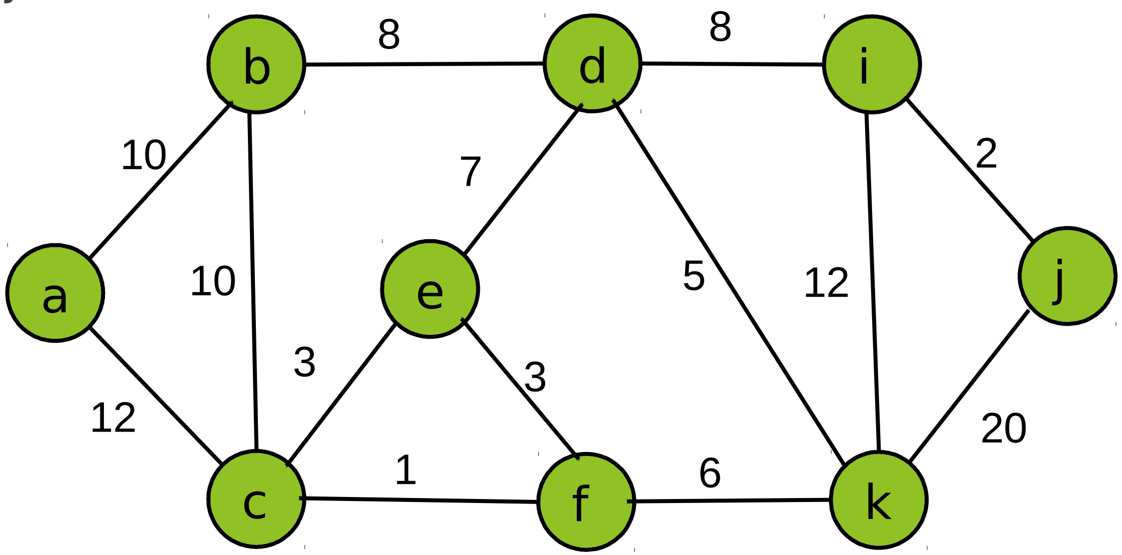
Graph - 03
Minimum Spanning Tree

Giảng viên: Tạ Việt Cường
Phòng HMI – Khoa CNTT

Ôn tập

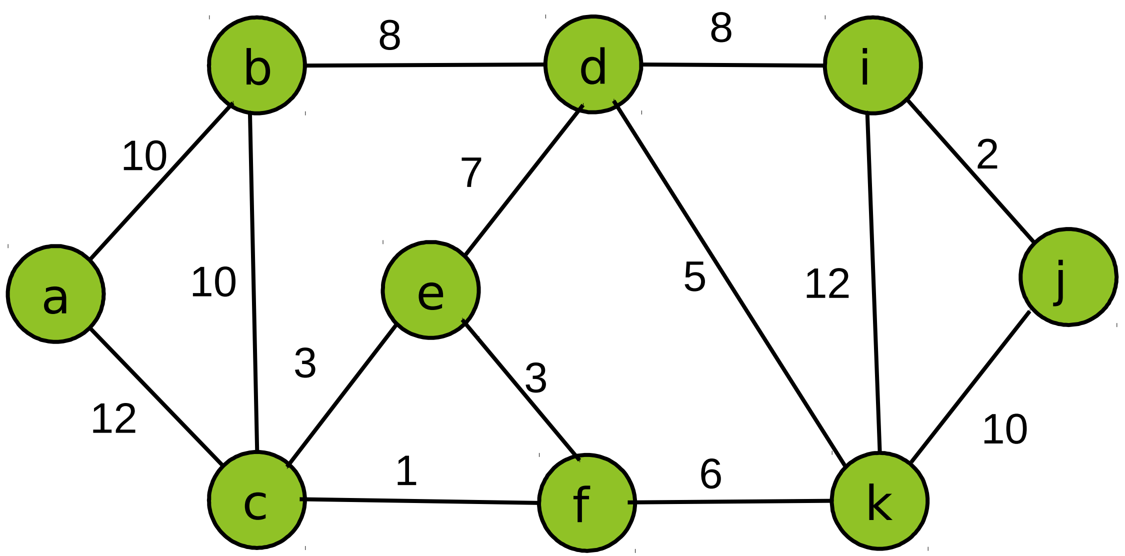
- Tìm đường đi ngắn nhất từ a đến j
- In ra danh sách các đỉnh được mở rộng theo thuật toán Dijkstra



Cây khung nhỏ nhất Minimum Spanning Tree (MST)

- MST là một bài toán nền tảng trên đồ thị **vô hướng** có **trọng số**.
- Bài toán thực tế:
  - Xây dựng mạng lưới giao thông với chi phí ít nhất
  - Có N thành phố, để xây đường bộ 2 chiều đi từ thành phố a đến thành phố b tốn C(a, b) (VND)
  - Làm thế nào xây mạng lưới giao thông để có thể đi được giữa N thành phố và tốn ít chi phí nhất

Cây khung nhỏ nhất



❖ Chú ý: Đồ thị ban đầu phải liên thông

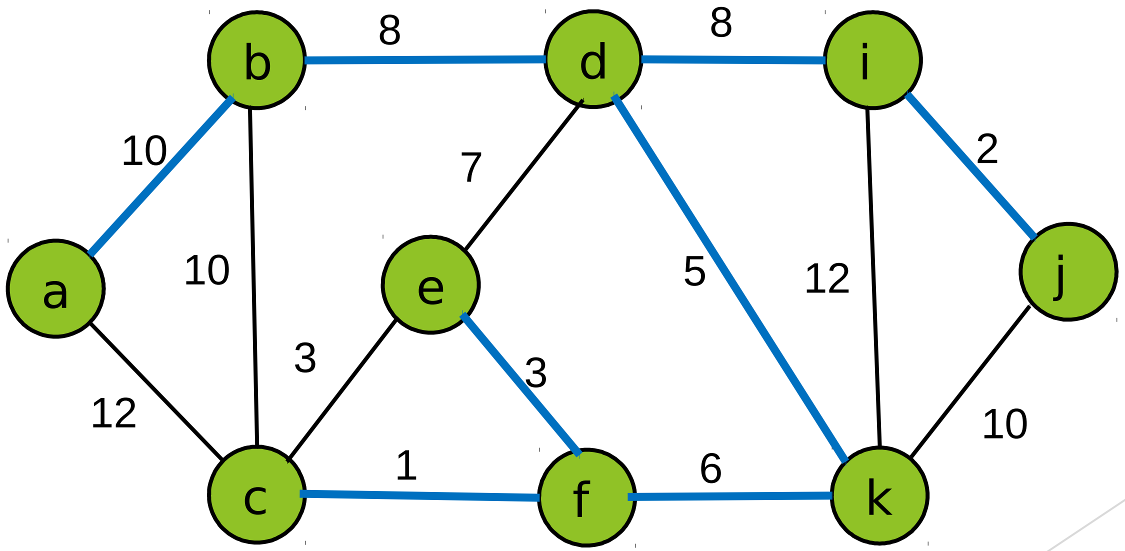
Cây khung nhỏ nhất

- Định nghĩa:
  - Cây khung là cây chứa tất cả các đỉnh của đồ thị và không có chu trình
  - Có thể tìm cây khung bằng BFS hoặc DFS (how?)
- Cây khung nhỏ nhất:
  - Tổng các trọng số nhỏ nhất

- Chú ý: Đồ thị vô hướng và liên thông

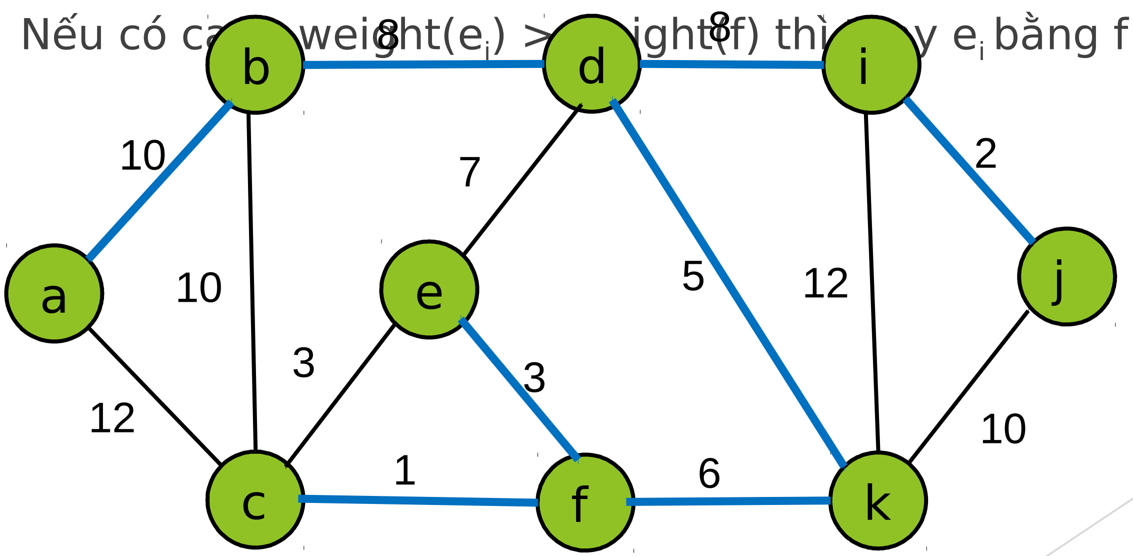
Cây khung nhỏ nhất

- Tính chất của cây khung:
  - Gọi T = danh sách các cạnh thuộc cây khung
  - T = e_1, e_2, ..., e_{N-1}
- Bỏ bất cứ e_i sẽ thu được không phải là cây khung nữa
- Thêm vào bất cứ cạnh nào sẽ tạo thành chu trình



Cây khung nhỏ nhất

- Định lí 1: Gọi T = e_1, e_2, ..., e_{N-1} là cây khung nhỏ nhất
  - Giả sử thêm cạnh f nối u đến v và tạo thành chu trình C
  - Thì trọng số của f >= tất cả các cạnh còn lại của chu trình C
  - Nếu có cạnh weight(e_i) > weight(f) thì thay e_i bằng f

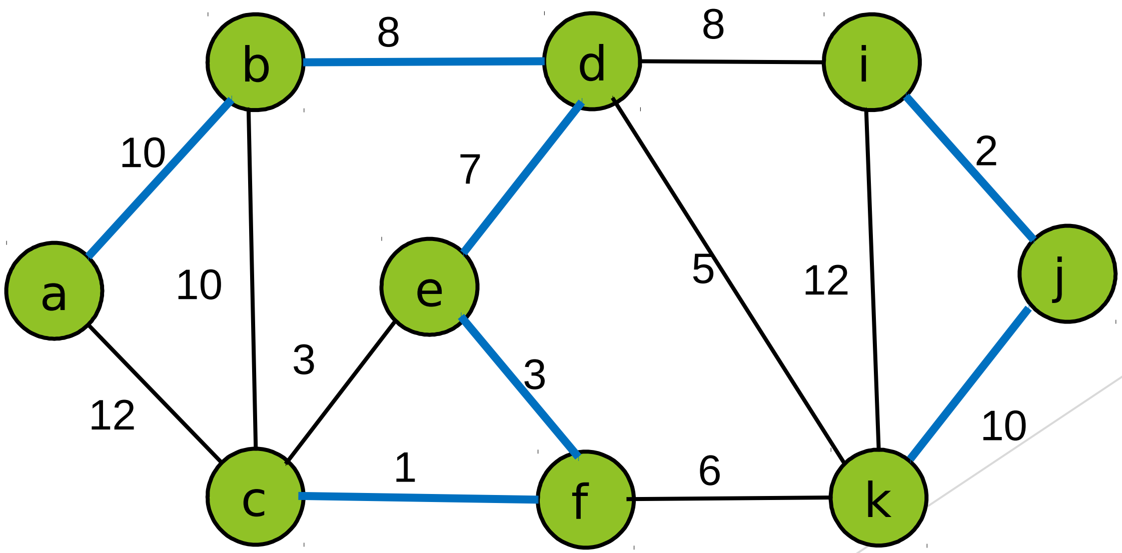


Cây khung nhỏ nhất

- Định lý 2:
  - Cho đồ thị có N đỉnh, N đỉnh được chia thành 2 tập hợp U và V rời nhau

    U = {a, b, c, d, e, f}

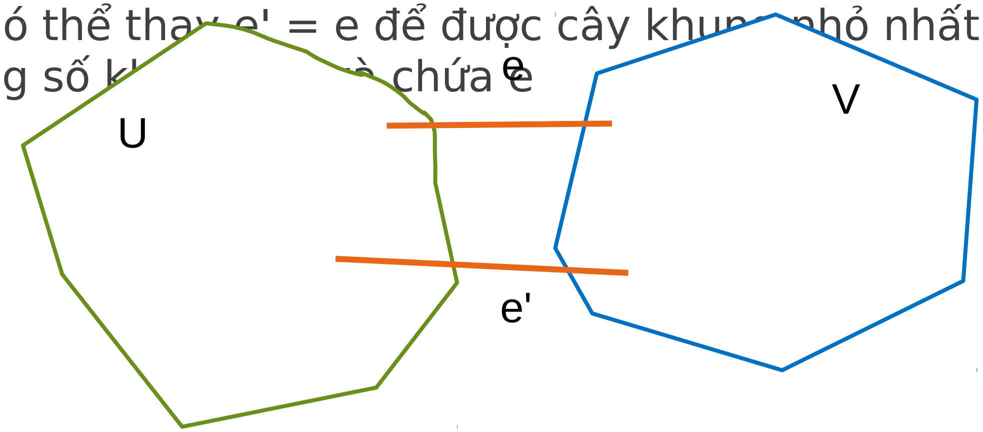
    V = {i, j, k}
  - Gọi e là cạnh nhỏ nhất trong tất cả các cặp cạnh có thể nối giữa u thuộc U và v thuộc V
  - Tồn tại cây khung nhỏ nhất chứa e



Cây khung nhỏ nhất

- Chứng minh định lý 2 bằng phản chứng:
  - Giả sử có T là cây khung nhỏ nhất, và e không thuộc T
  - Xét chu trình tạo bởi T + e
  - Tồn tại ít nhất 1 cạnh e' nối từ U sang V -> e' >= e
  - Từ định lí 1 suy ra cạnh e' với e phải có trọng số = nhau

  -> có thể thay e' = e để được cây khung nhỏ nhất có trọng số k[illegible] chứa e



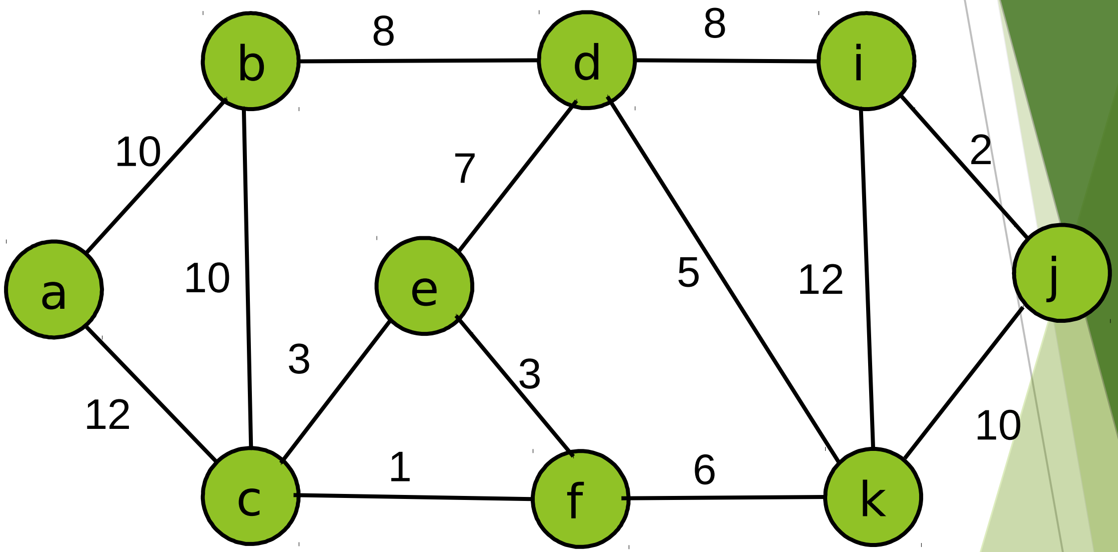
- Ai hiểu được chứng minh trên thì về tìm hiểu lý thuyết đồ thị (Graph Theory)

Cây khung nhỏ nhất

- Thuật toán Kruskal dựa vào định lý 1 và 2:
  - Lần lượt chọn các cạnh có trọng số từ thấp đến cao sao cho cạnh mới thêm vào không tạo thành chu trình
  - Dừng lại khi chọn đủ N-1 cạnh
- Greedy
- Tất cả N-1 cạnh được chọn như trên đều phù hợp với định lý 2
  - How: Để chứng minh phù hợp với định lý 2 cần chọn ra tập cut U, V thỏa mãn
  - Bài tập về nhà
  - Nhập môn Graph Theory

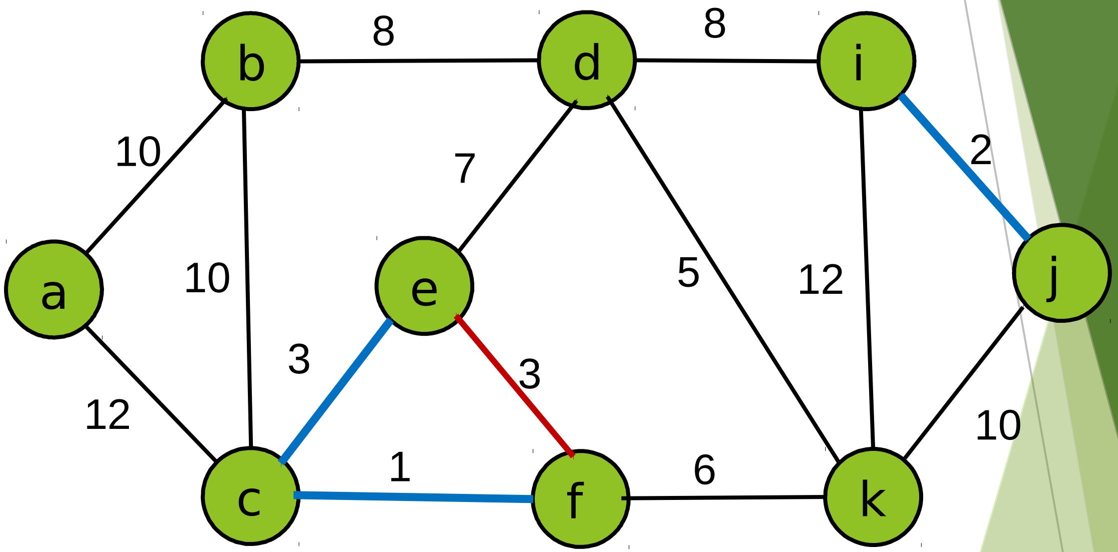
Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	f	1
i	j	2
c	e	3
e	f	3
d	k	5
f	k	6
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12



Step 1:

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	k	5
f	k	6
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12



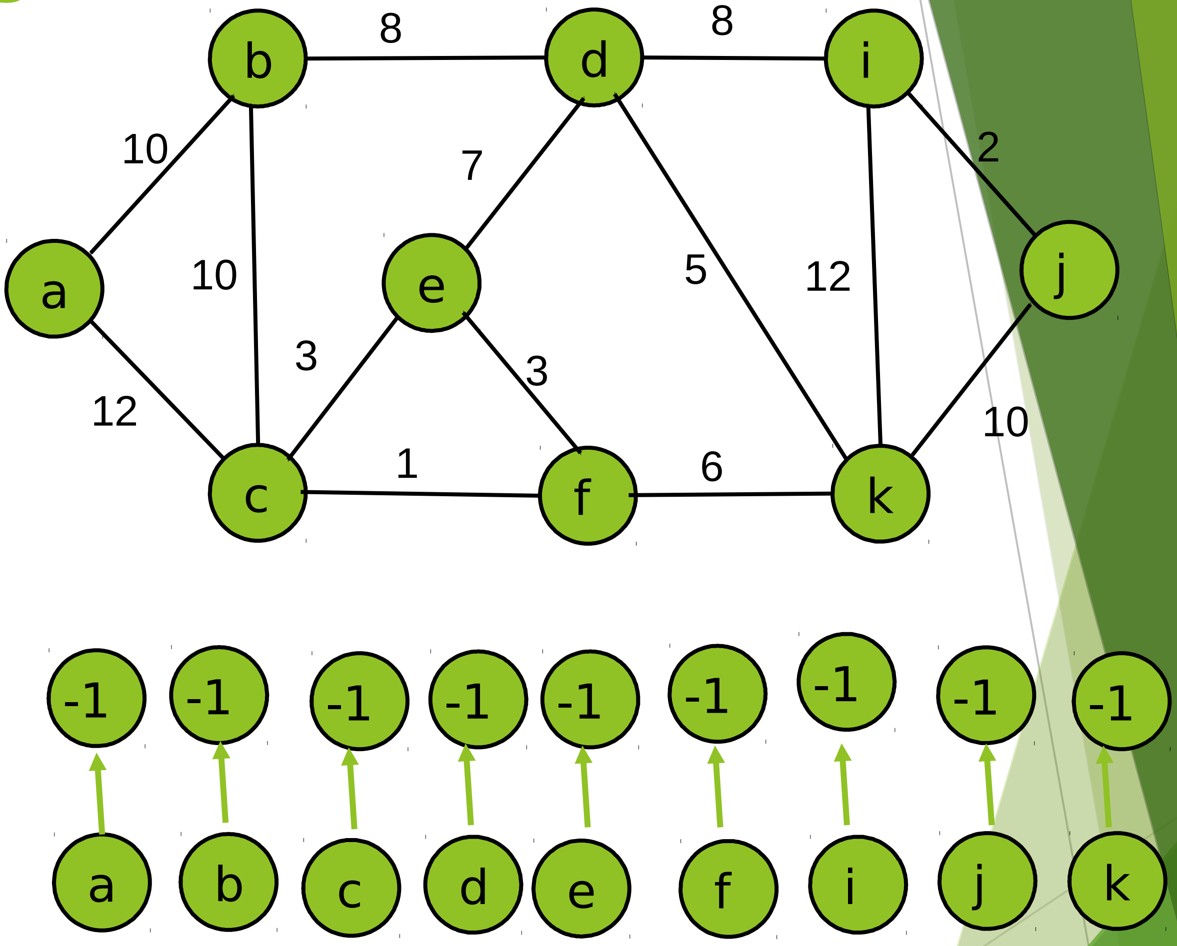
Bài toán con: cần kiểm tra nhanh 2 đỉnh u, v có thuộc 1 thành phần liên thông hay không

Bài toán con: Hợp và Tìm kiếm

- Union - Find
  - Union(u, v): hợp 2 thành phần chứa u và v
  - Find(u, v): kiểm tra xem u và v có thuộc cùng 1 thành phần liên thông hay không
- Cài đặt: dùng cấu trúc tree với biến phụ parent:
  - parent[u]: Đi ngược lên phía trên
  - Đi theo parent sẽ đi về gốc của cây
  - 2 đỉnh u, v cùng thuộc 1 thành phần nếu chung gốc
- Khởi tạo parent [u] = -1 với mọi u
- getroot(u): đi theo parent cho đến khi gặp nút có parent = -1
- Union(u, v): nối parent[getroot(u)] = getroot(v)

Cây khung nhỏ nhất

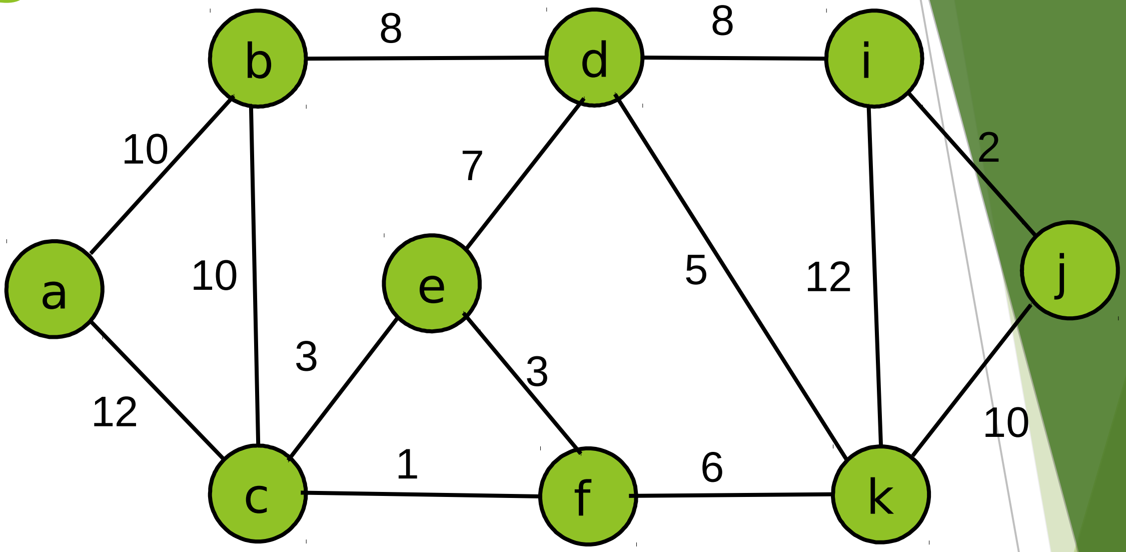
Đỉnh 1	Đỉnh 2	Trọng số
c	f	1
i	j	2
c	e	3
e	f	3
d	k	5
f	k	6
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12

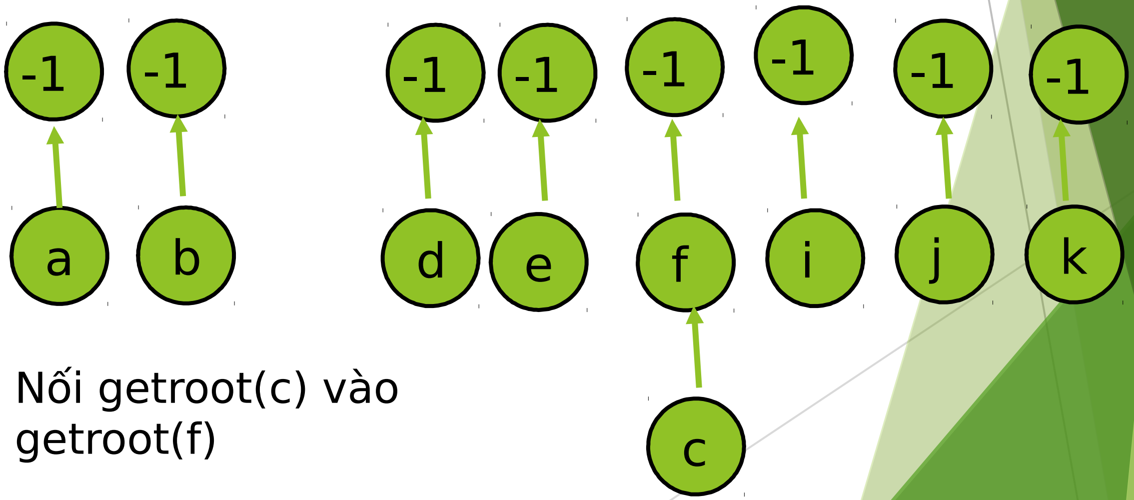


Các đỉnh đều đứng riêng rẽ

Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	j	2
c	e	3
e	f	3
d	k	5
f	k	6
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12

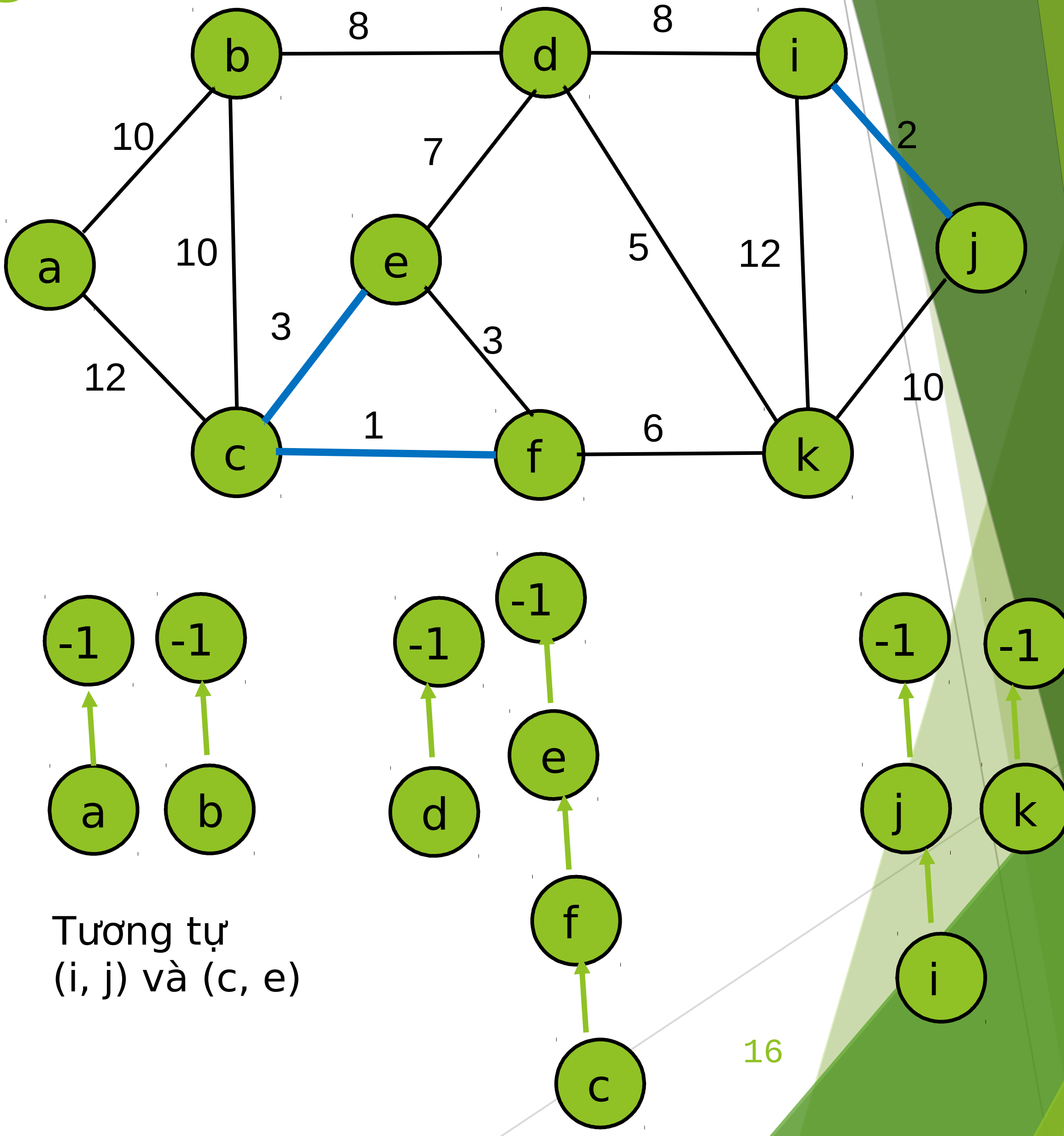




Nối getroot(c) vào getroot(f)

Cây khung nhỏ nhất

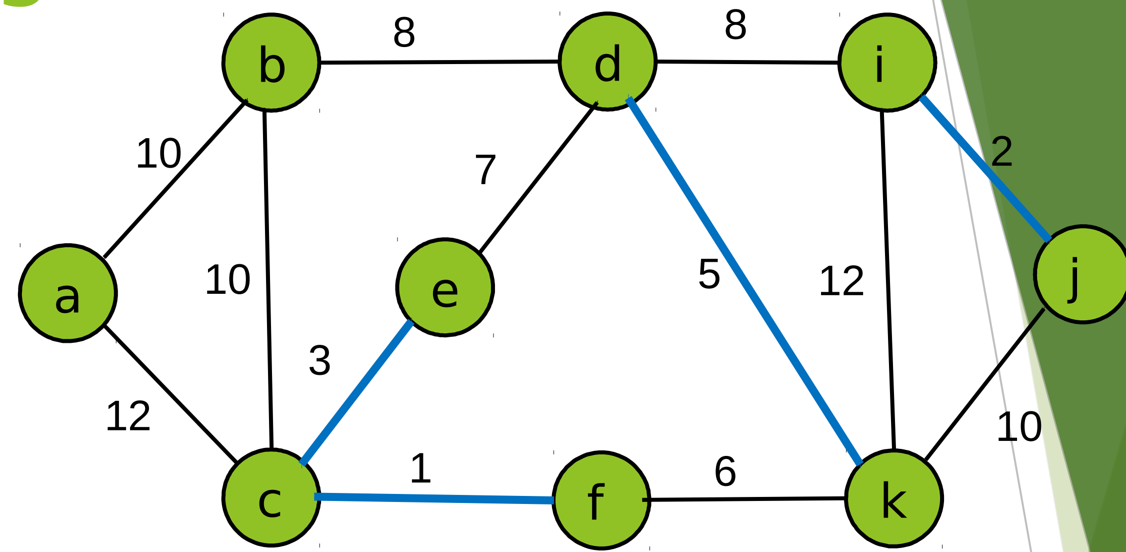
Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	k	5
f	k	6
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12

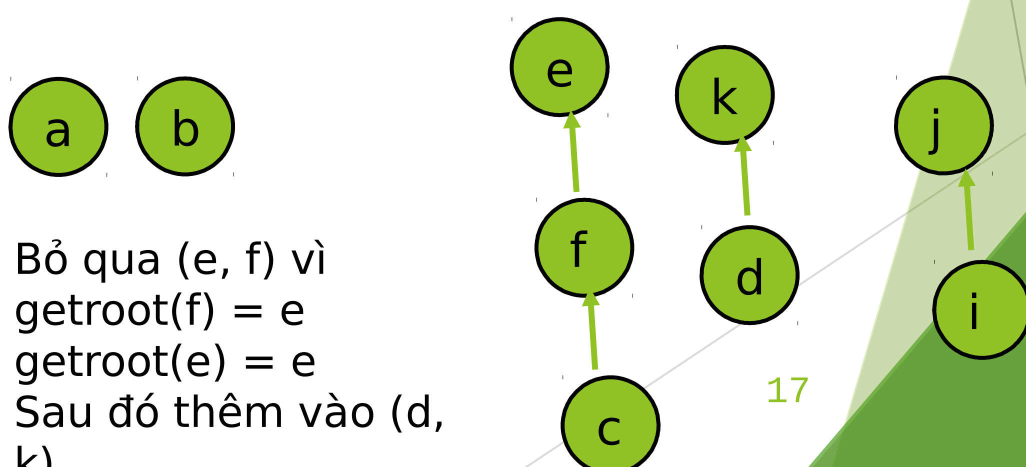


Tương tự
(i, j) và (c, e)

Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	**k**	**5**
f	k	6
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12

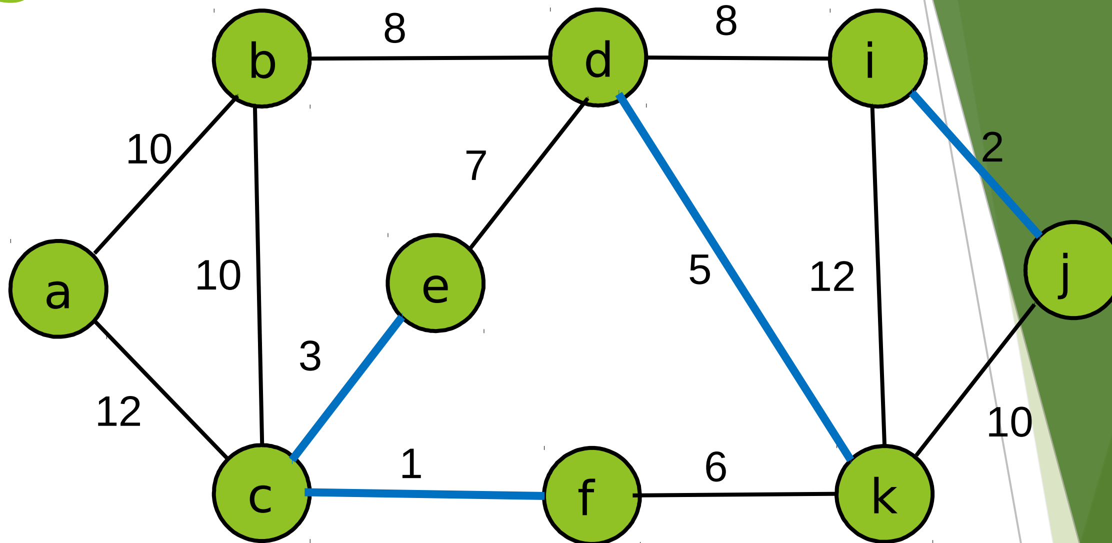


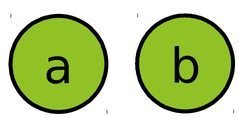


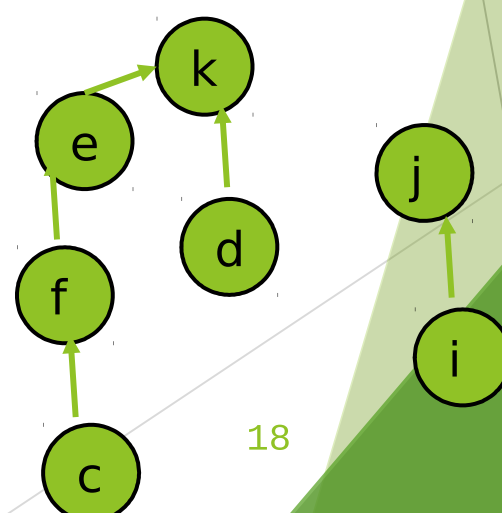
Bỏ qua (e, f) vì
getroot(f) = e
getroot(e) = e
Sau đó thêm vào (d, k)

Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	**k**	**5**
f	**k**	**6**
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12



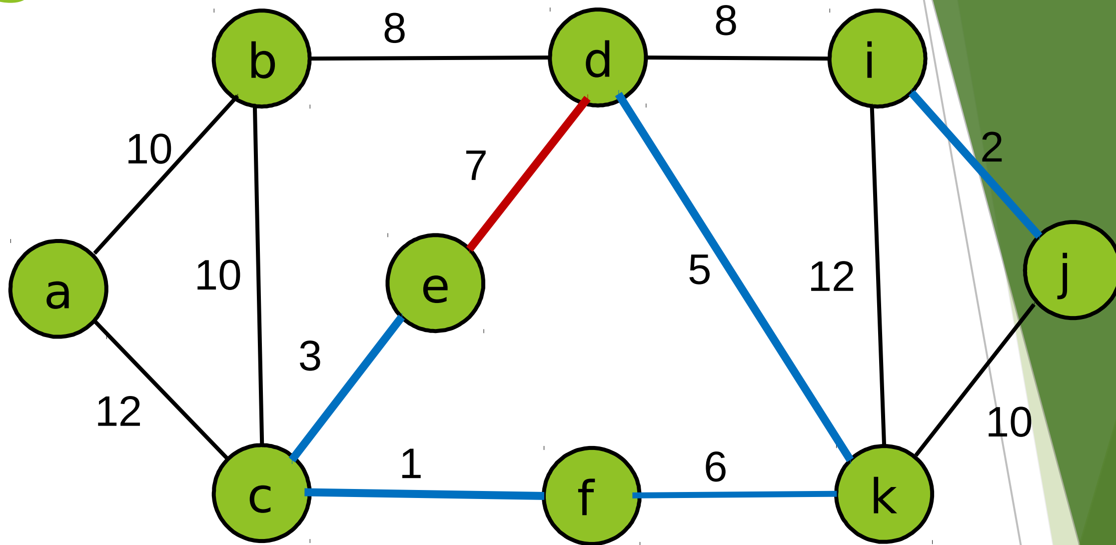


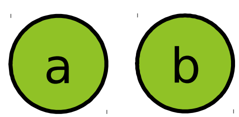


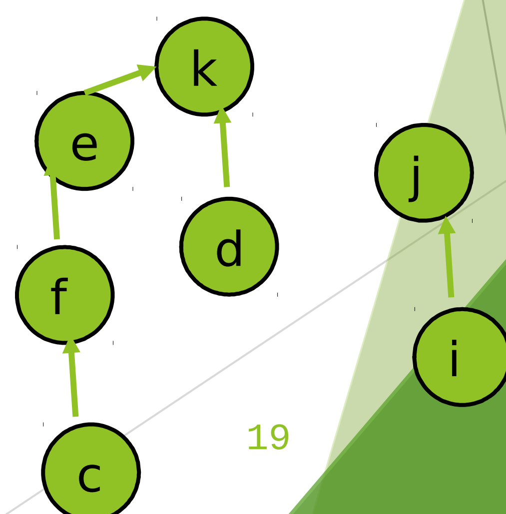
getroot(f) = e
getroot(k) = k
Thêm vào f, k

Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	**k**	**5**
f	**k**	**6**
d	e	7
b	d	8
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12



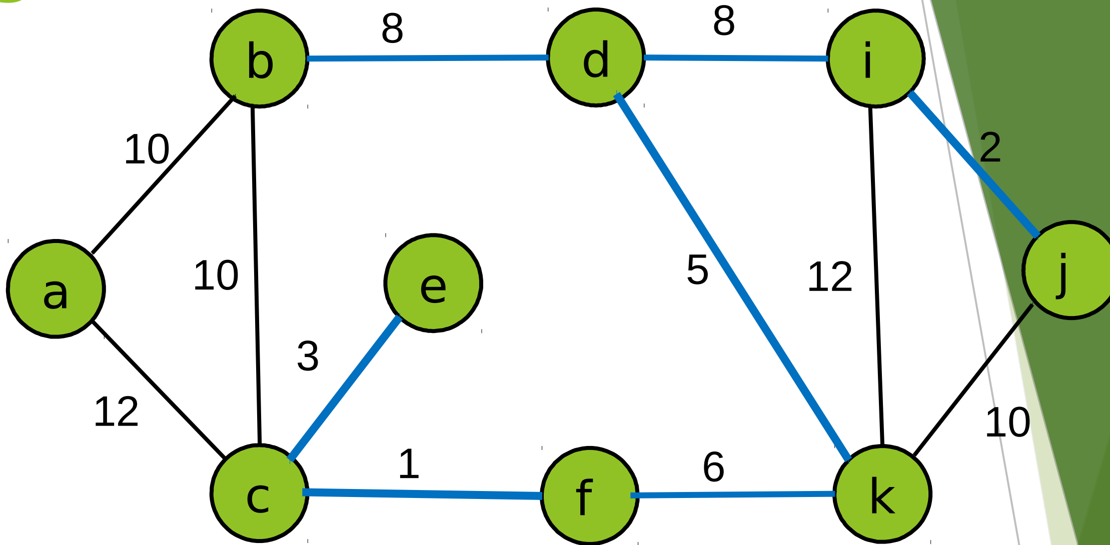


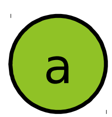


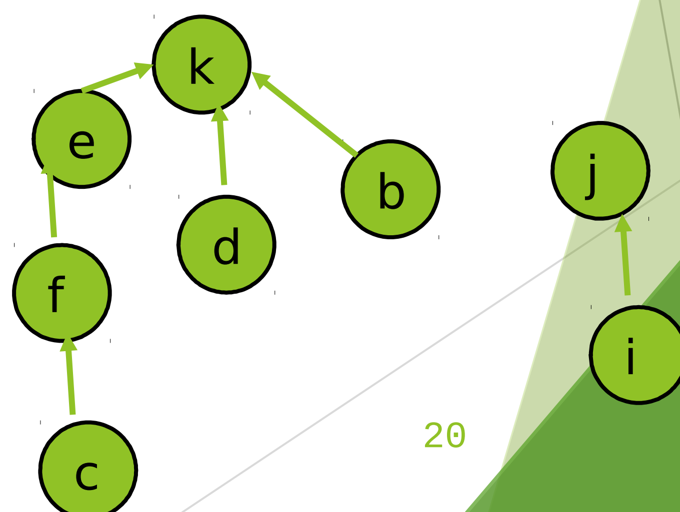
(d, e) có chung
gốc k -> bỏ qua

Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	**k**	**5**
f	**k**	**6**
d	e	7
b	**d**	**8**
d	i	8
a	b	10
j	k	10
a	c	12
i	k	12



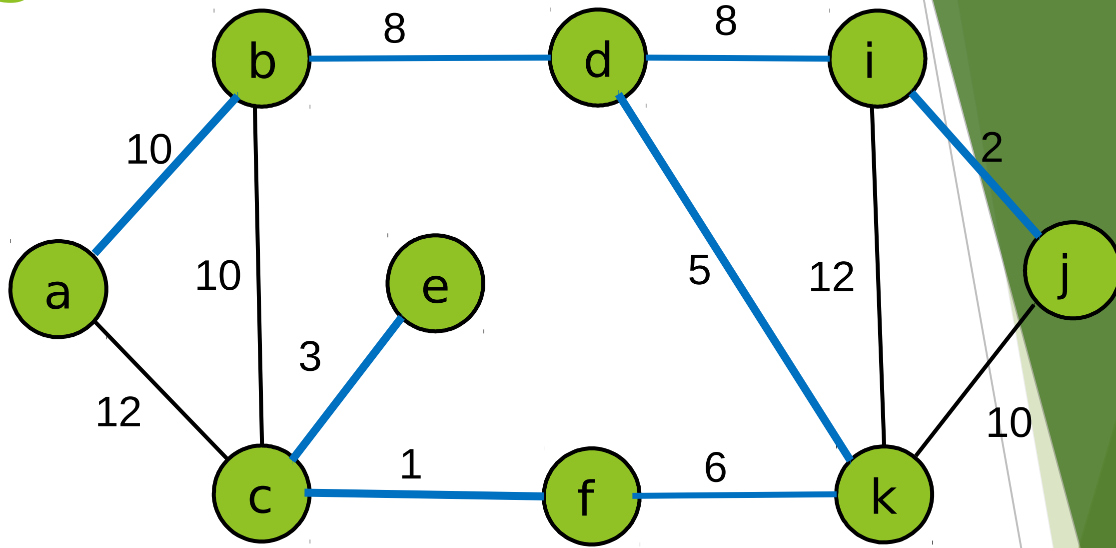




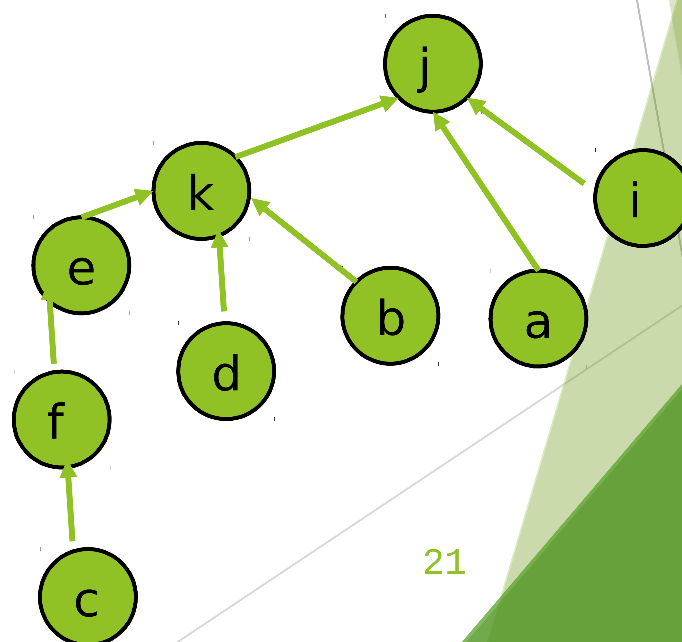
getroot(b) = b
getroot(d) = k
thêm vào (b, d)

Cây khung nhỏ nhất

Đỉnh 1	Đỉnh 2	Trọng số
c	**f**	**1**
i	**j**	**2**
c	**e**	**3**
e	f	3
d	**k**	**5**
f	**k**	**6**
d	e	7
b	**d**	**8**
d	**i**	**8**
a	**b**	**10**
j	k	10
a	c	12
i	k	12



getroot(d) = k
getroot(i) = j
thêm vào (d, i)
tương tự
thêm vào (a, b)
Đủ 8 cạnh
-> Dừng



Pseudocode

```
Algorithm KruskalMST(G)
   for each vertex V in G do
      parent[v] = -1

   let Q be a priority queue min heap
   Insert all edges into Q using their weights
as the key
   T ← Empty
   while T has fewer than n-1 edges do
      edge e = T.removeMin()
      Let u, v be the endpoints of e
      rootu = GetRoot(parent, u)
      rootv = GetRoot(parent, v)
      if rootu ≠ rootv then
         Add edge e to T
         parent[rootu] = rootv

   return T
```

Pseudocode

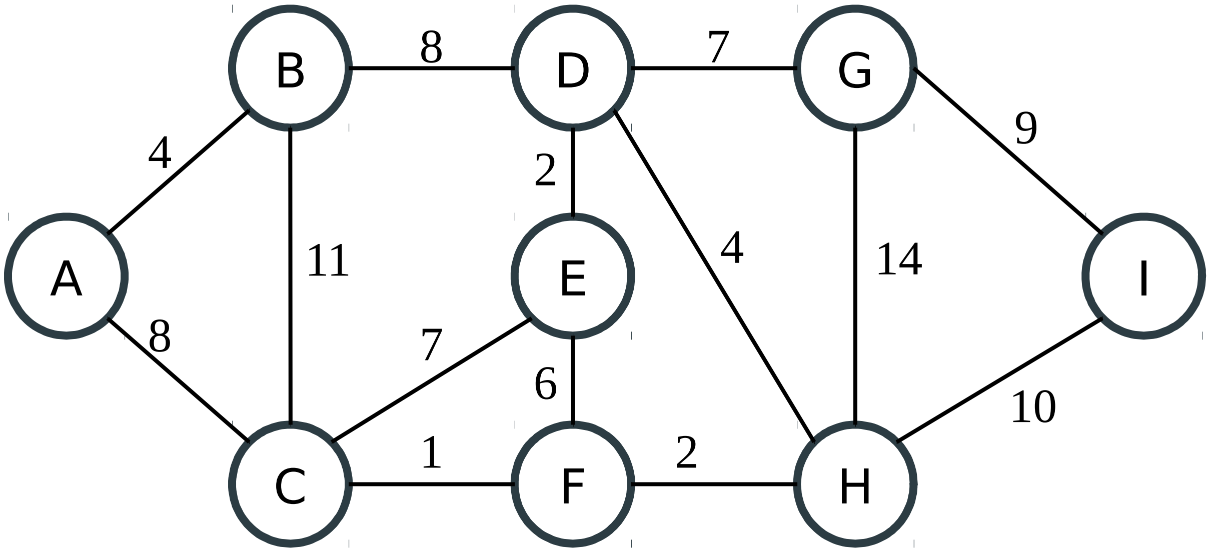
```
// Use recursion to follow the parent pointer to the root of the tree

// And compress the path

GetRoot(parent, u):

    if parent[u] == -1: return u

    v = GetRoot(parent, parent[u])

    parent[u] = v

    return v
```

Complexity

- Bước khởi tạo Queue: MlogM
- Vòng lặp chọn cạnh:
  - Tối đa M cạnh
  - Mỗi lần tìm getroot và update trong O(1)
- Độ phức tạp là O(MlogM)

Bài tập

- Tìm cây khung:



Thuật toán khác

- Thuật toán Prim:
  - Sửa lại thuật toán Dijkstra như sau:
    - Xuất phát từ cạnh bất kì
    - Hàm cập nhật khoảng cách: Thay là D[v] > cạnh(u, v)

Prim for MST

relaxation(u, v)

if d[v] > weight(u, v)

d[v] = weight(u, v)

prev[v] = u

Algorithm Prim ()

Bước khởi tạo: Chọn a là đỉnh bất kì

Gán K = {a} Gán U = V / {a}

d[a] = 0

d[u ∈ U] = weight[a, u] if exists, else = infinity

prev[u] = a, với mọi u

Bước tối ưu, lặp cho đến khi hết các đỉnh

Tìm u ∈ U với d[u] minimum

if ***u* ≠ *b*** then

K = K + {u} và U = U - {u}

for each edge (*u*, *v*) *do* ***relaxation*** *(u, v)*

else break // ***u == b***

Tracing step

path ← {***e***}

while (***e*** != ***s***) do

e ← prev[***e***]

path = [***e***] + path

Thuật toán khác

- Thuật toán Prim:
  - Sửa lại thuật toán Dijkstra như sau:
    - Xuất phát từ cạnh bất kì
    - Hàm cập nhật khoảng cách: Thay là D[v] > cạnh(u, v)

- Bài tập về nhà:
  - Chứng minh tính đúng đắn của thuật toán Prim (?)

Tổng kết

- Làm quen với các bài toán cơ bản của Graph
  - BFS:
    - Tìm thành phần liên thông
    - Tìm đường đi ít cạnh nhất
  - DFS:
    - Tìm thành phần liên thông
    - Topological sorting
  - Dijkstra
  - Kruskal

- Làm quen với lý thuyết đồ thị
- Làm quen với các cải tiến nhỏ để đảm bảo tốc độ chạy

Tổng kết

- Mai học thực hành bình thường
- Tuần tới nghỉ ôn tập:
  - nội dung: 10 slides lý thuyết + bài tập thực hành
- Thứ 5: 29/11 ôn tập
- Thứ 6: 30/11 kiểm tra giữa kì lần 2