.** Алгоритм Крускала (як і алгоритм Прима) є класичним алгоритмом розв'язання задачі пошуку мінімального кістякового дерева. У разі використання найшвидших реалізацій час його роботи становить *O (E\*logE)*. Основна частина часу витрачається на сортування ребер за вагою. Так само, як і в простій версії алгоритму Крускала, відсортуємо всі ребра за вагою у неспадному порядку. Потім помістимо кожну вершину в своє дерево (тобто свою множину) за допомогою виклику функції перебираємо всі ребра (у порядку сортування) і для кожного ребра за *O(1)* визначаємо, чи належать його кінці різних деревам (за допомогою двох викликів за *O(1)).* Нарешті, об'єднання двох дерев буде здійснюватися викликом функції. Таким чином ми отримуємо складність алгоритму оцінену *O(MlogN+N+M)=O(MlogN).*

## **АЛГОРИТМ** **ДЕЙКСТРИ**

Алгоритм Дейкстри вирішує завдання про найкоротших шляхах з однієї вершини для зваженого орієнтованого графа *G = (V, E)* з вихідною вершиною с, в якому ваги всіх ребер невід'ємні (*(U, V) ≥ 0* для всіх *(і, v) http://comp-science.narod.ru/KPG/Path.files/e.jpg E*).

**Формальне пояснення.** У процесі роботи алгоритму Дейкстри підтримується безліч *S http://comp-science.narod.ru/KPG/Path.files/3.jpg V*, що складається з вершин v, для яких *δ(S, V)* вже знайдено. Алгоритм вибирає вершину *u http://comp-science.narod.ru/KPG/Path.files/e.jpg V\S*  з найменшим *d[u],* додає *u* до множини *S* і виробляє релаксацію всіх ребер, що виходять з *U*, після чого цикл повторюється. Вершини, що не лежать в *S*, зберігаються в черзі *Q* з пріоритетами, визначеними значеннями функції *d*. Передбачається, що граф заданий за допомогою списків суміжних вершин.

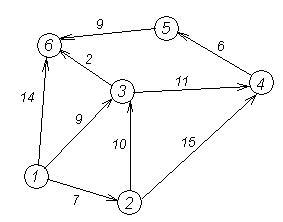
**Не формальне пояснення.** Кожній вершині з *v* зіставимо мітку - мінімальне відоме відстань від цієї вершини до. Алгоритм працює покроково - на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані.

**Ініціалізація.** Мітка самої вершини покладається рівною *0*, мітки інших вершин - нескінченності. Це відображає те, що відстані від до інших вершин поки невідомі. Всі вершини графа позначаються як невідвідані.

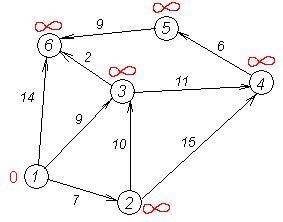
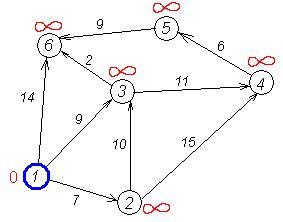
**Крок алгоритму**. Якщо всі вершини відвідані, алгоритм завершується. В іншому випадку з ще не відвіданих вершин вибирається вершина *v*, що має мінімальну позначку. Ми розглядаємо всілякі маршрути, в яких *v* є передостаннім пунктом. Вершини, з'єднані з вершиною *v* ребрами, назвемо сусідами цієї вершини. Для кожного сусіда розглянемо нову довжину шляху, що дорівнює сумі поточної мітки *u* і довжини ребра, що з'єднує у з цим сусідом. Якщо отримана довжина менше мітки сусіда, замінимо мітку цієї завдовжки. Розглянувши всіх сусідів, відмітимо вершину *u* як відвідану і повторимо крок.

## ПРИКЛАД

Розглянемо роботу алгоритму на прикладі графа, показаного на малюнку. Нехай потрібно знайти відстані від 1-й вершини до всіх інших.

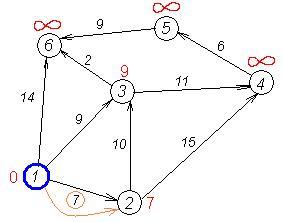


Кружками позначені вершини, лініями - шляху між ними (ребра графа). У кружках позначені номери вершин, над ребрами позначена їх «ціна» - довжина шляху. Поряд з кожною вершиною червоним позначена мітка - довжина найкоротшого шляху в цю вершину з вершини 1.

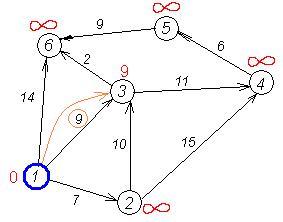
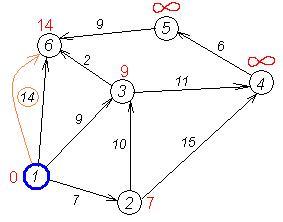
 

**Перший крок.** Розглянемо крок алгоритму Дейкстри для нашого прикладу. Мінімальну мітку має вершина 1. Її сусідами є вершини 2, 3 і 6.

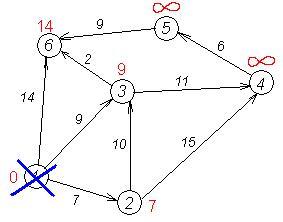
Перший по черзі сусід вершини 1 - вершина 2, тому що довжина шляху до неї мінімальна. Довжина шляху в неї через вершину 1 дорівнює найкоротшій відстані до вершини 1 + довжина ребра, що йде з 1 в 2, тобто 0 + 7 = 7. Це менше поточної мітки вершини 2, тому нова мітка 2-й вершини дорівнює 7.



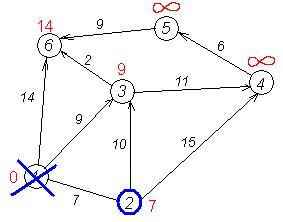
Аналогічну операцію виконуємо з двома іншими сусідами 1-й вершини - 3-й і 6-й.

Всі сусіди вершини 1 перевірені. Поточне мінімальна відстань до вершини 1 вважається остаточним і перегляду не підлягає (те, що це дійсно так, уперше довів Дейкстра). Викреслимо її з графа, щоб відзначити, що ця вершина переглянуло.



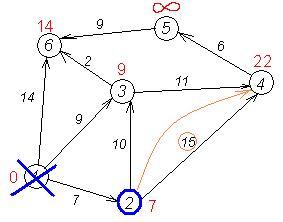
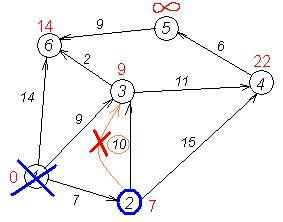
**Другий крок.** Крок алгоритму повторюється. Знову знаходимо «найближчу» з невідвіданих вершин. Це вершина 2 з міткою 7.



Знову намагаємося зменшити мітки сусідів обраної вершини, намагаючись пройти в них через 2-ю. Сусідами вершини 2 є 1, 3, 4.

Перший (по порядку) сусід вершини 2 - вершина 1. Але вона вже переглянуло, тому з 1-й вершиною нічого не робимо.

Наступний сусід вершини 2 - вершини 4 і 3. Якщо йти в неї через 2-ю, то довжина такого шляху буде = найкоротша відстань до 2 + відстань між вершинами 2 і 4 = 7 + 15 = 22. Оскільки 22 <∞, встановлюємо мітку вершини 4 рівною 22.

Ще один сусід вершини 2 - вершина 3. Якщо йти в неї через 2, то довжина такого шляху буде = 7 + 10 = 17. Але поточна мітка третьої вершини дорівнює 9 <17, тому мітка не змінюється.

Всі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і помічаємо її як відвідану.

**Третій крок.** Повторюємо крок алгоритму, вибравши вершину 3.

Подальші кроки. Повторюємо крок алгоритму для решти вершин (по порядку 6, 4 і 5).

Завершення виконання алгоритму. Алгоритм закінчує роботу, коли викреслені всі вершини. Результат його роботи: найкоротший шлях від вершини 1 до 2-й становить 7, до 3-й - 9, до 4-й - 20, до 5-й - 20, до 6-ї - 11.

Оскільки алгоритм Дейкстри всякий раз вибирає для обробки вершини з найменшою оцінкою найкоротшого шляху, можна сказати, що він відноситься до жадібним алгоритмам.

**Час роботи алгоритму Дейкстри**

Складність алгоритму Дейкстри залежить від способу знаходження вершини v, а також способу зберігання безлічі невідвіданих вершин і способи оновлення міток. Позначимо через n кількість вершин, а через т - кількість ребер в графі G.

• у найпростішому випадку, коли для пошуку вершини з мінімальним *d[v]* проглядається все безліч вершин, а для зберігання величин d - масив, час роботи алгоритму є *О (n2 + м).* основний цикл виконується порядку *п* раз, в кожному з них на знаходження мінімуму витрачається близько п операцій, плюс кількість релаксацій (змін міток), яка не перевершує кількості ребер у вихідному графі.

• для розріджених графів (тобто таких, для яких *т* багато менше *n2*) невідвідані вершини можна зберігати в двійковій купі, а в якості ключа використовувати значення *d [і],* тоді час вилучення вершини з u стане увійти *logn*, при тому, що час модифікації *d[і]* зросте до *logn.* так як цикл виконується порядку *n* разів, а кількість релаксації не більше *m* , швидкість роботи такої реалізації ***О(n\*log n + m\*log n).***

• якщо для зберігання невідвіданих вершин використовувати фібоначчієву купу, для якої видалення відбувається в середньому за *O(log n)*, а зменшення значення в середньому за *O(1)*, то час роботи алгоритму складе ***O(n\*log n + m)***.

## ПРИКЛАД ВИКОНАННЯ ВАРІАНТУ 0.

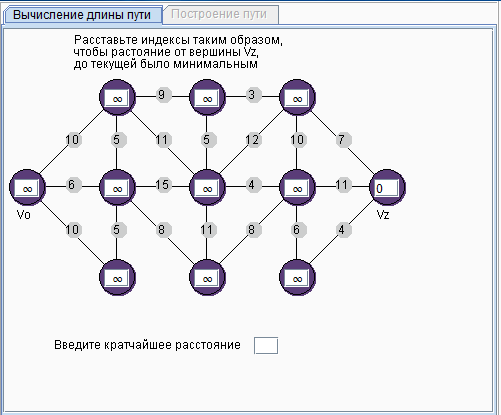
Завдання 1. Побудувати мінімальне остовне дерево використовуючи алгоритми Крускала-Прима.

Завдання 2. Побудувати максимальне остовне дерево використовуючи алгоритми Крускала-Прима.

Завдання 3. Розставити індекси в вершинах та знайти найкоротший шлях з вершини Vz до вершини Vo використовуючи алгоритм Дейкстри.

Завдання 4. Знайти найкоротший шлях з вершини Vz до вершини Vo

### Завдання 1. Побудувати мінімальне остовне дерево використовуючи алгоритми Крускала-Прима.



1

2

3

4

5

6

7

8

9

10

11

Складемо список ребер з їх довжинами

степені вершин: 3+4+5+2+3+6+3+4+5+2+3=40; 40/2=20 ребер.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (1,2) | (1,3) | (1,4) | (2,5) | (2,6) | (2,3) | (3,4) | (3,7) | (3,6) | (5,8) |
| 10 | 6 | 10 | 9 | 11 | 5 | 5 | 8 | 15 | 3 |

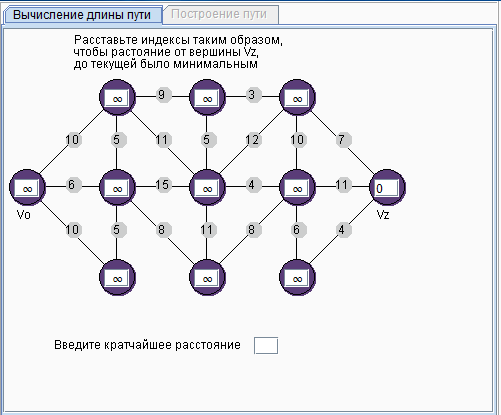
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (5,6 ) | (6,8) | (6,9) | (6,7) | (7,9) | (8,11) | (8,9) | (9,10) | (9,11) | (10,11) |
| 5 | 12 | 4 | 11 | 8 | 7 | 10 | 6 | 11 | 4 |

Впорядкуємо ребра по зростанню їх ваги (довжин).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 | 8 |
| (5,8) | (6,9) | (10,11) | (2,3) | (3,4) | (5,6 ) | (1,3) | (9,10) | (8,11) | (3,7) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 9 | 10 | 10 | 10 | 11 | 11 | 11 | 12 | 15 |
| (7,9) | (2,5) | (1,2) | (1,4) | (8,9) | (2,6) | (6,7) | (9,11) | (6,8) | (3,6) |

Вибераємо ребра починаючи з найменшого. Враховуючи не появу циклу!!!



1

2

3

4

5

6

7

8

9

10

11

Ребро (8,11) утворює цикл тому пропускається. Всі вершини зєднанні.

Отримане кістякове дерево найменшої ваги.

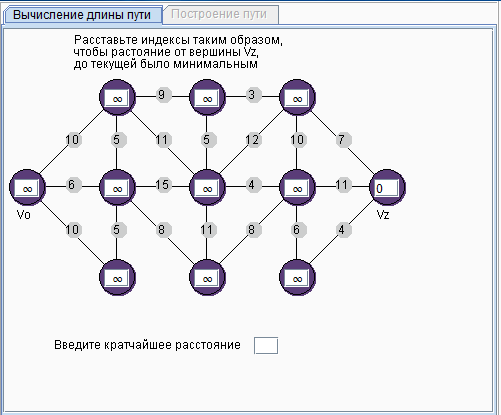
*W(G)=6+5+5+9+6+4+8+6+3+4=54.*

### Завдання 2. Побудувати максимальне остовне дерево використовуючи алгоритми Крускала-Прима.

Аналогічно побудувати кістякове дерево найбільшої ваги. Впорядкуємо ребра по спаданню їх ваги (довжин).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 12 | 11 | 11 | 11 | 10 | 10 | 10 | 9 | 8 |
| (3,6) | (6,8) | (9,11) | (6,7) | (2,6) | (8,9) | (1,4) | (1,2) | (2,5) | (7,9) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 6 | 5 | 5 | 5 | 4 | 4 | 3 |
| (3,7) | (8,11) | (9,10) | (1,3) | (5,6 ) | (3,4) | (2,3) | (10,11) | (6,9) | (5,8) |



1

2

3

4

5

6

7

8

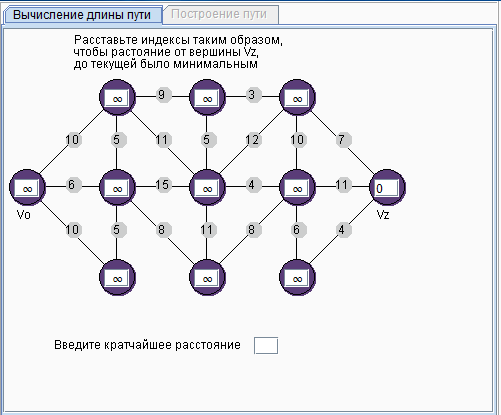
9

10

11

*W(G)=10+10+9+11+15+11+12+10+6+11=104,*

### *Завдання 3.* Розставити індекси в вершинах та знайти найкоротший шлях з вершини *Vz* до вершини *Vo* використовуючи алгоритм Дейкстри.



17

11

10

7

20

15

19

14

29

10

19

18

25

19

25

29

24

29

26

29

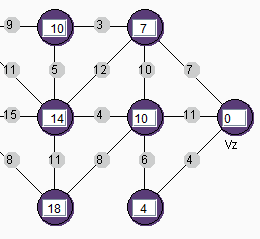
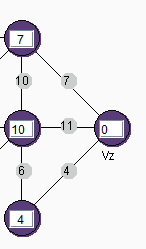
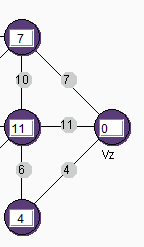
4

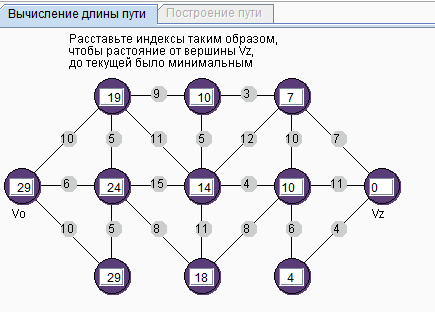
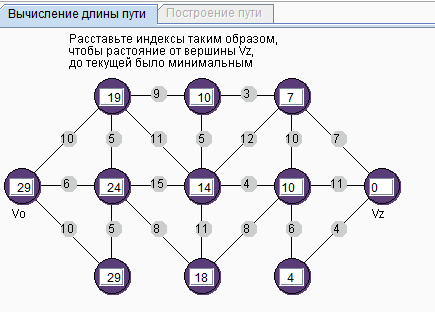
16

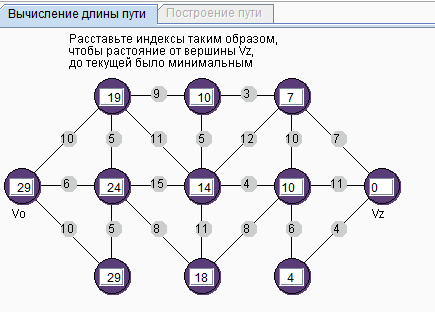
29

30

39



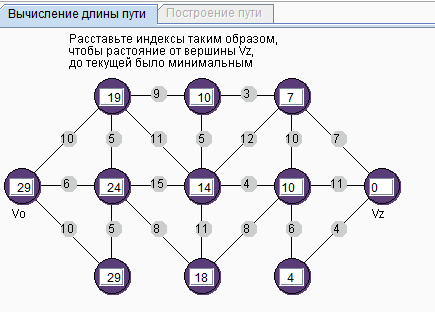
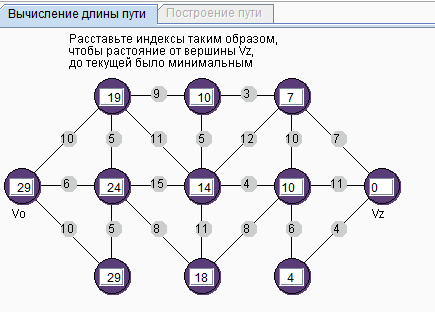




Найкоротша відстань з вершини *Vz* до вершини *Vo* співпадає з індексом *Vo* дорівнює *d(vz,v0)= 29.*

### *Завдання 4. З*находження найкоротшого шляху

Для знаходження найкоротшого шляху необхідно починати з вершини v0 і вибирати з доступних вершин таку індекс якої співпадав з різницею індексу v0 і довжиною ребра між ними.

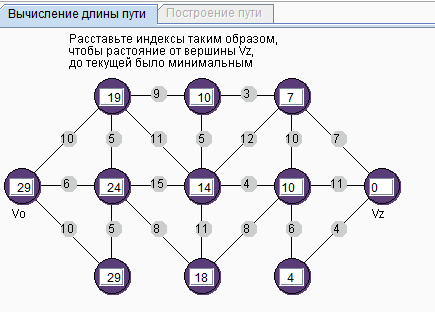
29-6=23 не 24

29-10=19 викон

29-10=19 не 29

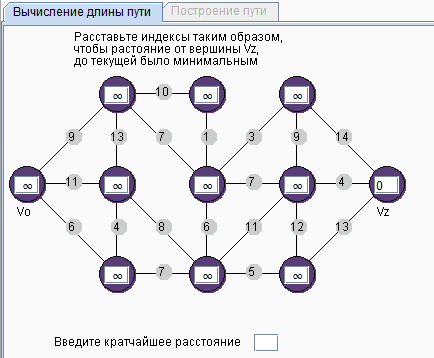
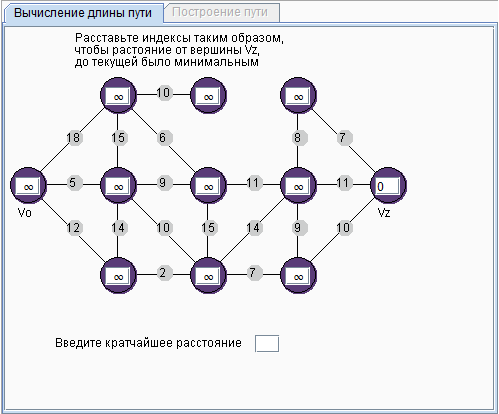
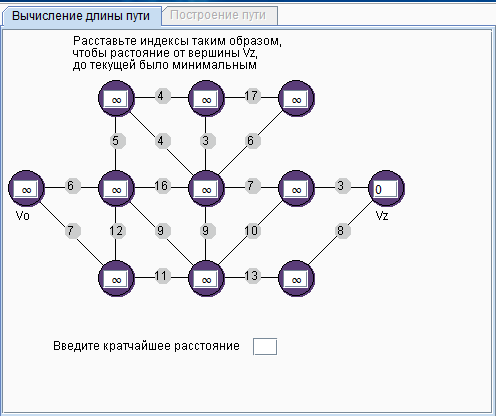
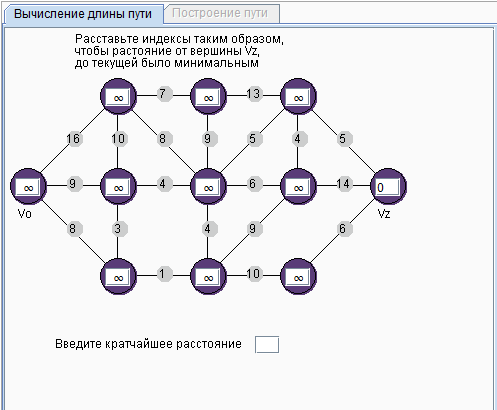
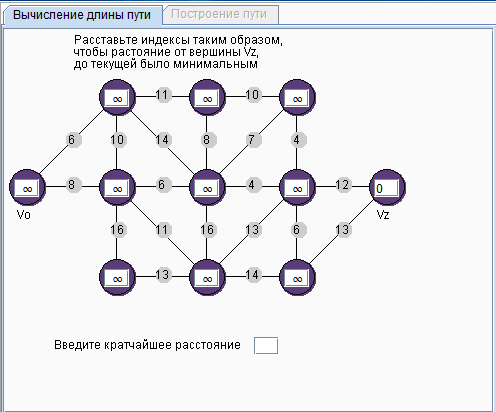
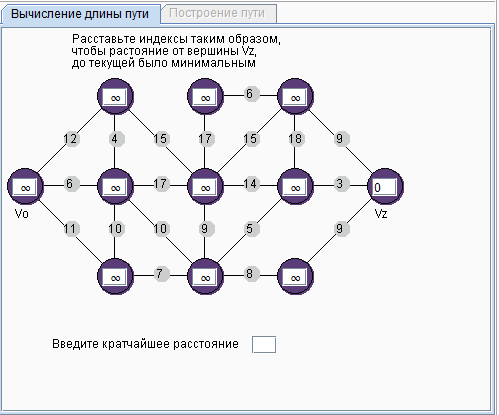
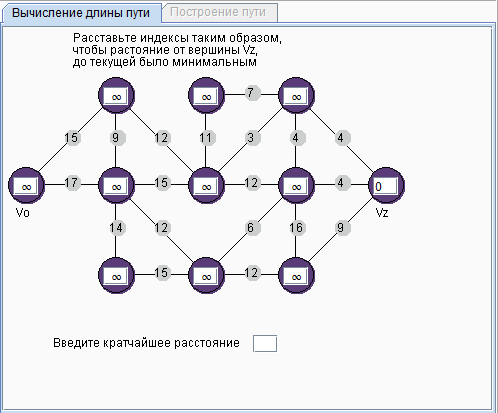
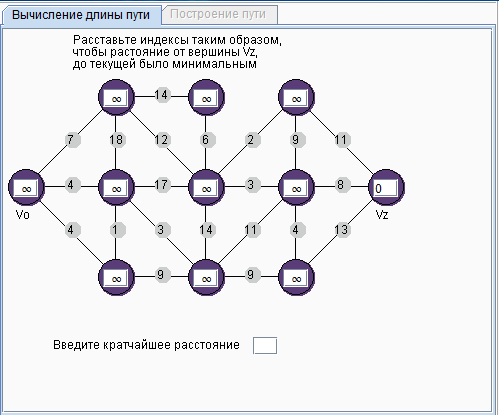
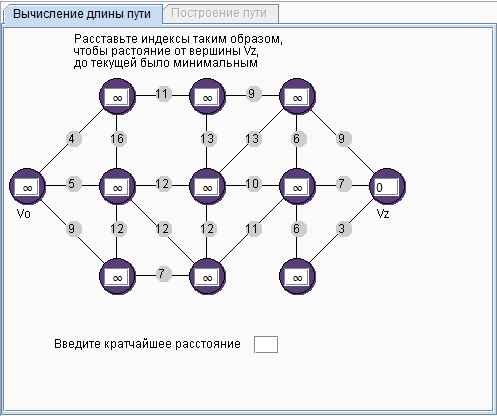
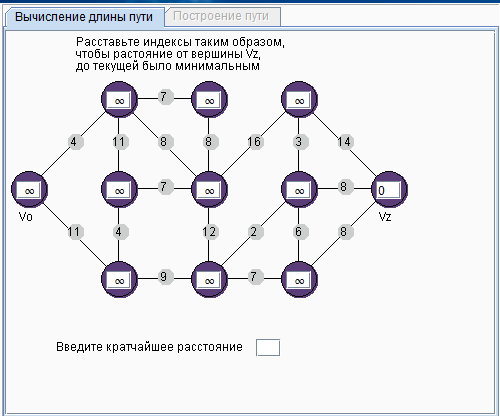
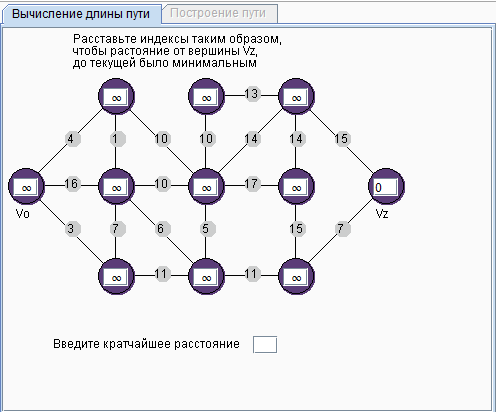
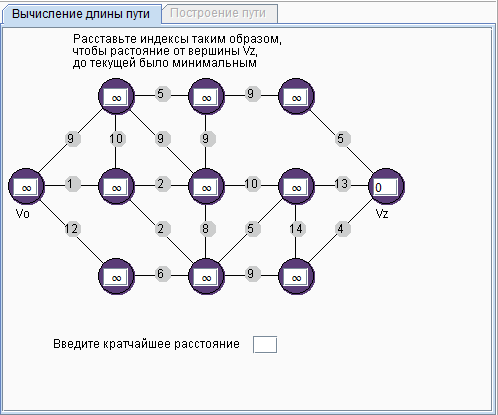
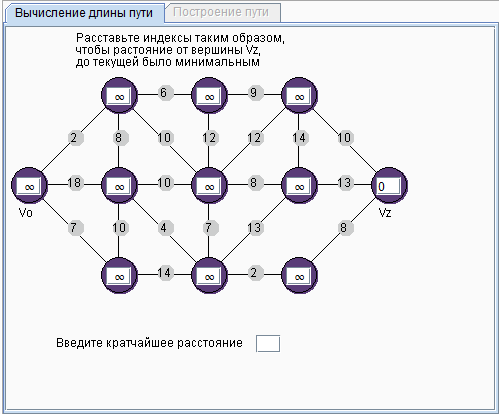
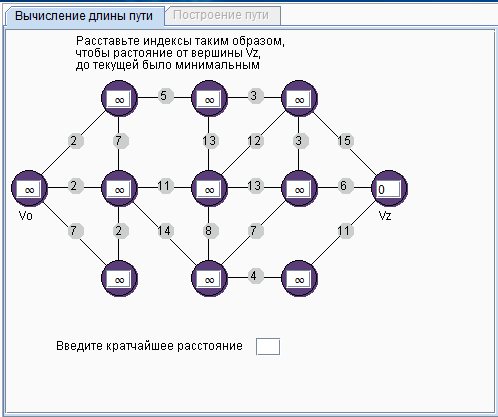
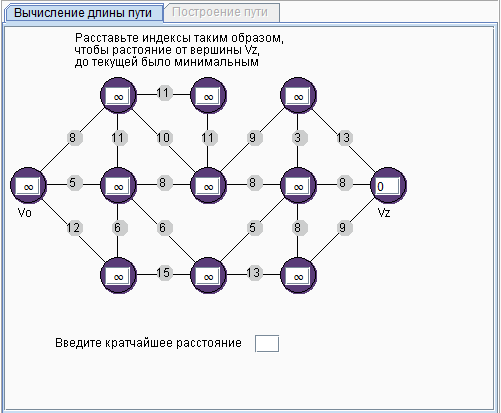
19-9=10 викон

19-11=8 не 14



Найкоротша відстань з вершини *Vz* до вершини *Vo* задається маршрутом зображеним стрілками.

## УМОВИ ПО ВАРІАНТАХ - ЗВАЖЕНІ ГРАФИ.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 