Final Exam Study Guide

Introduction to Algorithms

Fall 2014

Chapter 22: Elementary Graph Algorithms

- **Graph Representation:**
  - Adjacency lists.
  - Adjacency matrix.
- **Breadth-first search Algorithm:**

```
BFS(V, E, s)
for each u ∈ V − {s}
     do d[u] ← ∞
d[s] ← 0
Q ← Ø
ENQUEUE(Q, s)
while Q ≠ Ø
     do u ← DEQUEUE(Q)
          for each v ∈ Adj[u]
               do if d[v] = ∞
                    then d[v] ← d[u] + 1
                         ENQUEUE(Q, v)
```

  - Input: Graph G= (V, E), either directed or undirected, and a source vertex sϵ V.
  - Output: d[v]= distance (smallest # of edges) from s to v, for all vϵ V. also Π[v] = u such that (u , v) is last edge on the shortest path s ⇝v
  - Running time = O (V+E):
    - O (V) because every vertex enqueued at most once.
    - O€ because every vertex dequeued at most once and will examine (u,v) only when u is dequeued. Therefore, every edge examined at most once if directed, at most twice if undirected.

- **Depth-first search**

```
DFS(V, E)
for each u ∈ V
     do color[u] ← WHITE
time ← 0
for each u ∈ V
     do if color[u] = WHITE
           then DFS-VISIT(u)

DFS-VISIT(u)
color[u] ← GRAY           ▷ discover u
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]       ▷ explore (u, v)
     do if color[v] = WHITE
           then DFS-VISIT(v)
color[u] ← BLACK
time ← time + 1
f[u] ← time               ▷ finish u
```

- **Input:** $G = (V, E)$, directed or undirected. No source vertex given!
- **Output:** 2 ***timestamps*** on each vertex:
  - $d[v]$ = ***discovery time.***
  - $f[v]$ = ***finishing time.***

Will methodically explore *every* edge. Start over from different vertices as necessary.
As soon as we discover a vertex, explore from it. Unlike BFS, which puts a vertex on a queue so that we explore from it later. As DFS progresses, every vertex has a ***color***:

- WHITE = undiscovered
- GRAY = discovered, but not finished (not done exploring from it)
- BLACK = finished (have found everything reachable from it)

Discovery and finish times:

- Unique integers from 1 to $2|V|$.
- For all v, $d[v] < f[v]$.

In other words, $1 \le d[v] < f[v] \le 2|V|$.

Classification of edges

- ***Tree edge:*** in the depth-first forest. Found by exploring (u, v).
- ***Back edge:*** (u, v), where u is a descendant of v.
- ***Forward edge:*** (u, v), where v is a descendant of u, but not a tree edge.
- ***Cross edge:*** any other edge. Can go between vertices in same depth-first tree or in different depth-first trees.

Time: $\Theta(V + E)$.

Chapter 23: Minimum Spanning Trees

- **Kruskal's Algorithm :**

- $G = (V, E)$ is a connected, **undirected,** weighted graph. $w: E \rightarrow \mathbf{R}$.

```
KRUSKAL(V, E, w)
A ← Ø
for each vertex v ∈ V
     do MAKE-SET(v)
sort E into nondecreasing order by weight w
for each (u, v) taken from the sorted list
     do if FIND-SET(u) ≠ FIND-SET(v)
           then A ← A ∪ {(u, v)}
                UNION(u, v)
return A
```

Running time: O (E log V)

- **Prim's Algorithm :**

- Builds one tree, so *A* is always a tree.
- Starts from an arbitrary "root" *r.*
- At each step, find a light edge crossing cut $(V_A, V - V_A)$, where V_A = vertices that *A* is incident on. Add this edge to *A*.

```
PRIM(V, E, w, r)
Q ← Ø
for each u ∈ V
     do key[u] ← ∞
        π[u] ← NIL
        INSERT(Q, u)
DECREASE-KEY(Q, r, 0)     ▷ key[r] ← 0
while Q ≠ Ø
     do u ← EXTRACT-MIN(Q)
        for each v ∈ Adj[u]
            do if v ∈ Q and w(u, v) < key[v]
                  then π[v] ← u
                       DECREASE-KEY(Q, v, w(u, v))
```

Running time O(E log V).

Chapter 24: Single-Source shortest Paths:

- **Shortest paths :**

Input:

- Directed graph $G = (V, E)$
- Weight function $w : E \to \mathbf{R}$

Weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$= \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$=$ sum of edge weights on path p .

Shortest-path weight u to v:

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there exists a path } u \leadsto v , \\ \infty & \text{otherwise .} \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u, v)$.

Cycles:

Shortest paths can.t contain cycles:

- Already ruled out negative-weight cycles.
- Positive-weight ⇒we can get a shorter path by omitting the cycle.
- Zero-weight: no reason to use them ⇒ assume that our solutions won.t use them.

Initialization:

All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

```
INIT-SINGLE-SOURCE(V, s)
for each v ∈ V
    do d[v] ← ∞
       π[v] ← NIL
d[s] ← 0
```

Relaxing an edge *(u, v)*

```
RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
  then d[v] ← d[u] + w(u, v)
       π[v] ← u
```

- **Bellman-Ford Algorithm :**
  - Allows negative-weight edges.
  - Computes $d[v]$ and $\pi[v]$ for all $v \in V$.
  - Returns TRUE if no negative-weight cycles reachable from s, FALSE otherwise.

```
BELLMAN-FORD(V, E, w, s)
INIT-SINGLE-SOURCE(V, s)
for i ← 1 to |V| − 1
    do for each edge (u, v) ∈ E
            do RELAX(u, v, w)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
          then return FALSE
return TRUE
```

Core: The first **for** loop relaxes all edges $|V| - 1$ times.

Time: $\Theta(VE)$.

- **Dijkstra's Algorithm :**
  - No negative-weight *edges*.
  - Essentially a weighted version of breadth-first search.
  - Instead of a FIFO queue, uses a priority queue.
  - Keys are shortest-path weights ($d[v]$).
  - Have two sets of vertices:
    - S = vertices whose final shortest-path weights are determined,
    - Q = priority queue = $V - S$.

```
DIJKSTRA(V, E, w, s)
INIT-SINGLE-SOURCE(V, s)
S ← Ø
Q ← V          ▷ i.e., insert all vertices into Q
while Q ≠ Ø
    do u ← EXTRACT-MIN(Q)
       S ← S ∪ {u}
       for each vertex v ∈ Adj[u]
           do RELAX(u, v, w)
```

Running time: O(E lg V) , if binary heap