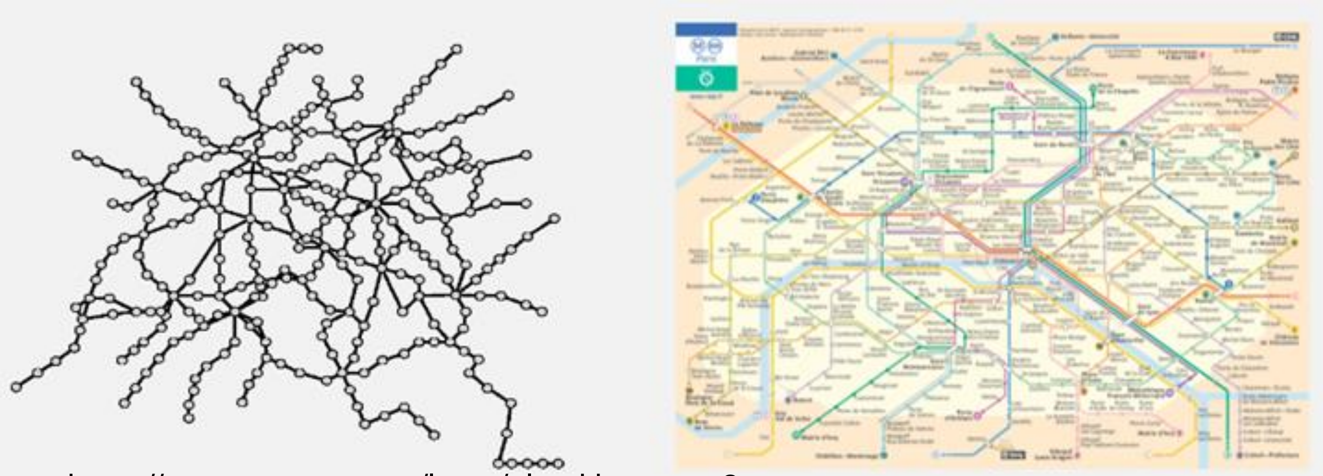
Графи



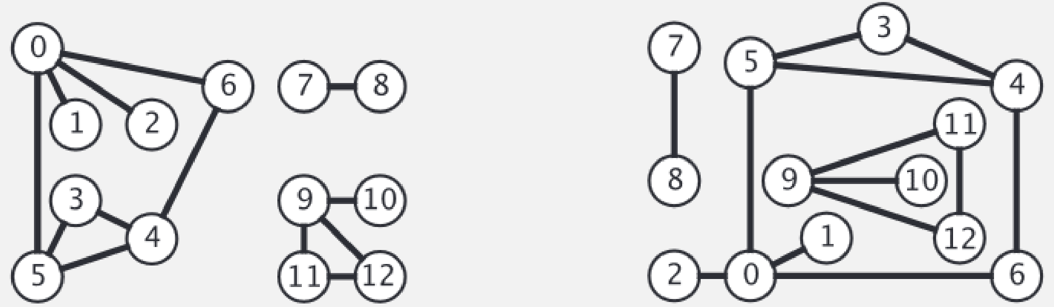
https://www.coursera.org/learn/algorithms-part2

Граф – це сукупність об’єктів (вершини графа) та зв’язків між ними (ребра графа)

$$G = \langle V, E \rangle$$

Vertex
(Вершини)

Edge
(Ребра)

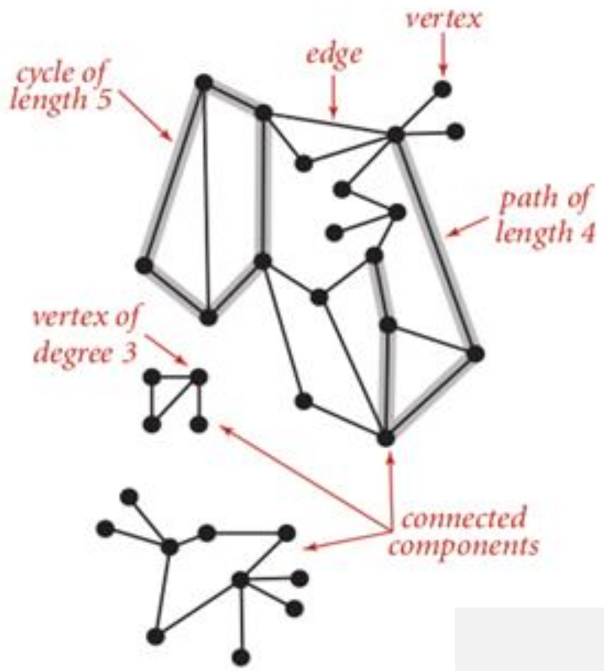


two drawings of the same graph

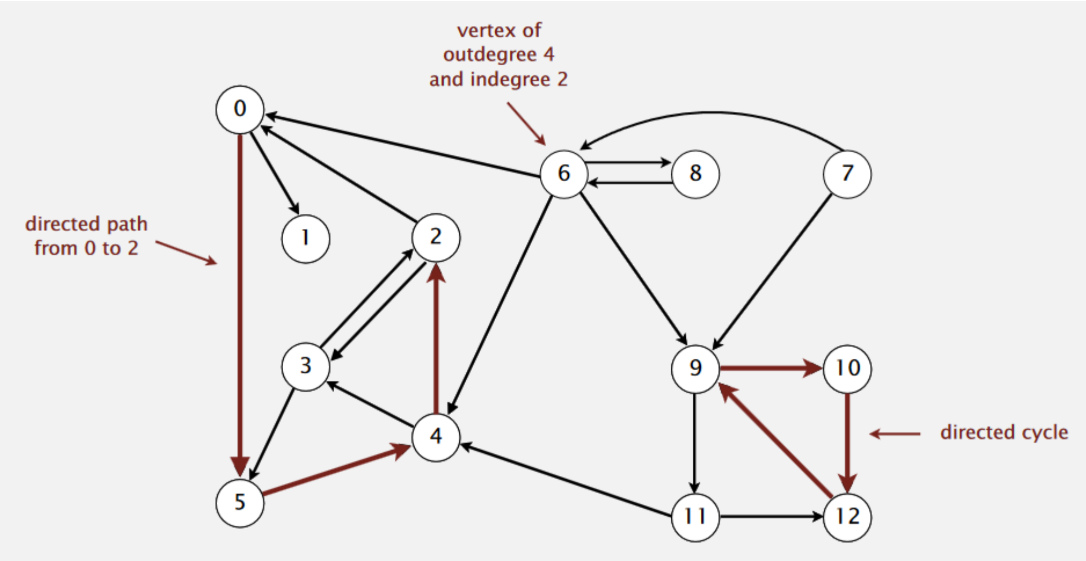
$|V|$ – кількість вершин графа

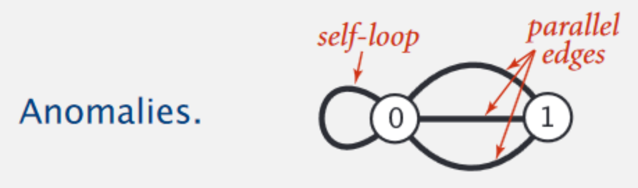
$|E|$ – кількість ребер графа

Неорієнтований граф

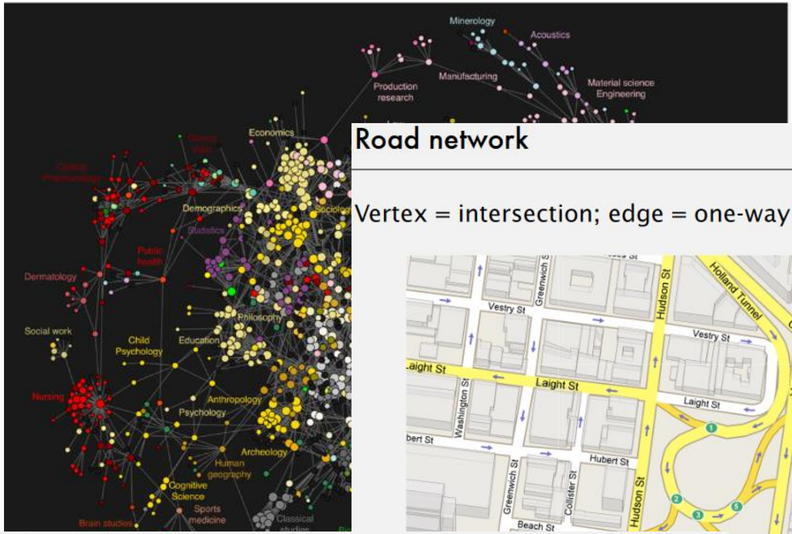


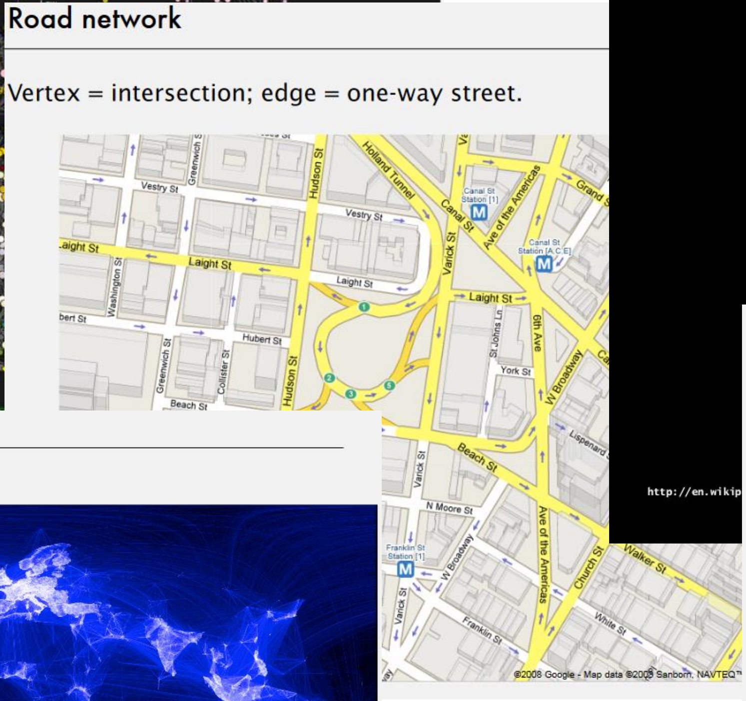
Орієнтований граф

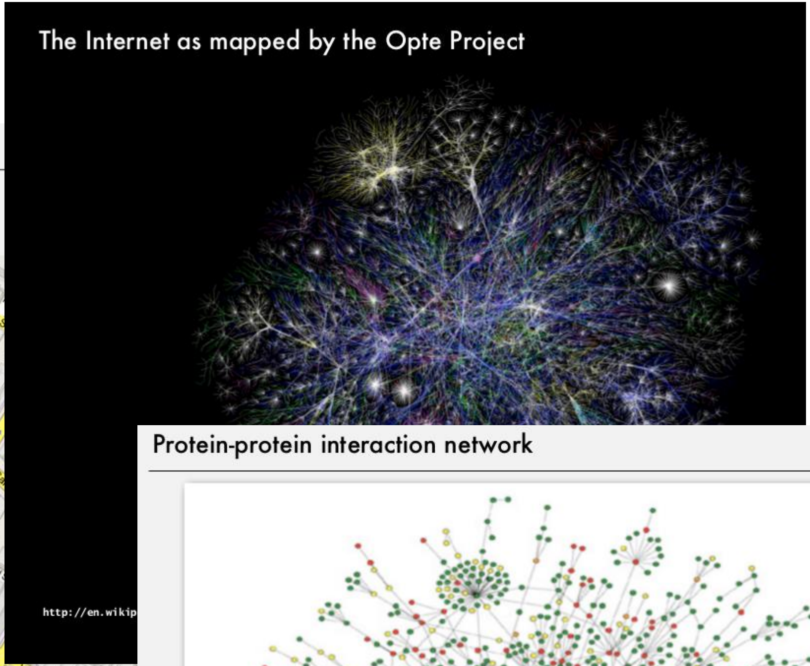


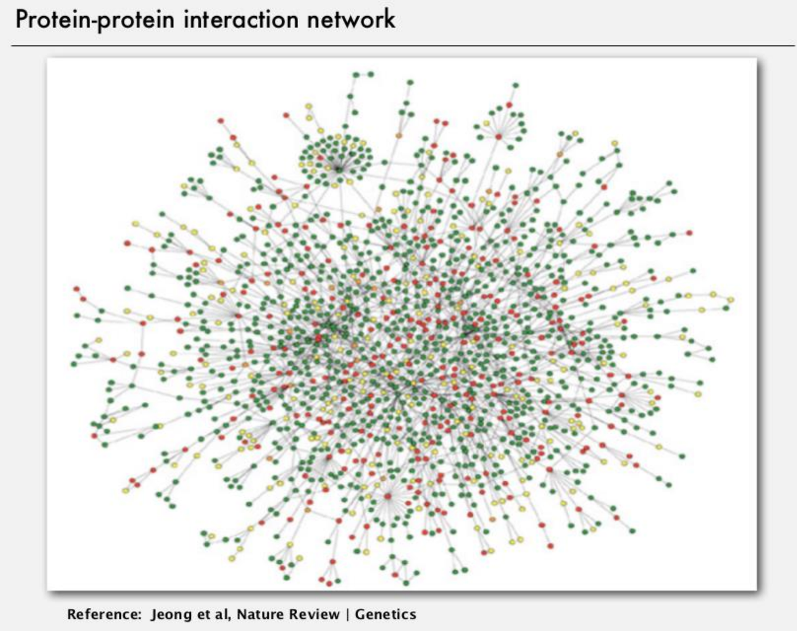


https://www.coursera.org/learn/algorithms-part2









https://www.coursera.org/learn/algorithms-part2

Graph applications

graph	vertex	edge
communication	telephone, computer	fiber optic cable
circuit	gate, register, processor	wire
mechanical	joint	rod, beam, spring
financial	stock, currency	transactions
transportation	street intersection, airport	highway, airway route
internet	class C network	connection
game	board position	legal move
social relationship	person, actor	friendship, movie cast
neural network	neuron	synapse
protein network	protein	protein-protein interaction
molecule	atom	bond

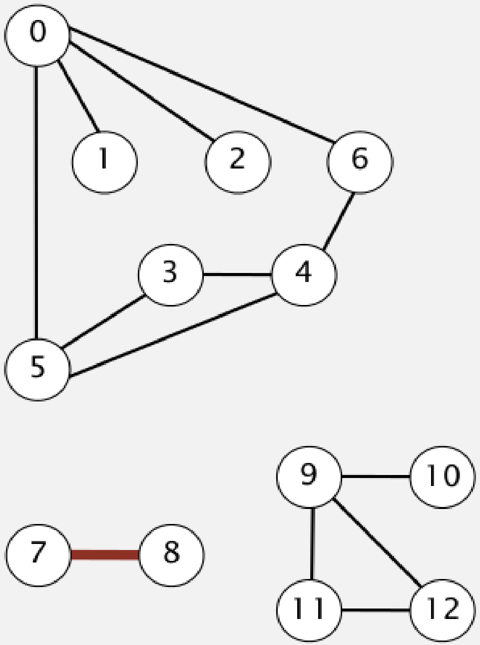
Digraph applications

digraph	vertex	directed edge
transportation	street intersection	one-way street
web	web page	hyperlink
food web	species	predator-prey relationship
WordNet	synset	hypernym
scheduling	task	precedence constraint
financial	bank	transaction
cell phone	person	placed call
infectious disease	person	infection
game	board position	legal move
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

https://www.coursera.org/learn/algorithms-part2

Set-of-edges graph representation

Maintain a list of the edges (linked list or array).

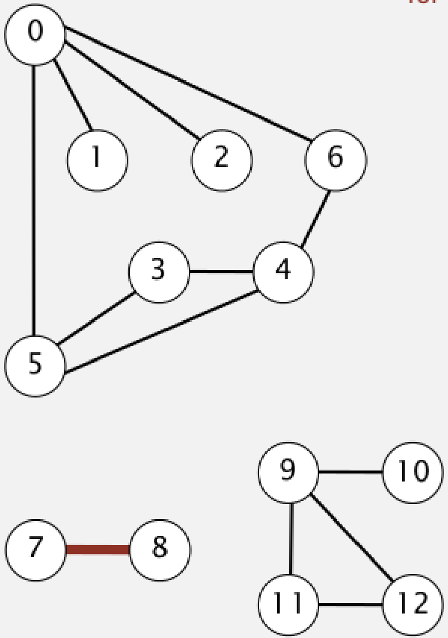


0	1
0	2
0	5
0	6
3	4
3	5
4	5
4	6
7	8
9	10
9	11
9	12
11	12

Adjacency-matrix graph representation

Maintain a two-dimensional V-by-V boolean array;

for each edge $v–w$ in graph: `adj[v][w] = adj[w][v] = true`.

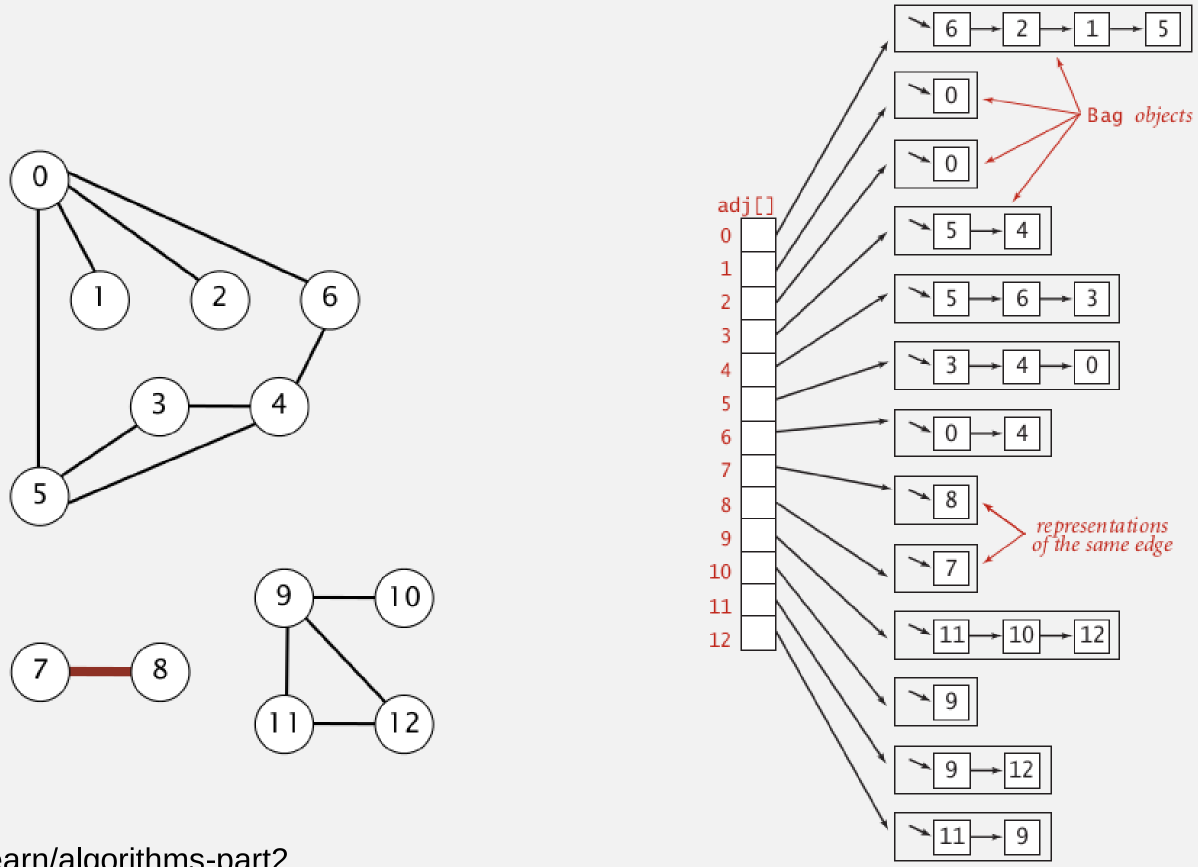


two entries
for each edge

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency-list graph representation

Maintain vertex-indexed array of lists.



Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

huge number of vertices,
small average vertex degree

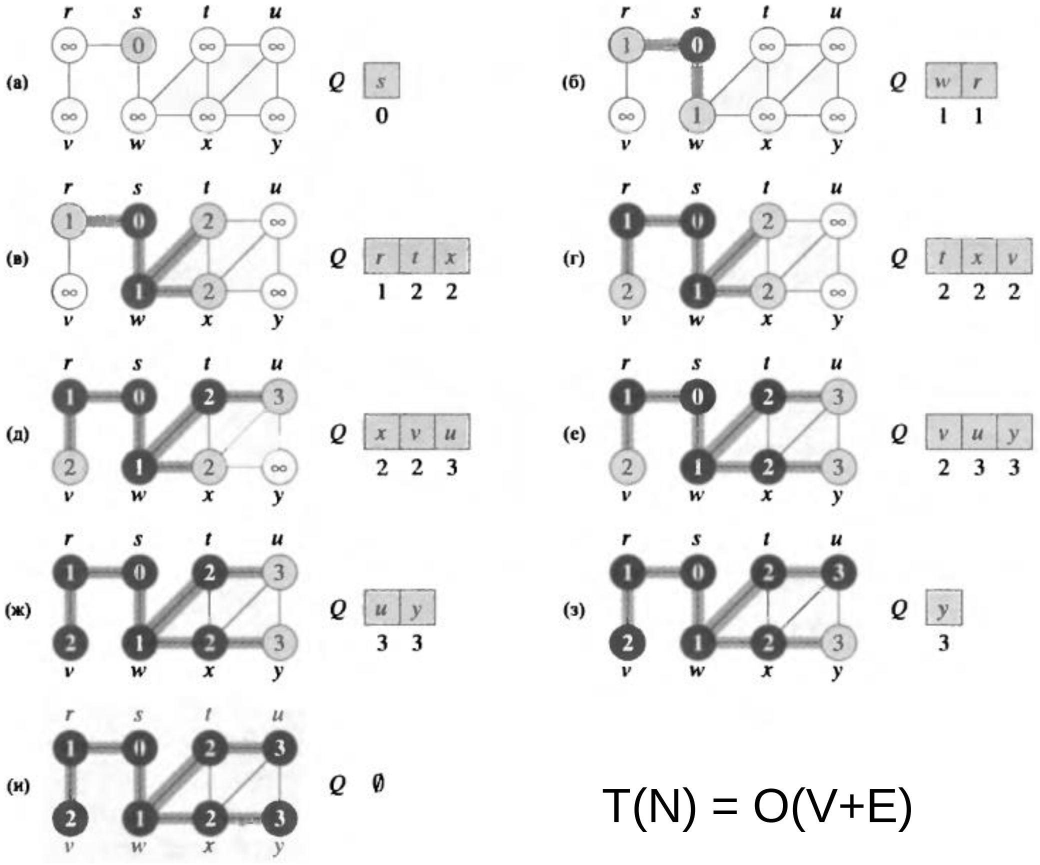
representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	E + V	1	degree(v)	degree(v)

* disallows parallel edges

Пошук вшир

Breadth-first search

```
BFS(G, s)
for Каждой вершины u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for Каждой вершины v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



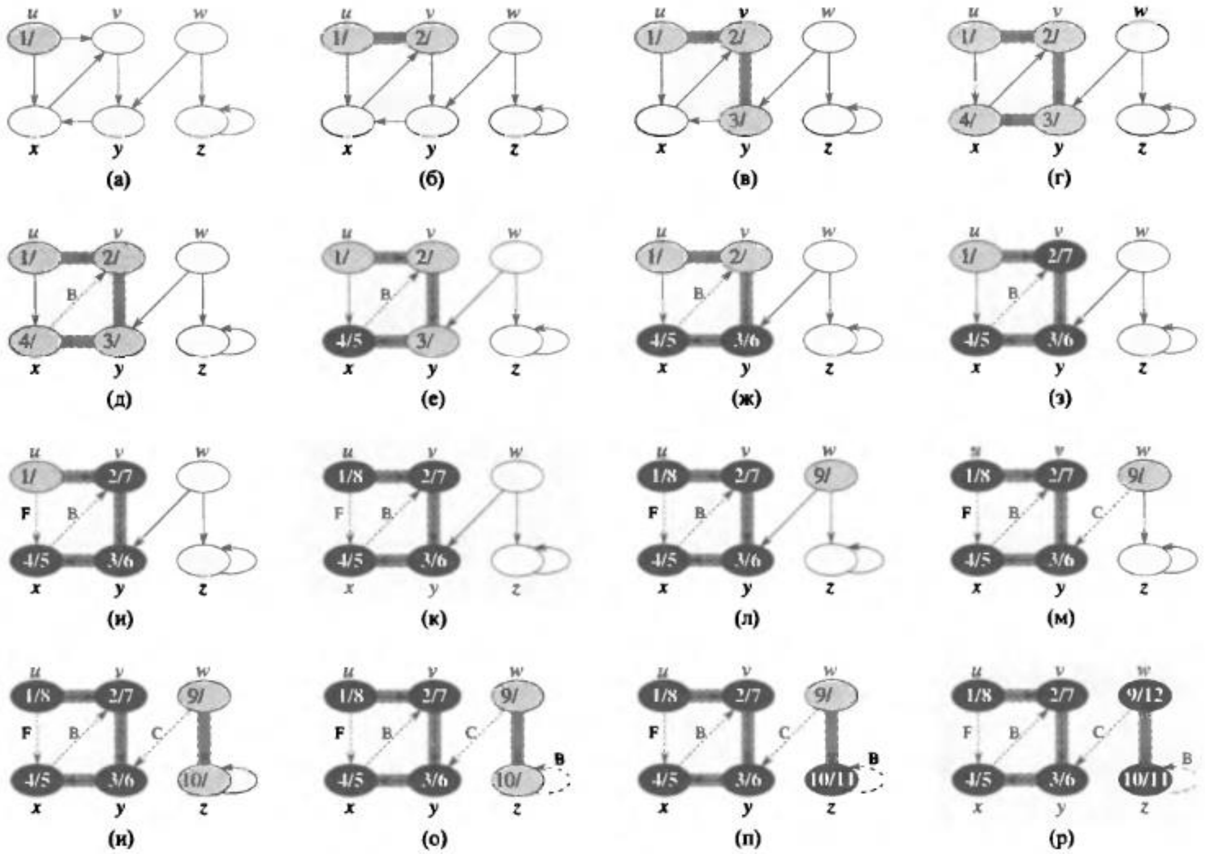
$T(N) = O(V+E)$

Пошук вглиб

Depth-first search

```
DFS(G)
for каждой вершины u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for каждой вершины u ∈ G.V
    if u.color == WHITE
        DFS-Visit(G, u)
```

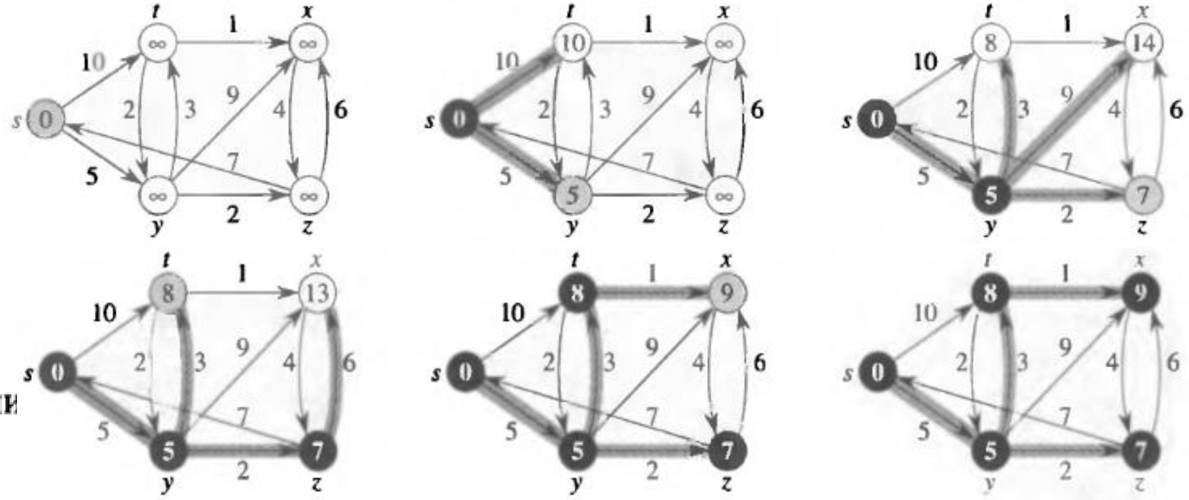
```
DFS-Visit(G, u)
time = time + 1
u.d = time
u.color = GRAY
for каждой v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-Visit(G, v)
u.color = BLACK
time = time + 1
u.f = time
```



$T(N) = O(V+E)$

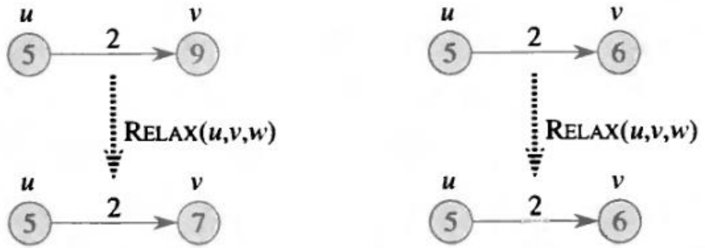
Пошук найкоротшого шляху. Алгоритм Дейкстри

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for каждой вершины v ∈ G.Adj[u]
              RELAX(u, v, w)
              // С соответствующими вызовами
                 DECREASE-KEY
```




```
INITIALIZE-SINGLE-SOURCE(G, s)
  for каждой вершины v ∈ G.V
        v.d = ∞
        v.π = NIL
  s.d = 0
```

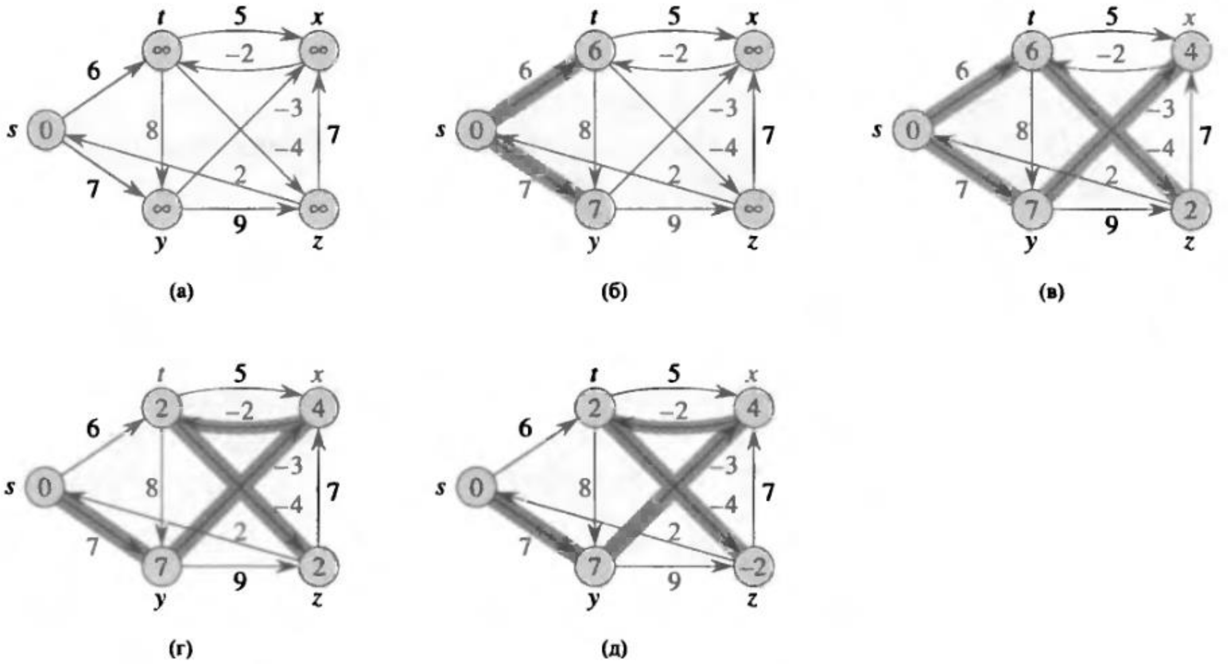
```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
        v.d = u.d + w(u, v)
        v.π = u
```



Пошук найкоротшого шляху. Алгоритм Беллмана-Форда

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for каждого ребра (u, v) ∈ G.E
        RELAX(u, v, w)
for каждого ребра (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$

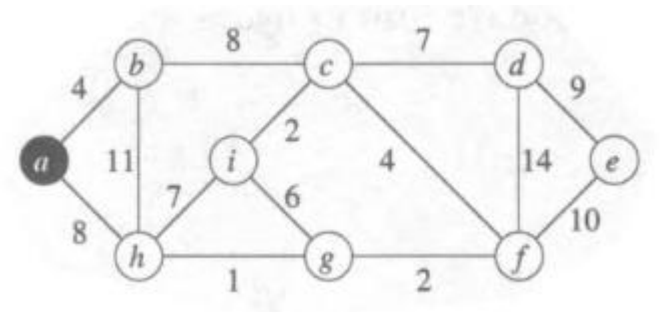


(а) (б) (в) (г) (д)

Побудова мінімального кістякового дерева. Алгоритм Пріма

MST-PRIM(G, w, r)

for каждой вершины $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for каждой вершины $v \in G.Adj[u]$
        if $v \in Q$ и $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$
            // С вызовом DECREASE-KEY$(Q, v, w(u, v))$



Алгоритм Пріма. Приклад