Пошук найкоротшого шляху

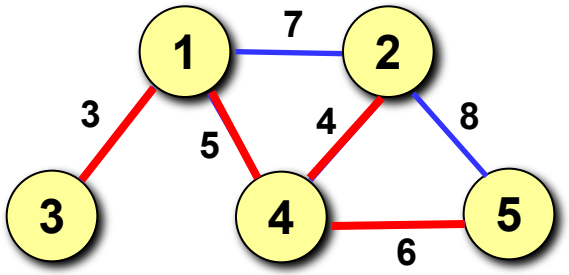
Графи

Переклад: Р. М. Васильчик

Задача Прима-Краскала

Завдання: з'єднати **N** міст телефонною мережею так, щоб довжина телефонних ліній була мінімальною.

Те ж завдання: дано зв'язний граф з **N** вершинами, веги ребер задані ваговою матрицею **W**. Потрібно знайти набір ребер, що з'єднує всі вершини графа (***остовне дерево***) і має найменшу вагу.



	1	2	3	4	5
1	0	7	3	5	∞
2	7	0	∞	4	8
3	3	∞	0	∞	∞
4	5	4	∞	0	6
5	∞	8	∞	6	0

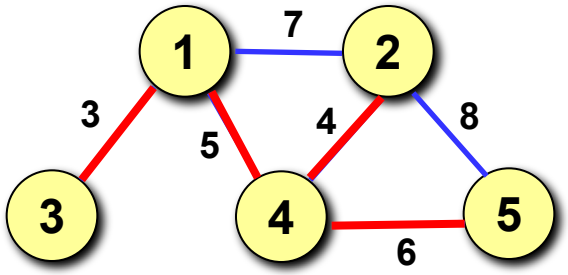
Жадібний алгоритм

Жадібний алгоритм – це багатокроковий алгоритм, в якому на кожному кроці приймається рішення, краще в даний момент.



В цілому може вийти не оптимальне рішення (послідовність кроків)!

Крок в задачі Прима-Краскала – це вибір ще невибраного ребра і додавання його до рішення.

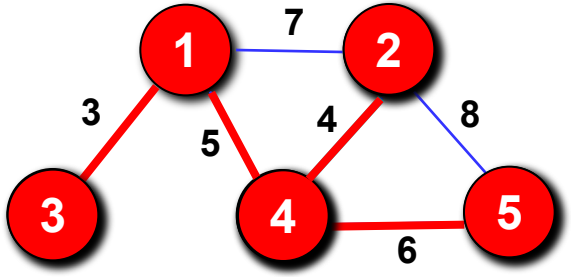


В задачі Прима-Краскала жадібний алгоритм дає оптимальне рішення!

Реалізація алгоритму Прима-Краскала

Проблема: як перевірити, що
1) ребро не вибрано, і
2) ребро не утворює циклу з вибраними ребрами.

Рішення: присвоїти кожній вершині свій колір і перефарбовувати вершини при додаванні ребра.



Алгоритм:

1) пофарбувати всі вершини в різні кольори;
2) зробити **`N-1`** раз в циклі:
   - вибрати ребро **`(i,j)`** мінімальної довжини з усіх ребер, що з'єднують вершини різного кольору;
   - перефарбувати всі вершини, що мають колір **`j`**, в колір **`i`**.
3) вивести знайдені ребра.

Реалізація алгоритму Прима-Краскала

Структура «ребро»:

```
type rebro = record
       i, j: integer; { номери вершин }
     end;
```

Основна програма:

вагова матриця

колір вершин

```
const N = 5;
var W: array[1..N,1..N] of integer;
    Color: array[1..N] of integer;
    i, j, k, min, col_i, col_j: integer;
    Reb: array[1..N-1] of rebro;
begin
...     { тут треба ввести матрицю W }
for i:=1 to N do { розфарбувати в різні кольори }
  Color[i] := i;
... { основний алгоритм – заповнення масиву Reb }
... { вивести знайдені ребра (масив Reb)}
end.
```

Реалізація алгоритму Прима-Краскала

Основний алгоритм:

```
for k:=1 to N-1 do begin
  min := MaxInt;
  for i:=1 to N do
    for j:=i+1 to N do
     if (Color[i] <> Color[j]) and
        (W[i,j] < min) then begin
       min := W[i,j];
    Reb[k].i := i;
       Reb[k].j := j;
       col_i := Color[i];
       col_j := Color[j];
     end;
  for i:=1 to N do
    if Color[i] = col_j then
       Color[i] := col_i;
end;
```

потрібно вибрати всього N-1 ребро

цикл по всіх парах вершин

враховуємо тільки пари з різним кольором вершин

запам'ятовуємо ребра і кольори вершин

перефарбовуєм вершини кольору col_j

Складність алгоритму

Основний цикл:

```
for k:=1 to N-1 do begin
  ...
  for i:=1 to N do
    for j:=i+1 to N do
    ...
end;
```

три вкладених цикли, в кожному кількість кроків <=N

Кількість операцій:

$O(N^3)$ зростає не швидше, ніж N^3

Необхідна пам'ять:

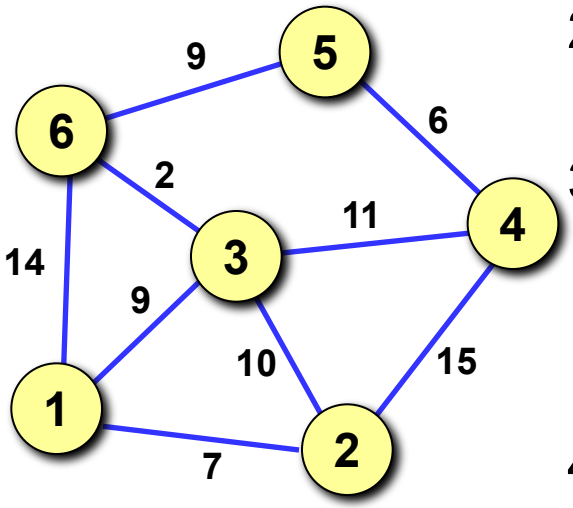
```
var W: array[1..N,1..N] of integer;
    Color: array[1..N] of integer;
    Reb: array[1..N-1] of rebro;
```

➡ $O(N^2)$

Найкоротші шляхи (алгоритм Дейкстри)

Завдання: задана мережа доріг між містами, частина яких можуть мати односторонній рух. Знайти найкоротші відстані від заданого міста до всіх інших міст.

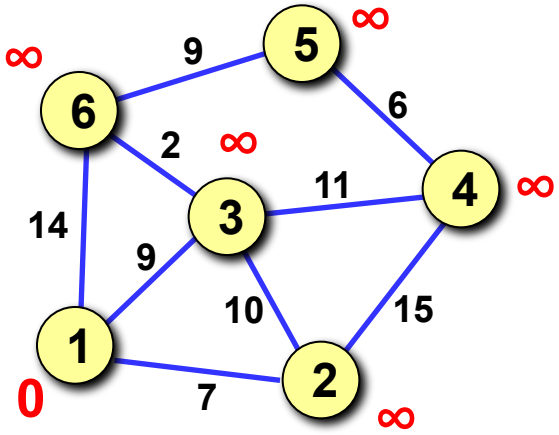
Та же завдання: дано зв'язний граф з **N** вершинами, ваги ребер задані матрицею **W**. Знайти найкоротші відстані від заданої вершини до всіх інших.

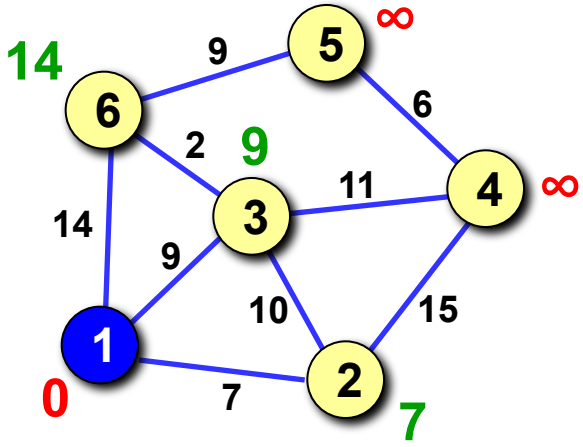


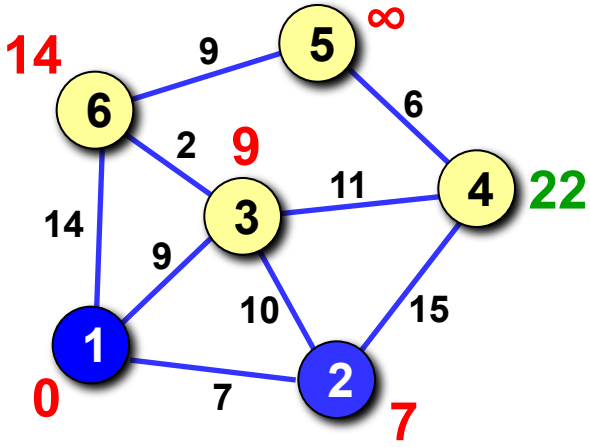
Алгоритм Дейкстри (*E.W. Dijkstra*, 1959)

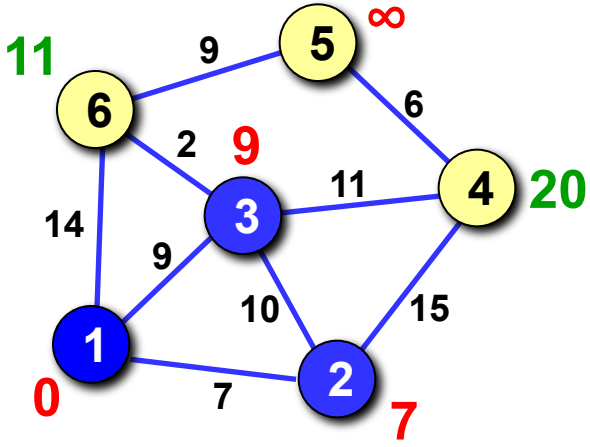
1) присвоїти всім вершинам мітку ∞;
2) серед нерозглянутих вершин знайти вершину **j** з найменшою міткою;
3) для кожної необробленої вершини **i**: якщо шлях до вершини **i** через вершину **j** менше існуючої мітки, замінити мітку на нову відстань;
4) якщо залишилися необроблені вершини, перейти до кроку 2;
5) мітка = мінімальна відстань.

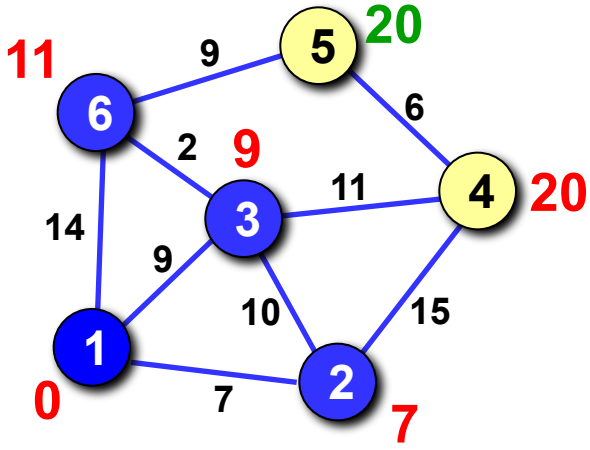
Алгоритм Дейкстри

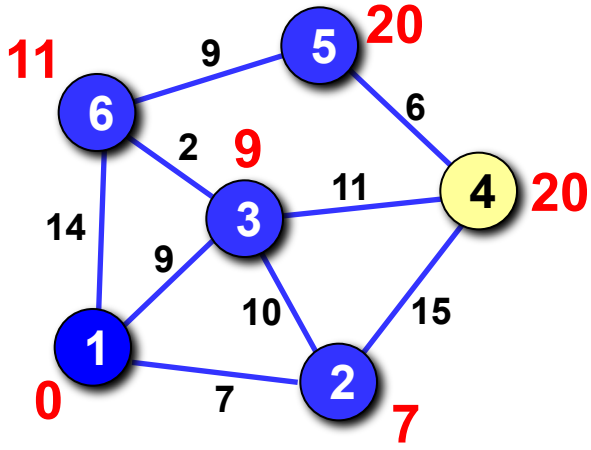












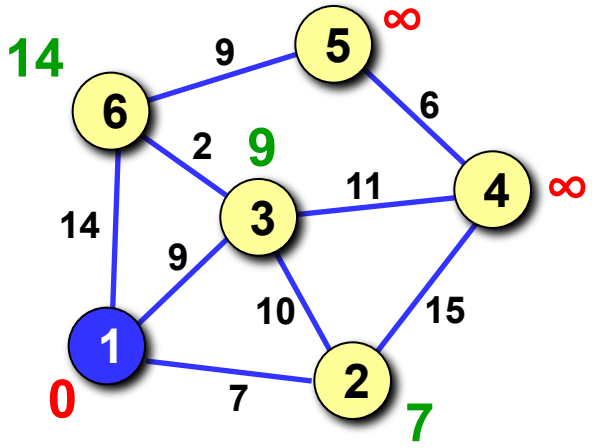
Реалізація алгоритму Дейкстри

Масиви:

1) масив **a**, такий що **a[i]=1**, якщо вершина вже розглянута, і **a[i]=0**, якщо ні.
2) масив **b**, такий що **b[i]** – довжина поточного найкоротшого шляху з заданої вершини **x** в вершину **i**;
3) масив **c**, такий що **c[i]** – номер вершини, з якої потрібно йти в вершину **i** в поточному найкоротшому шляху.

Ініціалізація:

4) заповнити масив **a** нулями (вершини не оброблені);
5) записати в **b[i]** значення **W[x][i]**;
6) заповнити масив **c** значеннями **x**;
7) записати **a[x]=1**.



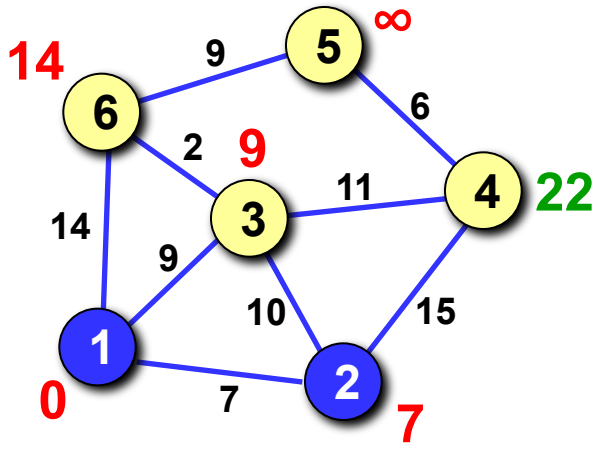
	1	2	3	4	5	6
a	1	0	0	0	0	0
b	0	7	9	∞	∞	14
c	0	0	0	0	0	0

Реалізація алгоритму Дейкстри

Основний цикл:

1) якщо всі вершини розглянуті, то стоп.
2) серед всіх нерозглянутих вершин (**a[i]=0**) знайти вершину **j**, для якої **b[i]** – мінімальне;
3) записати **a[j]:=1**;
4) для всіх вершин **k**: якщо шлях в вершину **k** через вершину **j** коротший, ніж знайдений раніше найкоротший шлях, запам'ятати його: записати **b[k]:=b[j]+W[j,k]** і **c[k]=j**.

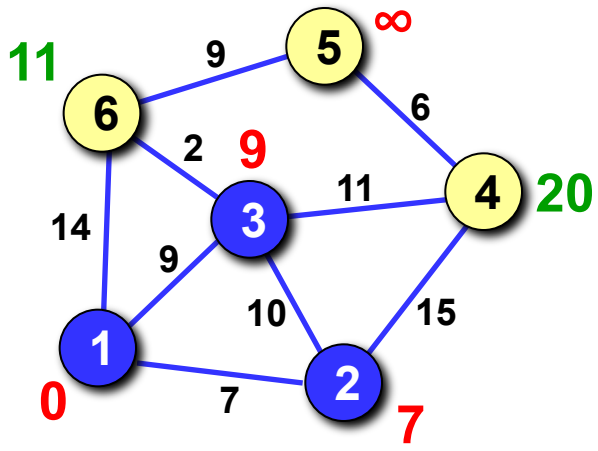
Крок 1:



	1	2	3	4	5	6
a	1	1	0	0	0	0
b	0	7	9	22	∞	14
c	0	0	0	1	0	0

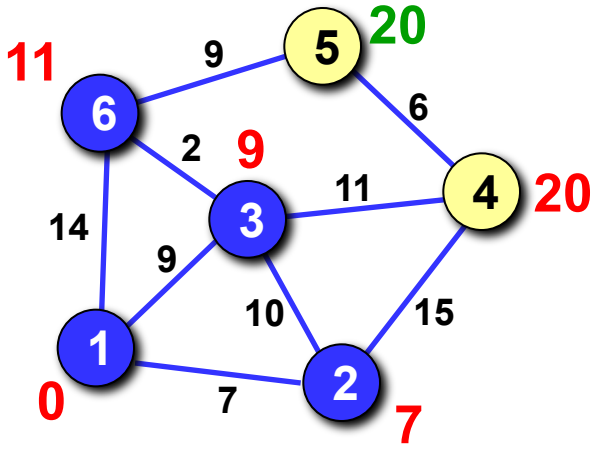
Реалізація алгоритму Дейкстри

Крок 2:



	1	2	3	4	5	6
a	1	1	1	0	0	0
b	0	7	9	20	∞	11
c	0	0	0	2	0	2

Крок 3:



	1	4	3	4	5	6
a	1	1	1	0	0	1
b	0	7	9	20	20	11
c	0	0	0	2	5	2

Далі масиви не змінюються!

Як вивести маршрут?

Результат роботи алгоритму Дейкстри:

	1	2	3	4	5	6
a	1	1	1	1	1	1
b	0	7	9	20	20	11
c	0	0	0	2	5	2

довжини шляхів

Маршрут з вершини 0 в вершину 4:

4 ➡ 5 ➡ 2 ➡ 0

Виведення маршруту в вершину `i` (використання масиву `c`):

1) встановити **`z:=i`**;
2) поки **`c[i]<>x`** присвоїти **`z:=c[z]`** і вивести **`z`**.

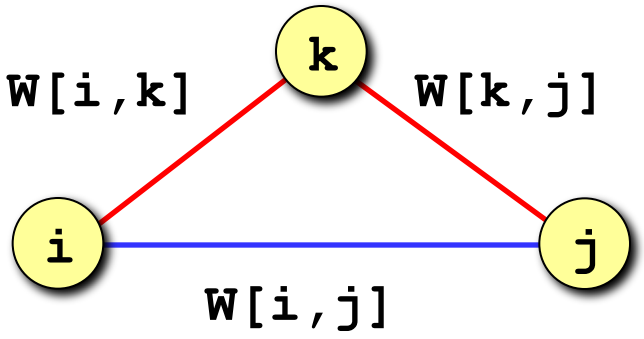
Складність алгоритму Дейкстри:

два вкладених цикли по **N** кроків $O(N^2)$

Алгоритм Флойда-Уоршелла

Завдання: задана мережа доріг між містами, частина яких може мати односторонній рух. Знайти **всі** **найкоротші відстані**, від кожного міста до всіх інших міст.

```
for k:=1 to N
  for i:=1 to N
    for j:=1 to N
      if W[i,j] > W[i,k]+W[k,j] then
          W[i,j]:=W[i,k]+W[k,j];
```



Якщо з вершини **i** в вершину **j** коротше їхати через вершину **k**, ми їдемо через вершину **k**!

Немає інформації про маршрут, тільки найкоротші відстані!

Алгоритм Флойда-Уоршелла

Версія з запам'ятовуванням маршруту:

```
for i:=1 to N
  for j :=1 to N
    c[i,j] :=i;
...
for k: =1 to N
  for i: =1 to N
    for j: =1 to N
      if W[i,j] > W[i,k] + W[k,j] then begin
        W[i,j] := W[i,k] + W[k,j];
        c[i,j] := c[k,j];
      end;
```

i–ий рядок будується так само, як масив **c** в алгоритмі Дейкстри

в кінці циклу **c[i,j]** – передостання вершина в найкоротшому маршруті з вершини **i** в вершину **j**

Яка складність алгоритму?

$O(N^3)$

Завдання комівояжера

Завдання комівояжера. Комівояжер (бродячий торговець) повинен вийти з першого міста і, відвідавши по разу в невідомому порядку міста `2,3,...N`, повернуться назад в перше місто. У якому порядку треба обходити міста, щоб замкнутий шлях (тур) комівояжера був найкоротший?

Це NP-повна задача, яка строго вирішується тільки перебором варіантів (поки що)!

Точні методи:

1) простий перебір;
2) метод гілок і меж;
3) метод Літтла;
4) …

великий час рахунку для великих **N**

`O(N!)`

Наближені методи:

5) метод випадкових перестановок (*Matlab*);
6) генетичні алгоритми;
7) метод мурашиних колоній;
8) …

не гарантується оптимальне рішення

Інші класичні завдання

Завдання на мінімум суми. Є **N** населених пунктів, у кожному з яких живе p_i школярів (**i=1,...,N**). Треба розмістити школу в одному з них так, щоб загальна відстань, яку проходять всі учні по дорозі в школу, була мінімальною.

Завдання про найбільший потік. Є система труб, які мають з'єднання в **N** вузлах. Один вузол **S** є джерелом, ще один – стоком **T**. Відомі пропускні спроможності кожної труби. Треба знайти найбільший потік від джерела до стоку.

Завдання про найбільше парасполучення. Є **M** чоловіків і **N** жінок. Кожен чоловік вказує декілька (від **0** до **N**) жінок, на яких він згоден одружуватися. Кожна жінка вказує кілька чоловіків (від **0** до **M**), за яких вона згодна вийти заміж. Потрібно укласти найбільшу кількість моногамних шлюбів.