CH08-320201

Algorithms and Data Structures

ADS

Lecture 22

Dr. Kinga Lipskoch

Spring 2019

Depth-First Search (DFS)

DFS Strategy:

First follow one path all the way to its end, before we step back to follow the next path ($u.d$ and $u.f$ are start/finish time for vertex processing)

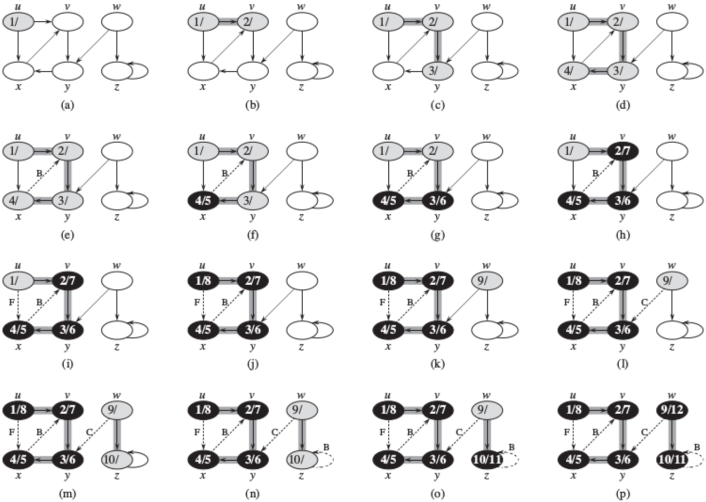
```
DFS(G)

for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

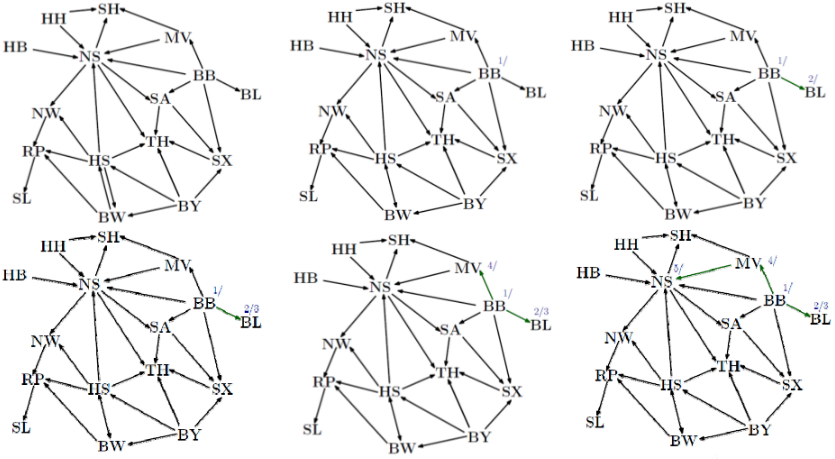
```
DFS-VISIT(G, u)

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

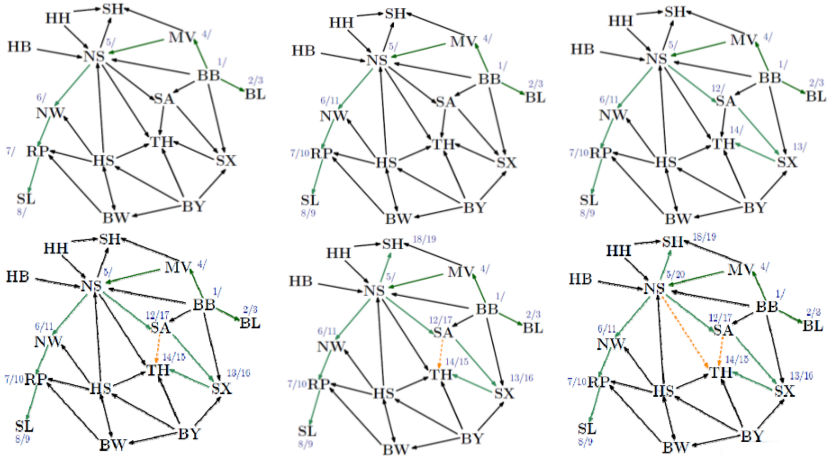
DFS Example



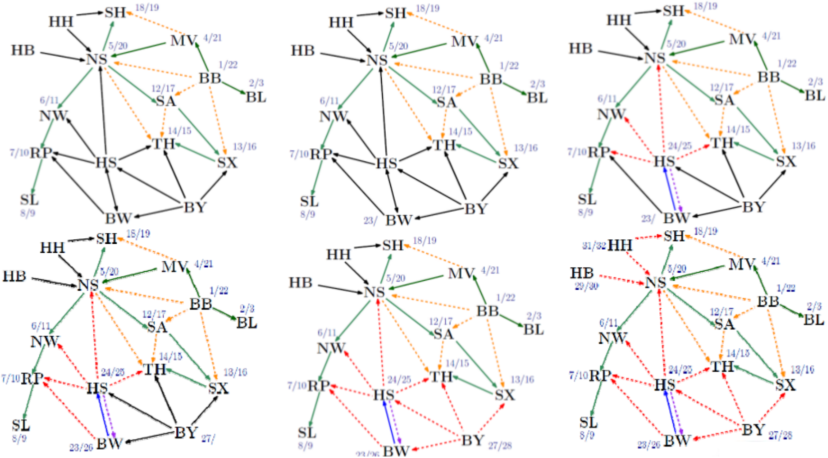
Another DFS Example (1)



Another DFS Example (2)



Another DFS Example (3)



DFS Analysis

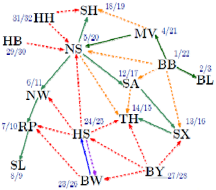
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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

Each vertex and each edge is processed once.
Hence, time complexity is $\Theta(|V| + |E|)$.

Edge Types

- Different edge types for (u, v):
  - Tree edges (solid): v is white.
  - Backward edges (purple): v is gray.
  - Forward edges (orange): v is black and $u.d < v.d$
  - Cross edges (red): v is black and $u.d > v.d$
- The tree edges form a forest.
- This is called the depth-first forest.
- In an undirected graph, we have no forward and cross edges.



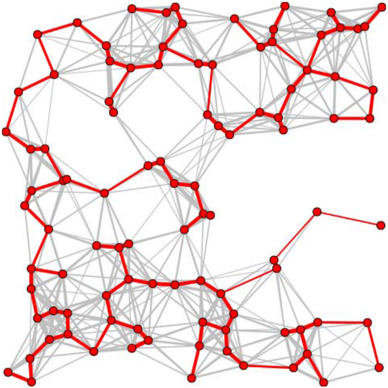
Minimum Spanning Tree: Problem

- Given a connected undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.
- Compute a minimum spanning tree (MST), i.e., a tree that connects all vertices with minimum weight

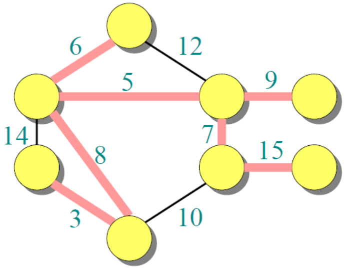
$$w(T) = \sum_{(u,v)\in T} w(u,v).$$

- Why of interest?
  One example would be a telecommunications company laying out cables to a neighborhood.

Example Spanning Tree

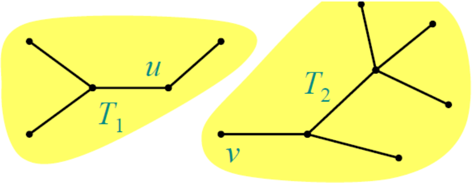


Example MST



Optimal Substructure

- Consider an MST T of graph G (other edges not shown).
- Remove any edge $(u, v) \in T$.
- Then, T is partioned into subtrees T_1 and T_2.



MST: Theorem

(a) Subtree T_1 is a MST of graph $G_1 = (V_1, E_1)$ with V_1 being the set of all vertices of T_1 and E_1 being the set of all edges $\in G$ that connect vertices $\in V_1$.

(b) Subtree T_2 is a MST of graph $G_2 = (V_2, E_2)$ with V_2 being the set of all vertices of T_2 and E_2 being the set of all edges $\in G$ that connect vertices $\in V_2$.

Proof (only (a), (b) is analogous):

(1) $w(T) = w(T_1) + w(T_2) + w(u, v)$

(2) Assume S_1 was a MST for G_1 with lower weight than T_1.

(3) Then, $S = S_1 \cup T_2 \cup \{(u, v)\}$ would be an MST for G with lower weight than T.

(4) Contradiction.

Greedy Choice Property (1)

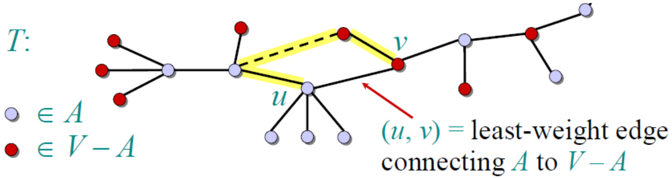
Theorem:

- Let T be the MST of graph $G = (V, E)$ and let $A \subset V$.
- Let $(u, v) \in E$ be the edge with least weight connecting A to $V \setminus A$.
- Then, $(u, v) \in T$.

Greedy Choice Property (2)

Proof:

- Suppose (u, v) is not part of T.
- Then, consider the path from u to v within T.
- Replace the weight of the first edge on this path that connects a vertex in A to a vertex in $V \setminus A$ with the weight of (u, v).
- This results in a spanning tree with smaller weight. Contradiction.



Prim's Algorithm

Idea:

- Develop a greedy algorithm that iteratively increases A and, consequently, decreases $V \setminus A$.
- Maintain $V \setminus A$ as a min-priority queue Q (min-priority queue analogous to max-priority queue).
- Key each vertex in Q with the weight of the least weight edge connecting it to a vertex in A (if no such edge exists, the weight shall be infinity).
- Then, always add the vertex of $V \setminus A$ with minimal key to A.

Min-Priority Queues

Definition (recall):
A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key.

Definition (implementation as min-heap):
A min-priority queue is a priority queue that supports the following operations:

- `Minimum(S)`: return element from S with smallest key. [$O(1)$]
- `Extract-Min(S)`: remove and return element from S with smallest key. [$O(\lg n)$]
- `Decrease-Key(S, x, k)`: decrease the value of the key of element x to k, where k is assumed to be smaller or equal than the current key. [$O(\lg n)$]
- `Insert(S, x)`: add element x to set S. [$O(\lg n)$]