CS483 Design and Analysis of Algorithms

Lectures 10-11 Paths in Graphs

Instructor: Fei Li

lifei@cs.gmu.edu with subject: CS483

Office hours: STII, Room 443, Friday 4:00pm - 6:00pm or by appointments

Course web-site:

http://www.cs.gmu.edu/∼lifei/teaching/cs483_fall08/

Figures unclaimed are from books "Algorithms" and "Introduction to Algorithms"

Paths in Graphs

1. Breath-First Search
2. Dijkstra's Algorithm
3. Shortest Paths in the Presence of Negative Edges
4. Shortest Paths in Directed Acyclic Graphs

Depth-First Search

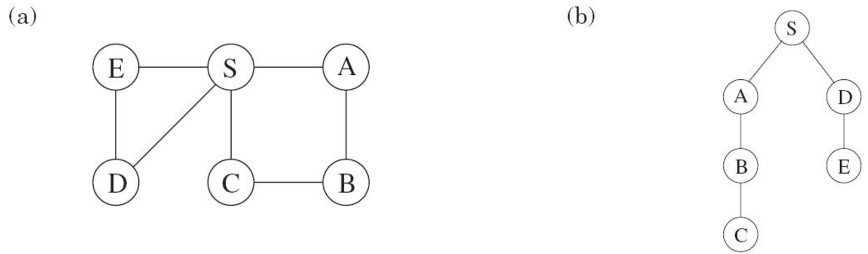
Remark

Depth-first search readily identifies all the vertices of a graph that can be reached from a designated starting point. It also finds explicit paths to these vertices, summarized in its search tree. However, these paths might not be the most economical ones possible.

Problem

Is there a way to find the shortest path *in graphs?*

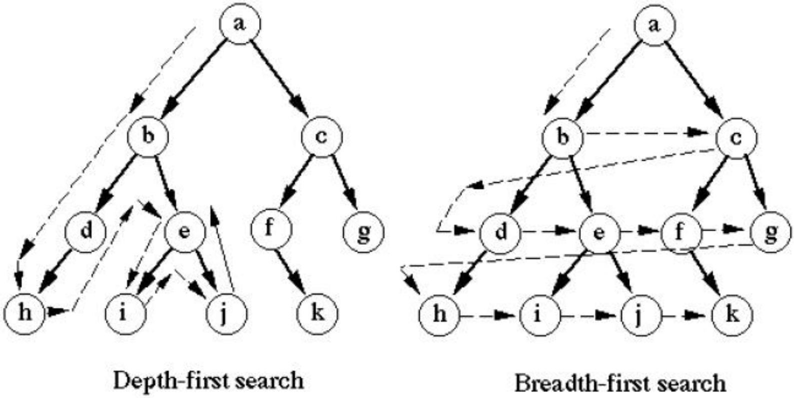
Figure 4.1 (a) A simple graph and (b) its depth-first search tree.



Distances

Definition

The *distance* between two nodes is the length of the shortest path between them



http://www.cse.unsw.edu.au/ billw/Justsearch1.gif

Breath-First Search

1. Initially, the queue Q consists only of s, the one node at distance 0.
2. For each subsequent distance $d = 1, 2, \ldots$, there is a point in time at which Q contains all the nodes at distance d and nothing else.
3. As these nodes are processed (ejected off the front of the queue), their as-yet-unseen neighbors are injected into the end of the queue.

Proof.

For each $d = 0, 1, 2, \ldots$, there is a moment at which

1. all nodes at distance $\leq d$ from s have their distances correctly set;
2. all other nodes have their distances set to ∞; and
3. the queue contains exactly the nodes at distance d.

□

Theorem

The overall running time of this algorithm is $O(|V| + |E|)$.
Each vertex is put on the queue exactly once, when it is first encountered, so there are $2 \cdot |V|$ queue operations.
Over the course of execution, the innermost loop looks at each edge once (in directed graph) or twice (in undirected graphs), and therefore takes $O(|E|)$ time.

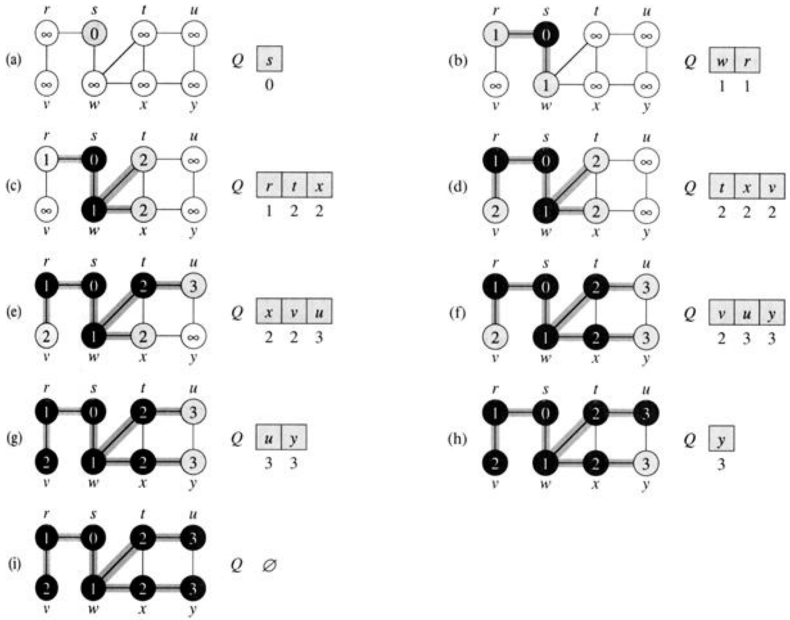
Breath-First Search

```
procedure bfs(G, s)
Input:  Graph G = (V, E), directed or undirected; vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set
        to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞

dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
   u = eject(Q)
   for all edges (u, v) ∈ E:
      if dist(v) = ∞:
         inject(Q, v)
         dist(v) = dist(u) + 1
```

Breath-First Search



Analysis of BFS

Theorem

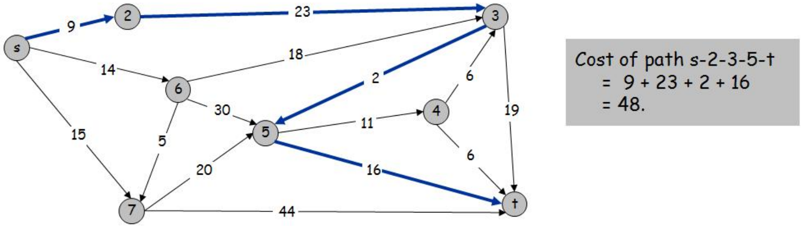
BFS *runs in* $O(m+n)$ *time if the graph is given by its adjacency representation,* n *is the number of nodes and* m *is the number of edges*

Proof.

When we consider node u, there are $deg(u)$ incident edges (u, v). Thus, the total time processing edges is $\sum_{u \in V} deg(u) = 2 \cdot m$ □

Dijkstra's Algorithm

Annotate every edge $e \in E$ with a *length* l_e. If $e = (u, v)$, let $l_e = l(u, v) = l_{uv}$
Input: Graph $G = (V, E)$ whose edge lengths l_e are *positive integers*
Output: The shortest path from s to t



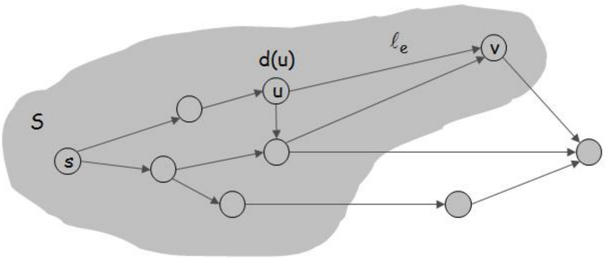
from Wayne's slides on "Algorithm Design"

Dijkstra's Algorithm

1. Maintain a set of explored nodes S for which we have determined the shortest path distance $d(u)$ from s to u
2. Initialize $S = \{s\}$, $d(s) = 0$
3. Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e=(u,v), u \in S} d(u) + l_e$$

4. Add v to S, and set $d(v) = \pi(v)$



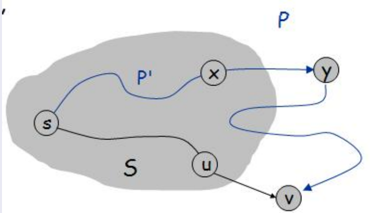
from Wayne's slides on "Algorithm Design"

Dijkstra's Algorithm

Theorem

Dijkstra's algorithm finds the shortest path from s to any node v: $d(v)$ is the length of the shortest $s \rightsquigarrow v$ path

Proof.



□

from Wayne's slides on "Algorithm Design"

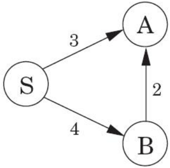
Theorem

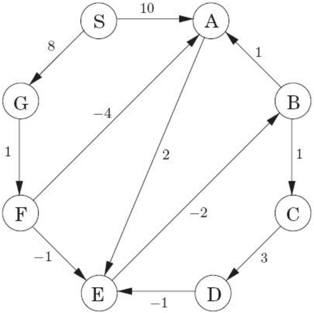
The overall running time of Dijkstra's algorithm is $O((|V| + |E|) \cdot \log |V|)$

Shortest Paths in the Presence of Negative Edges

Simply update *all* the edges, $|V| - 1$ times

Dijkstra's algorithm will not work if there are negative edges





	Iteration							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Shortest Paths in Directed Acyclic Graphs

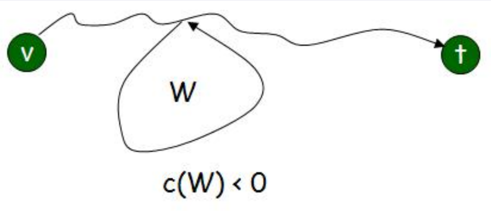
Definition

$OPT(i, v) :=$ length of shortest $v \rightsquigarrow t$ path P using at most i edges

Lemma

If $OPT(n, v) = OPT(n-1, v)$ for all v, then no negative cycles

Proof.



□

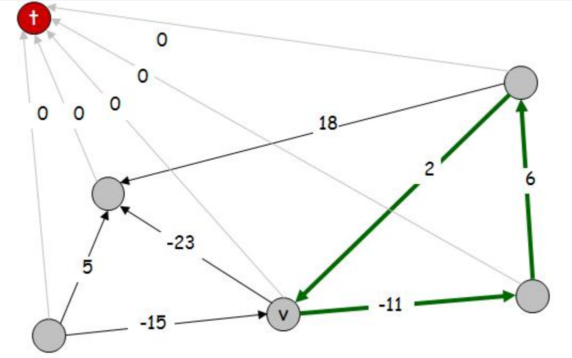
from Wayne's slides on "Algorithm Design"

Detecting Negative Cycles

Theorem

Negative cycles can be detected in time $O(m \cdot n)$

Proof.



□

from Wayne's slides on "Algorithm Design"

Shortest Paths in Directed Acyclic Graphs

Figure 4.15 A single-source shortest-path algorithm for directed acyclic graphs.

```
procedure dag-shortest-paths(G, l, s)
Input:  Dag G = (V, E);
        edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set
        to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
Linearize G
for each u ∈ V, in linearized order:
   for all edges (u, v) ∈ E:
      update(u, v)
```

Demo