Design and Analysis of Algorithms
Unit -4

Bharathi R
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Unit 4: Greedy Technique

Dijkstra’s Algorithm

Bharathi R

Department of Computer Science & Engineering

Single Source Shortest Paths Problem: Given a weighted connected (directed) graph G, find shortest paths from source vertex ***s*** to each of the other vertices

Dijkstra's algorithm: Similar to Prim's MST algorithm, with a different way of computing numerical labels: Among vertices not already in the tree, it finds vertex ***u*** with the smallest sum

$$d_v + w(v,u)$$

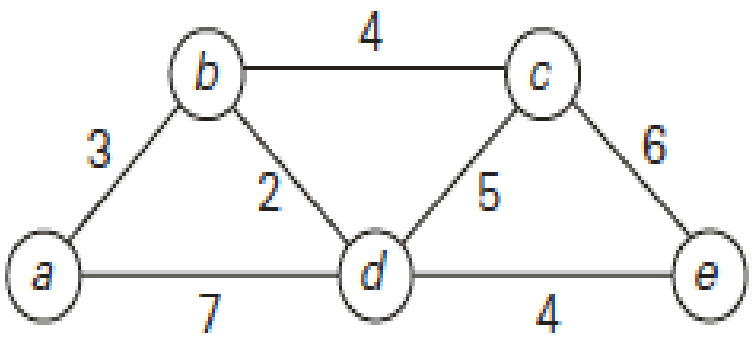
where

v is a vertex for which shortest path has been already found on preceding iterations (such vertices form a tree rooted at ***s***)

d_v is the length of the shortest path from source ***s*** to ***v***

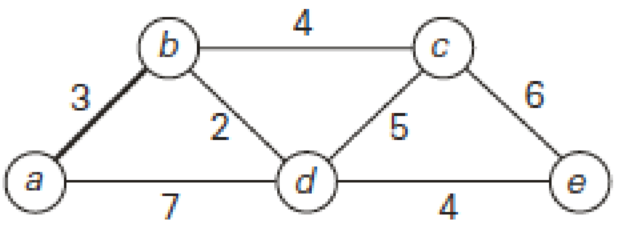
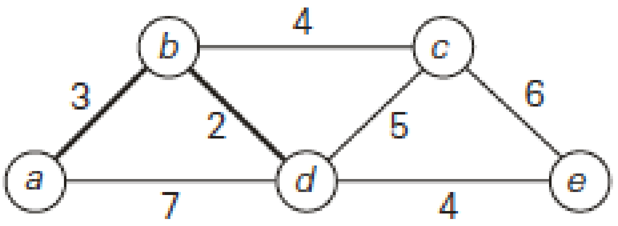
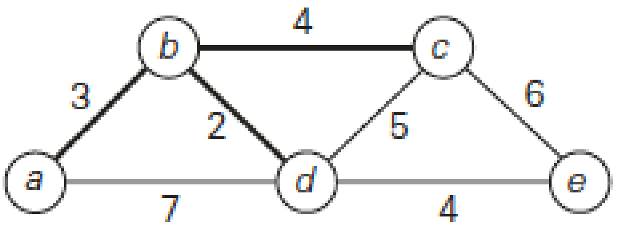
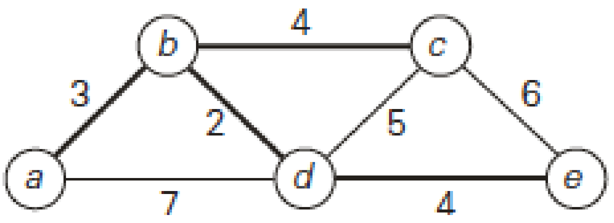
$w(v,u)$ is the length (weight) of edge from ***v*** to ***u***

Example



Design and Analysis of Algorithms

Dijkstra's Algorithm-Example

Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

The next closest vertex is shown in bold.

Dijkstra's Algorithm

ALGORITHM *Dijkstra*(G, s)

```
//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph G = ⟨V, E⟩ with nonnegative weights
//        and its vertex s
//Output: The length d_v of a shortest path from s to v
//          and its penultimate vertex p_v for every vertex v in V
Initialize(Q)   //initialize priority queue to empty
for every vertex v in V
    d_v ← ∞;   p_v ← null
    Insert(Q, v, d_v)   //initialize vertex priority in the priority queue
d_s ← 0;   Decrease(Q, s, d_s)   //update priority of s with d_s
V_T ← ∅
for i ← 0 to |V| − 1 do
    u* ← DeleteMin(Q)   //delete the minimum priority element
    V_T ← V_T ∪ {u*}
    for every vertex u in V − V_T that is adjacent to u* do
        if d_u* + w(u*, u) < d_u
            d_u ← d_u* + w(u*, u);   p_u ← u*
            Decrease(Q, u, d_u)
```

Dijkstra's Algorithm- Time Efficiency

The time efficiency of Dijkstra's algorithm depends on the data structures used for implementing the priority queue and for representing an input graph itself.

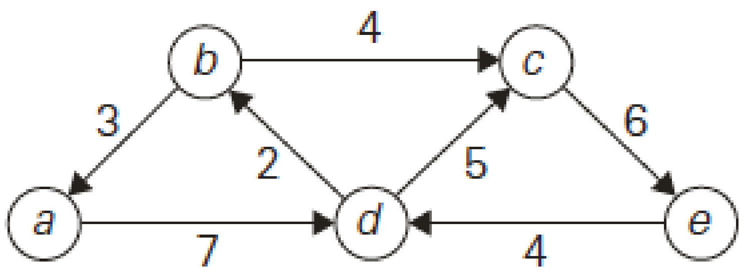
- **$O(|V|^2)$** for graphs represented by weight matrix and array implementation of priority queue
- **$O(|E|\log|V|)$** for graphs represented by adj. lists and min-heap implementation of priority queue

Notes on Dijkstra's Algorithm

- Correctness can be proven by induction on the number of vertices.
- Doesn't work for graphs with negative weights (whereas Floyd's algorithm does, as long as there is no negative cycle).
- Applicable to both undirected and directed graphs
- Don't mix up Dijkstra's algorithm with Prim's algorithm!

Exercises

Solve the following instances of the single-source shortest-paths problem with vertex a as the source:



References

Chapter-9 Greedy Technique
Introduction to the Design & Analysis of Algorithms- Anany Levitin

THANK YOU

Bharathi R

Department of Computer Science & Engineering

rbharathi@pes.edu