Advanced Analysis of Algorithms

Dr. Qaiser Abbas
Department of Computer Science & IT,
University of Sargodha, Sargodha, 40100, Pakistan
qaiser.abbas@uos.edu.pk

Shortest Path Problem

- In a ***shortest-path problem***, we are given a weighted, directed graph G = (V,E) with weight function w: E→R mapping edges to real-valued weights. The ***weight*** w(p) of **path** p = (v_0,v_1,…v_k) is the sum of the weights of its constituent edges:

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

- We define the **shortest-path weight** δ(u,v) from **u** to **v** by

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

Shortest Path Problem

- **Variants**
- In this Lecture, we shall focus on the ***single-source shortest-paths problem***: given a graph G=(V,E), we want to find a shortest path from a given ***source*** vertex s ε V to each vertex v ε V . The algorithm for the single-source problem can solve many other problems, including the following variants.
  - **Single-destination shortest-paths problem:** Find a shortest path to a given ***destination*** vertex t from each vertex v. By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.
  - **Single-pair shortest-path problem:** Find a shortest path from u to v for given vertices u and v. If we solve the single-source problem with source vertex u, we solve this problem also.
  - **All-pairs shortest-paths problem:** Find a shortest path from u to v for every pair of vertices u and v. Although we can solve this problem by running a single- source algorithm once from each vertex, we usually can solve it faster. (see Chapter 25).

Shortest Path Problem

- **Optimal substructure of a shortest path**
  - Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it. (The Edmonds-Karp maximum-flow algorithm in Chapter 26)
  - Recall that optimal substructure is one of the key indicators that dynamic programming (Chapter 15) and the greedy method (Chapter 16) might apply.
  - Dijkstra’s algorithm, which we shall see next, is a greedy algorithm, and the Floyd- Warshall algorithm, which finds shortest paths between all pairs of vertices (will see next), is a dynamic-programming algorithm.

Shortest Path Problem

- Negative Weight Edges:

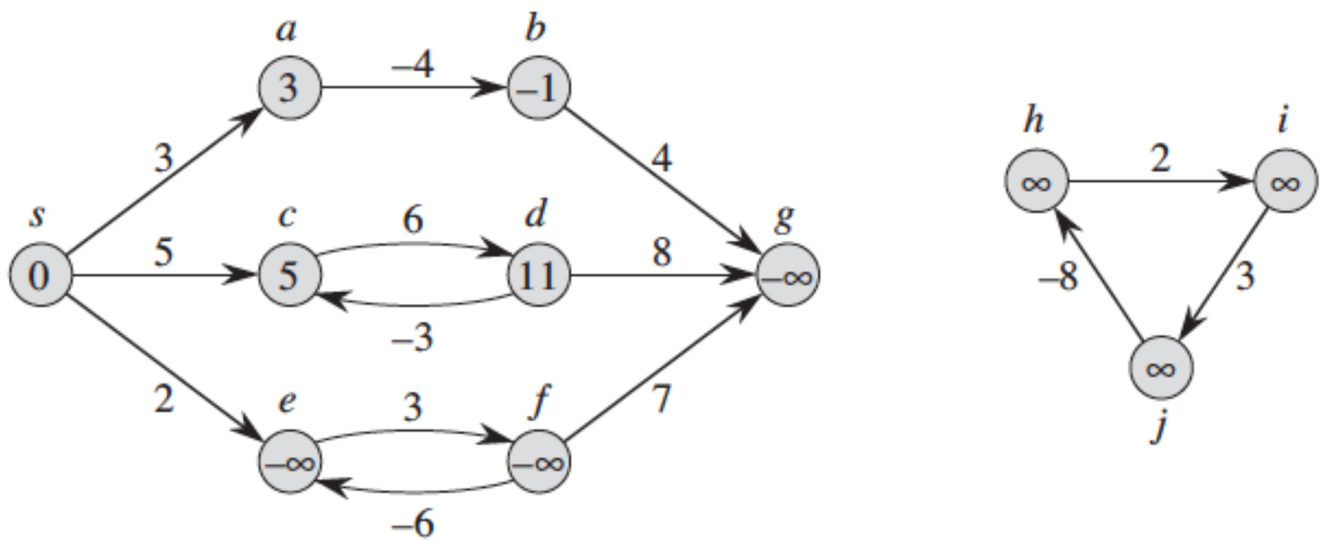


Figure 24.1 Negative edge weights in a directed graph. The shortest-path weight from source s appears within each vertex. Because vertices e and f form a negative-weight cycle reachable from s, they have shortest-path weights of $-\infty$. Because vertex g is reachable from a vertex whose shortest-path weight is $-\infty$, it, too, has a shortest-path weight of $-\infty$. Vertices such as h, i, and j are not reachable from s, and so their shortest-path weights are ∞, even though they lie on a negative-weight cycle.

Shortest Path Problem

- Cycles:
  - when we are finding shortest paths, they have no cycles, i.e., they are simple paths. Since any acyclic path in a graph G = (V,E) contains at most |V| distinct vertices, it also contains at most |V|- 1 edges.

Shortest Path Problem

- **Representing shortest paths**

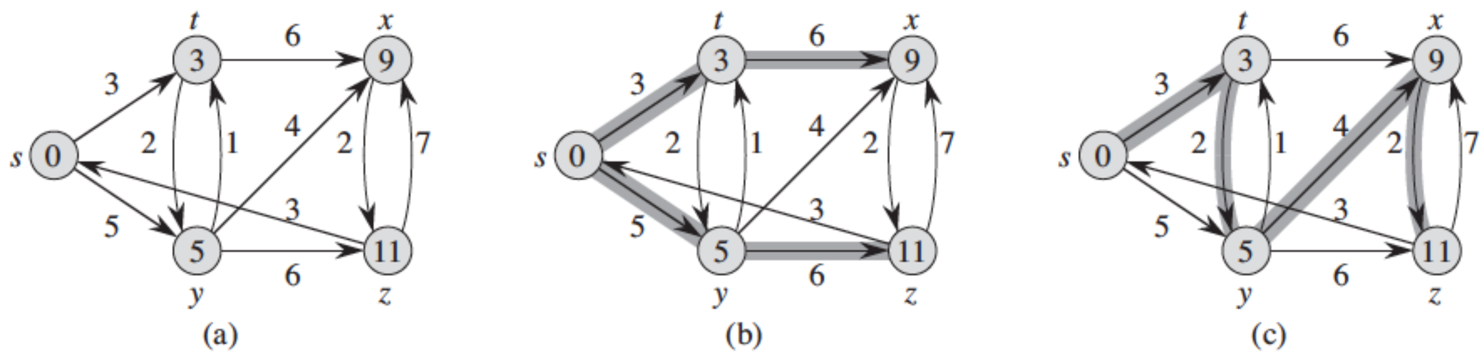


Figure 24.2 (a) A weighted, directed graph with shortest-path weights from source *s*. (b) The shaded edges form a shortest-paths tree rooted at the source *s*. (c) Another shortest-paths tree with the same root.

- Shortest paths are not necessarily unique, and neither are shortest-paths trees. For example, Figure 24.2 shows a weighted, directed graph and two shortest-paths trees with the same root.

Shortest Path Problem

- **Initialization**
  - For each vertex v ε V , we maintain an attribute v.d, which is an upper bound on the weight of a shortest path from source s to v.
  - We call v.d a ***shortest-path estimate***. We initialize the shortest-path estimates and predecessors by the following O(V)-time procedure:

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

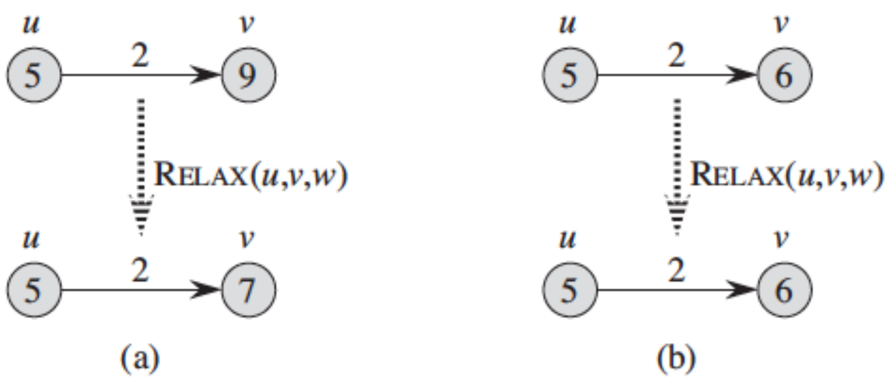
Shortest Path Problem

- **Relaxation**
  - The process of ***relaxing*** an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating v.d and v.π. A relaxation step may decrease the value of the shortest-path estimate v.d and update v's predecessor attribute v.π.
  - The following code performs a relaxation step on edge (u,v) in O(1) time:

RELAX(u, v, w)

if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$

Shortest Path Problem

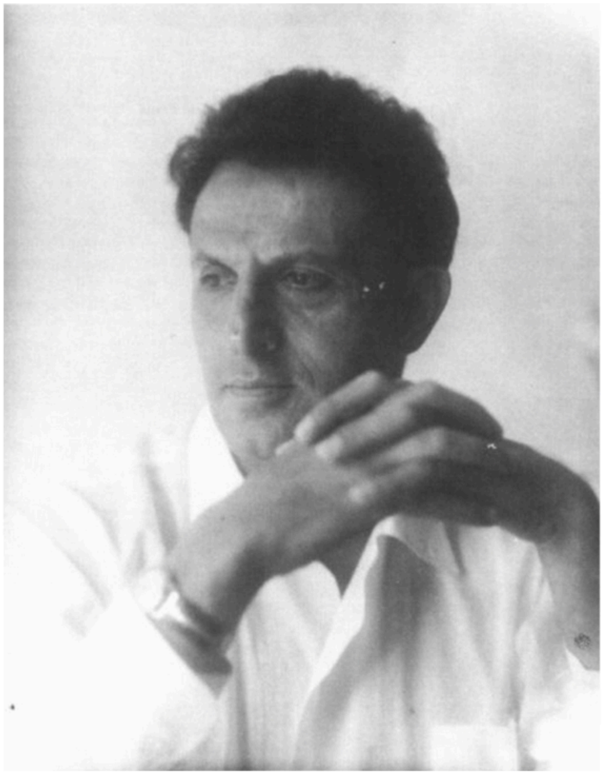


- Dijkstra's algorithm and the shortest-paths algorithm for directed acyclic graphs relax each edge exactly once. The Bellman-Ford algorithm relaxes each edge |V|-1 times.

The Bellman-Ford algorithm

- The ***Bellman-Ford algorithm*** solves the single-source shortest-paths problem in which edge weights may be negative.
- Given a weighted, directed graph G = (V,E) with source s and weight function w: E→R, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.
- If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.

Bellman & Ford



Richard E. Bellman
(1920–1984)
IEEE Medal of Honor, 1979

http://www.amazon.com/Bellman-Continuum-Collection-Works-Richard/dp/9971500906

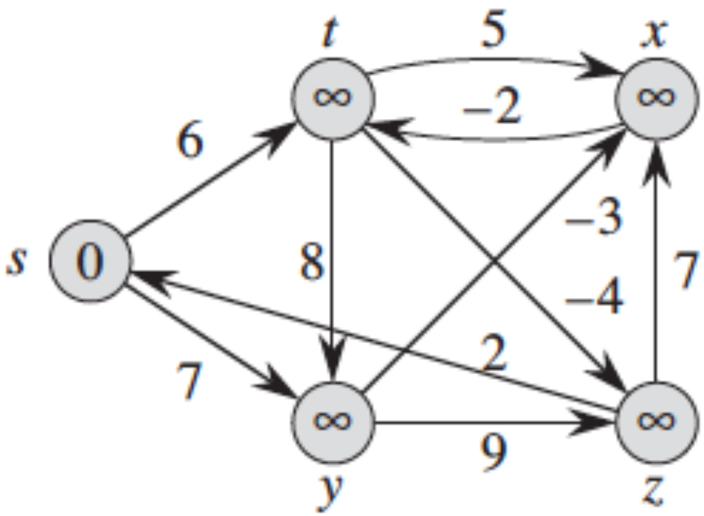
Lester R. Ford, Jr.
(1927–)
president of MAA, 1947–48

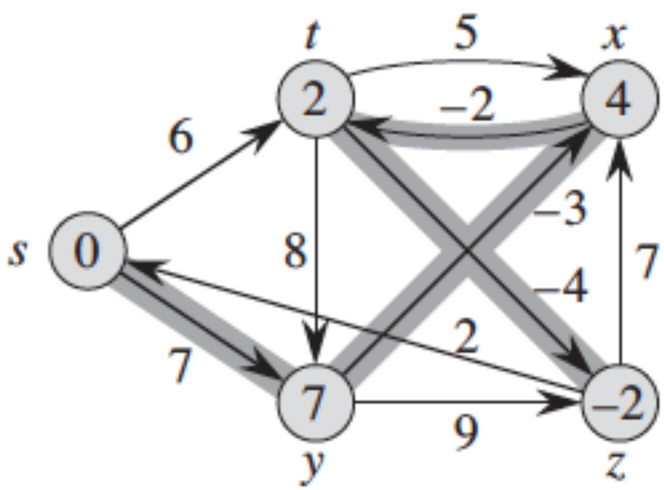
http://www.maa.org/aboutmaa/maaapresidents.html

The Bellman-Ford algorithm

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

The Bellman-Ford algorithm

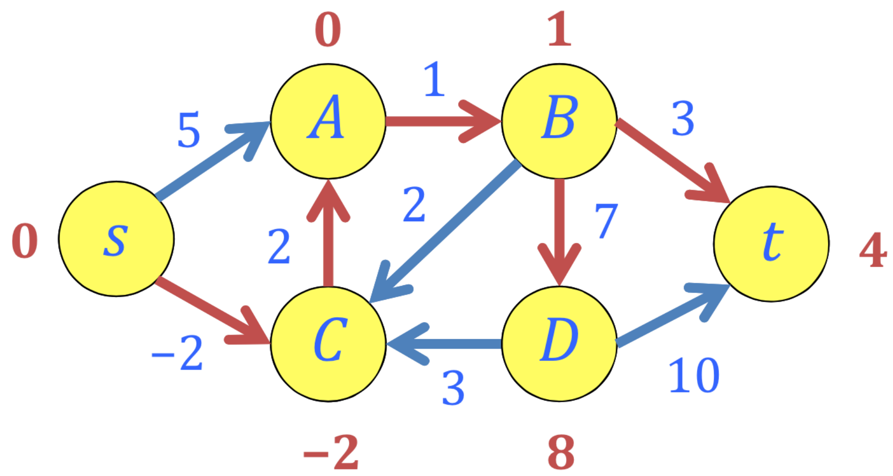




Steps	s.d/s.π	t.d/t. π	x.d/x. π	y.d/y. π	z.d/z. π
init	0/Nil	∞/Nil	∞/Nil	∞/Nil	∞/Nil
S,t		6/s			
S,y				7/s	
T,x			11/t		
T,y				7/s	
T,z					2/t
X,t		6/s			
Y,x			4/y		
Y,z					2/t
Z,s	0/nil				
Z,x			4/y		
. .	. .	. .	. .	. .	. .

The Bellman-Ford algorithm

- How did it become?



Slide Adopted From the Work of An MIT Professor, Erik Demaine

- Distance-vector routing protocol
  - Repeatedly relax edges until convergence
  - Relaxation is local!
- On the Internet:
  - Routing Information Protocol (RIP)
  - Interior Gateway Routing Protocol (IGRP)


photo by Ross Imlach, 2011
http://www.flickr.com/photos/rossimlach/5446205998/

Slide Adopted From the Work of An MIT Professor, Erik Demaine

Bellman-Ford Analysis

TOTAL: O(VE)

for v in V:
    $v.d = \infty$
    $v.\pi =$ None
$s.d = 0$
} O(V)

for i from 1 to $|V| - 1$:
    for (u, v) in E:
        relax(u, v) } O(1)
    } O(E)
} O(VE)

for (u, v) in E:
    if $v.d > u.d + w(u, v)$:
        report that a negative-weight cycle exists
} O(E)

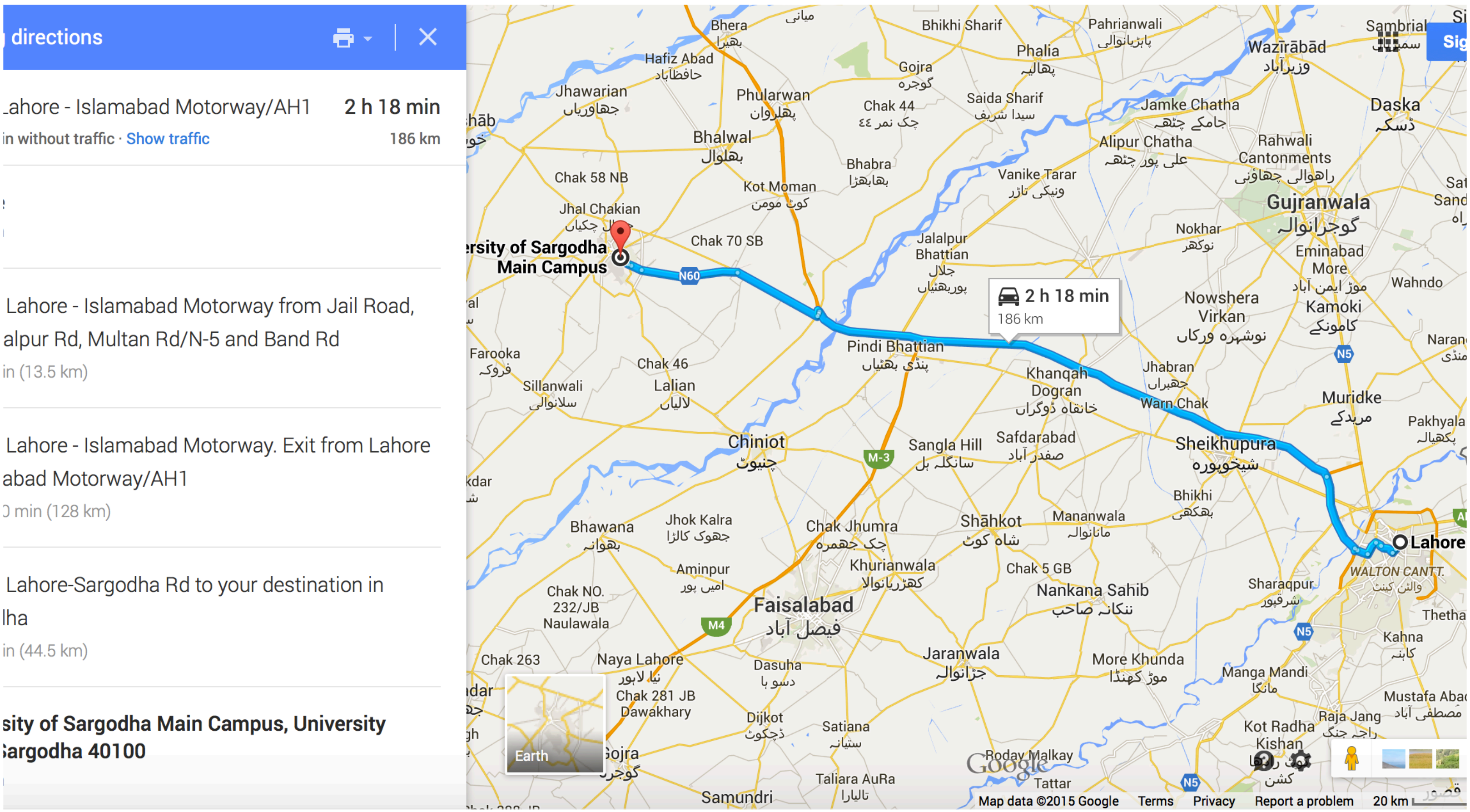
Single-source shortest paths in directed acyclic graphs

- Section 24.2 (Read it yourself)

Dijkstra's algorithm

- Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph G=(V, E) for the case in which all edge weights are nonnegative.

Dijkstra's algorithm



Dijkstra’s algorithm

- Dijkstra’s algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.
- The algorithm repeatedly selects the vertex u ε V-S with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u.
- In the following implementation, a min-priority queue Q of vertices is used, keyed by their d values.

Dijkstra's algorithm

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Dijkstra’s algorithm

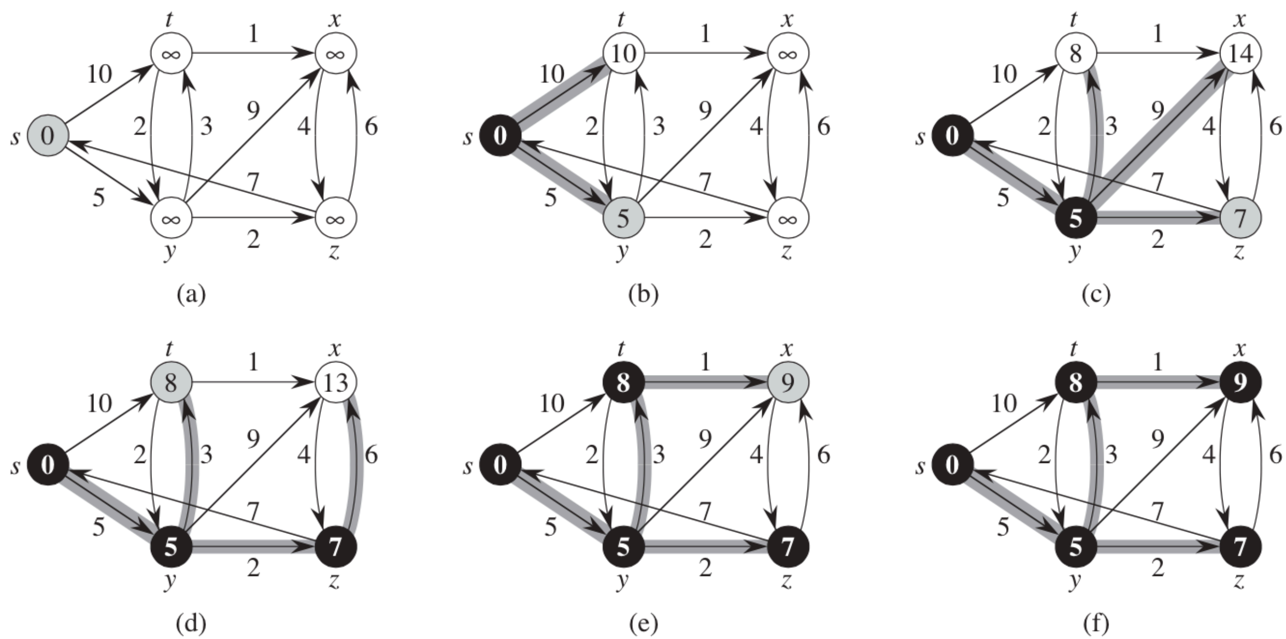


Figure 24.6 The execution of Dijkstra’s algorithm. The source s is the leftmost vertex. The shortest-path estimates appear within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation just before the first iteration of the **while** loop of lines 4–8. The shaded vertex has the minimum d value and is chosen as vertex u in line 5. **(b)–(f)** The situation after each successive iteration of the **while** loop. The shaded vertex in each part is chosen as vertex u in line 5 of the next iteration. The d values and predecessors shown in part (f) are the final values.

Dijkstra's algorithm

- Dijkstra's algorithm always chooses the "lightest" or "closest" vertex in V-S to add to set S , and hence uses the greedy strategy.
- Dijkstra's algorithm resembles both breadth-first search (Read Section 22.2 by yourself) and Prim's algorithm for computing minimum spanning trees (Read Section 23.2 by yourself).
- Time Complexity of the implementation is O(V^2). If the input graph is represented using adjacency list, it can be reduced to O(E log V) (Section 6.5) with the help of binary heap and O(E+VlogV) (Chapter 19) in case of Fibonacci heap.

Analysis of Dijkstra's algorithm

ShortestPath(G, v)

```
For all v∈V, D[v]=∞
D[s]=0
For all v∈V, P[v]=Nil
Put all v∈V into a data structure Q, using Q.Insert(v, D[v])
while (!Q.Empty()) // Q.Empty() return Boolean value
        c = Q.removeMin()
        for each neighbors c of v in Q do
              w= weight of (c,v) ∈E
               if D[c] + w < D[v] then
                  D[v] = D[c] + w
                  P[v] = c
                  Q.DecreaseKey(v,D[v])
return D[t] and optionally P
```

Analysis of Dijkstra's algorithm

- Running time using an array as a priority queue Q
  - = O(|V|) + O(|V|.time of Q.Insert()) + O(|V| . (time of Q.Empty() + time of Q.RemoveMin()) + |E| . (O(1) + time of Q.DecreaseKey())) + O(|V|)
  - = O(|V| . (time of Q.Insert() + time of Q.Empty() + time of Q.RemoveMin()) + O (|E| . time of Q.Decreasekey())
- In case of an array, Q.Insert(), Q.Empty(), and Q.DecreaseKey() take O(1) time, and Q.RemoveMin() takes O(V) time, so
  - = O(|V|. (O(1)+O(1)+O(V)) + O(|E|. O(1))
  - = $O(V^2+E)$ = $O(V^2)$

Analysis of Dijkstra's algorithm

- Running time using a heap as a priority queue Q
  - = O(|V| . (time of Q.Insert() + time of Q.Empty() + time of Q.RemoveMin()) + O (|E| . time of Q.Decreasekey())
- In case of a heap, Q.Insert, Q.RemoveMin(), and Q.DecreaseKey() take O(logV) time, and Q.Empty() takes O(1) time, so
  - = O(|V|. (O(logV)+O(1)+O(logV)) + O(|E|. O(logV))
  - = O(VlogV+ElogV) = O(V+E)logV
  - If the graph is sparse, then |E| = |V|, otherwise, E logV wins in comparison of V logV and E logV which beats the complexity of $O(V^2)$

Analysis of Dijkstra's algorithm

- Running time using a Fibonacci heap as a priority queue Q
  - = O(|V| . (time of Q.Insert() + time of Q.Empty() + time of Q.RemoveMin()) + O (|E| . time of Q.Decreasekey())
- In case of a Fibonacci heap, Q.Insert and Q.Empty() take O(1) time, similarly, Q.DecreaseKey() takes O(1) amortized time, and Q.RemoveMin() takes O(logV) time, so
  - = O(|V|. (O(1)+O(1)+O(logV)) + O(|E|. O(1))
  - = O(VlogV+E)
  - If |E| = |V|, then O(VlogV+E) becomes E logE which is equal to previous binary heap approach.
  - If $|E| = |V^2|$, then O(VlogV+E) becomes $O|V^2|$which equal to the first version of array
  - Etc.

Term Paper

- Have a look over the state-of-the-art algorithms and their issues.
  1. Build a comparative study if you find similar algorithms to solve a same problem.
  2. After understanding a state-of-the-art algorithmic model with its issue, try to propose a solution.
  3. Criticize or negate the way, a state-of-the-art algorithm is designed.
- Your term paper should include the following as a sample:
  – Title, authors profiles, abstract, keywords, introduction, methodology or design, etc; implementation, etc; discussion and issues, etc; conclusion and references.
- Deadline: before the final term paper.

Home Work # 6

24.1-5 ★

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$. Give an $O(VE)$-time algorithm to find, for each vertex $v \in V$, the value $\delta^*(v) = \min_{u \in V} \{\delta(u, v)\}$.

24.1-6 ★

Suppose that a weighted, directed graph $G = (V, E)$ has a negative-weight cycle. Give an efficient algorithm to list the vertices of one such cycle. Prove that your algorithm is correct.

24.2-1

Run DAG-SHORTEST-PATHS on the directed graph of Figure 24.5, using vertex r as the source.

24.3-8

Let $G = (V, E)$ be a weighted, directed graph with nonnegative weight function $w : E \to \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(WV + E)$ time.