Design and Analysis of Algorithms

L24: Dijkstra's Algorithm

Single Source Shortest Path

+

Bellman-Ford Algorithm

Dr. Ram P Rustagi

Sem IV (2020-Even)

Dept of CSE, KSIT

rprustagi@ksit.edu.in

Resources

- Text book 1: Sec 9.3 - Levitin
- R1: Introduction to Algorithms
  - Cormen et al.
- NPTel: DAA
  - – https://onlinecourses.nptel.ac.in/noc20_cs27/unit?unit=29&lesson=30
  - – https://onlinecourses.nptel.ac.in/noc20_cs27/unit?unit=29&lesson=31

Single Source Shortest Path

- Applications
  - Supplying deliveries from a factory to various godowns
    - Minimum time/cost
  - KSIT: Moving from quadrangle to your class rooms
    - Minimum time taken

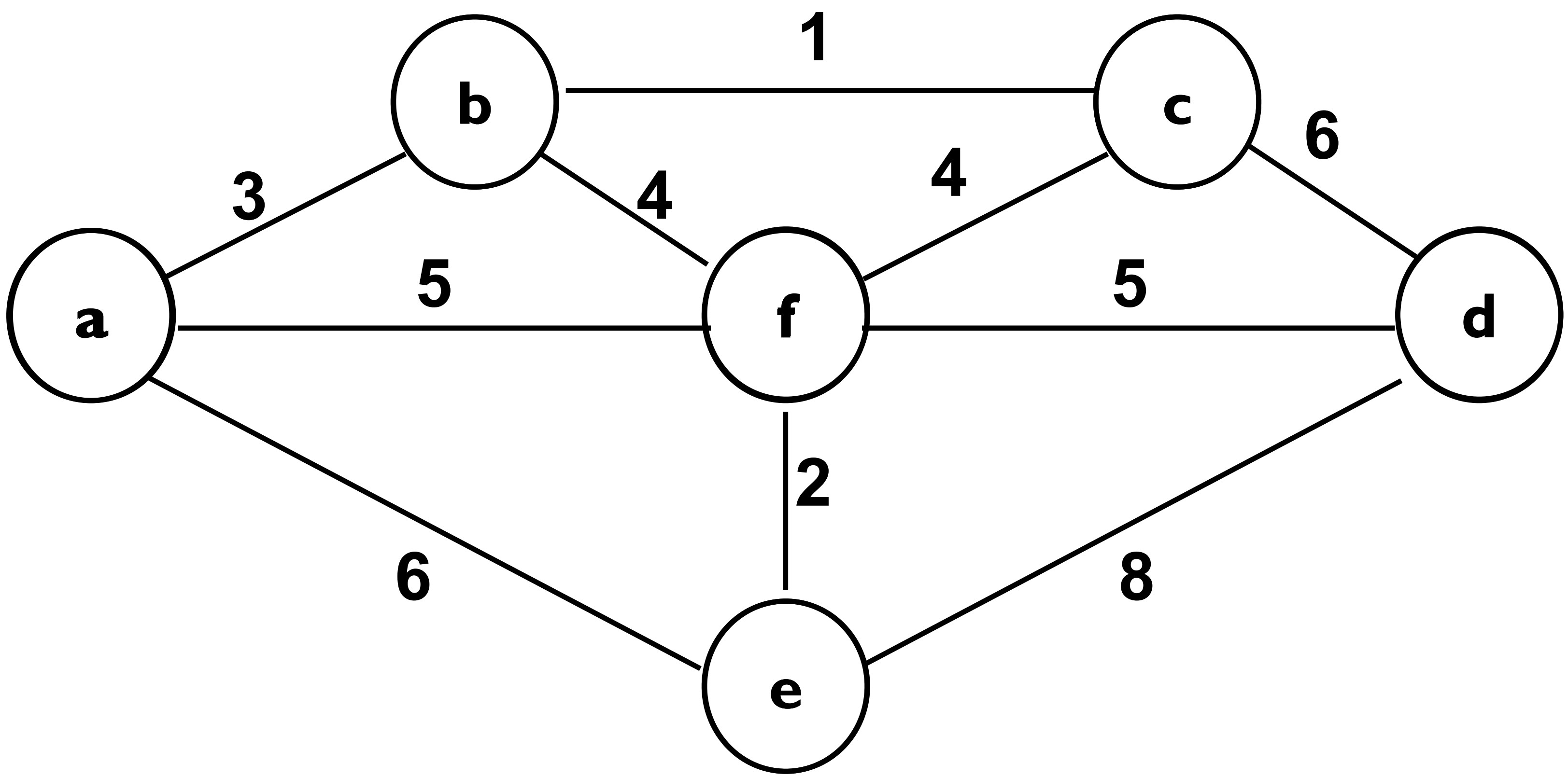
DFS and BFS Traversal

- Both DFS and BFS perform graph traversal
- Both take linear time in terms of size of graph when using adjacency lists i.e. `O(|E|)`
- We can find path by keeping parent information
- BFS computes shortest path in terms of number of edges i.e. all edges has have same cost
  – Does not work when edges have different costs
- In DFS, vertex numbering (previsit(), postvisit()) reveals interesting information about graph
  – e.g. backedges, forward edges, cross edges
  – it does not find shortest path though

Single Source Shortest Path

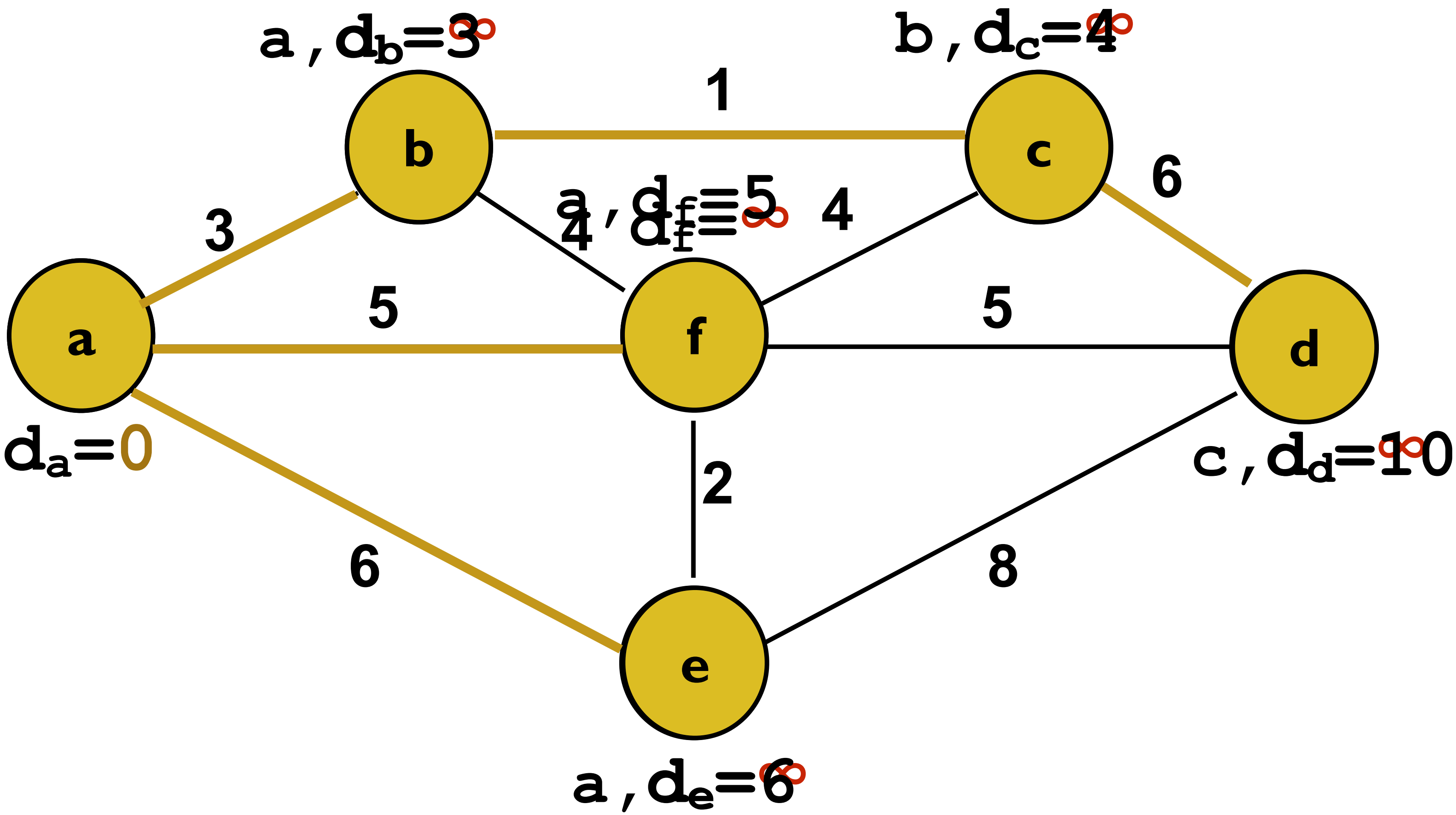
- Goal: Given a weighted connected (directed) graph G, find shortest paths from source vertex s to each of the other vertices
  - Path cost: sum of cost each edge on the path
- Dijkstra's algorithm
  - Approach similar to Prim's algorithm for MST
  - Computes numerical labels differently
  - Among vertices not in the tree,
    - Find the vertex v with the smallest sum $d_v + w(u, v)$, where
  - $u \in V$ whose shortest path found in previous iteration
  - d_v is the length of shortest path from s to v

Example: Dijkstra's Algorithm



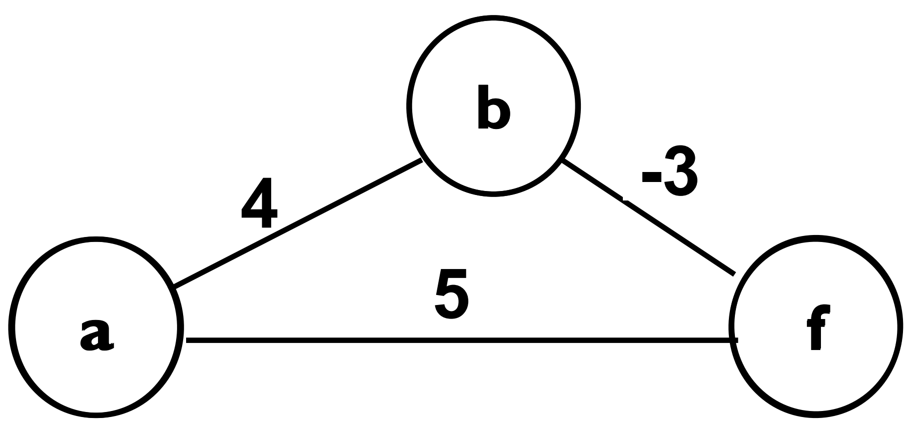
- Q: Construct an SSSP using Dijkstra's algo starting from vertex `a`

Example: Dijkstra's Algorithm



Notes on Dijkstra's Algorithm

- Proof of correctness:
  - Using induction
- Works with graph with +ve weights only
  - Build a counter example with -ve weight where Dijkstra's algorithm does not work
- Works for both directed and undirected graphs



Algorithm: Dijkstra's Algorithm

```
Algo Dijkstra(G,s)
```

// i/p: a weighted connected graph G=(V,E), and src s
//   all edges are non-negative weights
//o/p: Length dv of a shortest path from s to v.
//   along with it predecessor vertex from v to s.
Initialize(Q) // priority queue of vertices is empty initially
for each vertex v∈V, do
    $d_V \leftarrow \infty$; $p_V \leftarrow$ Null;
    Insert(Q, v, d_V) // initialize vertex priority in priority Q
$d_S \leftarrow 0$;
Decrease (Q, s, d_S)
$p_S \leftarrow$ Null;
$V_T \leftarrow \emptyset$

Algorithm: Dijkstra's Algorithm…

```
Algo Dijkstra(G,s)…
    for i=0 to |V|-1 do
        u = DeleteMin(Q)  //time implememtation based
        V_T= V_T U{u}
        for every vertex w∈V-V_T adjacent to w, do
            if d_u+weight(u,w)< d_w, then
                d_w←d_u+weight(u,w)
                p_w←u
                Decrease(Q,w,d_w) //time implementation based
            fi
        end //for w∈V-V_T
    end //for i=0
end //algo
```

Analysis: Dijkstra's Algorithm…

- Implementation using Adjacency matrix
  - Priority Q using unsorted array
  - Outer for loop (`i=0 to |V|-1`) : $O(|V|)$ times
  - DeleteMin takes $O(|V|)$ times
    - Total time for all vertices: $O(|V|^2)$
  - Decrease(Q, w, d_w) takes $O(1)$ time
    - Total time for all vertices: $O(|E|)$
  - Time Complexity: $O(|V|^2)$

Analysis: Dijkstra's Algorithm

- Implementation using Adjacency List
  - Priority Q using Heap
  - Outer for loop (`i=0 to |V|-1`): `O(|V|)` times
  - DeleteMin takes `O(lg|V|)` times
    - Total time for all vertices: `O(|V|lg|V|)`
  - Decrease(`Q`, `w`, d_w) takes `O(lg|V|)` time
    - Total time for all vertices: `O(|E|lg|V|)`
  - Time Complexity: `O(|E|lg|V|)`

Greedy Approach: Disjkstra's Algo

- Greedy Approach:
  – Make a sequence of choices
    - One choice at a time
  – Next choice is based on current best value
    - Once a choice is made (a node is chosen), this choice is never changed
- Dijkstra's Algo:
  - Select vertex with minimum disance from source
    - (from via previously selected vertices)
  - Greedy approach is optimal in this case

Questions

- Q1: what adjustments if any need to be made in Dijkstra's algorithm to solve the single-source shortest-paths problem for directed weighted graphs.
- Ans:
  - Do we need any changes? Just follow the directed edges.

Questions

- Q2: Find a shortest path between two given vertices of a weighted graph or digraph. (This variation is called the single-pair shortest-path problem.)
- Ans:
  – Start from one vertex as source
  – Iterate the for loop till you find 2nd vertex.

Questions

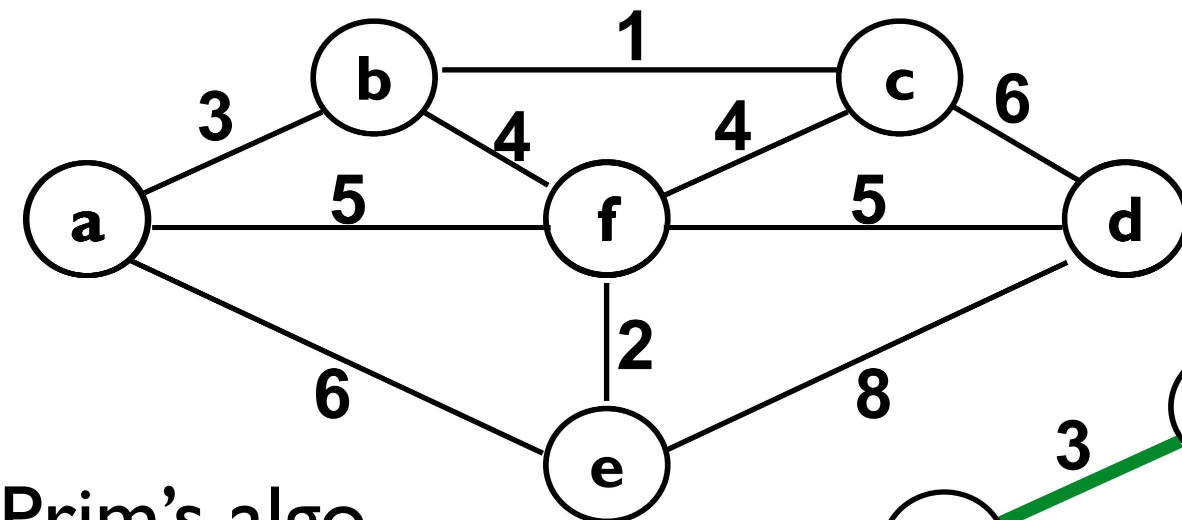
- Q3: Find the shortest paths to a given vertex from each other vertex of a weighted graph. (This variation is called the single destination shortest-paths problem.)
- Ans: Undirected graph
  - Start from the destination vertex as source
  - Find the shortest path from this to all other vertices
  - Reverse the path
- Ans: directed graph
  - Reverse the direction of all edges.
  - Start from the destination vertex as source
  - Find the shortest path from this src to all other vertices
  - Reverse the path

Questions

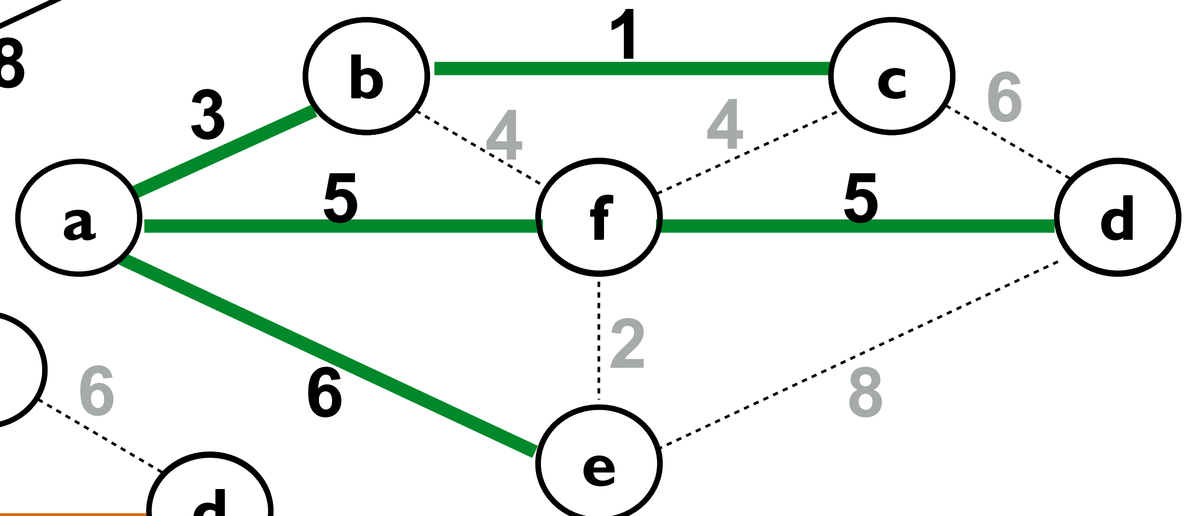
- Q4: Solve the single-source shortest-path problem in a graph with non-negative numbers assigned to its vertices (and the length of a path defined as the sum of the vertex numbers on the path).
- Hint:
  - The weight of the edge is sum of non-negative numbers assigned to vertices of the corresponding edge.

Dijkstra vs Prim Algo

- Do both gave same tree? Consider the example

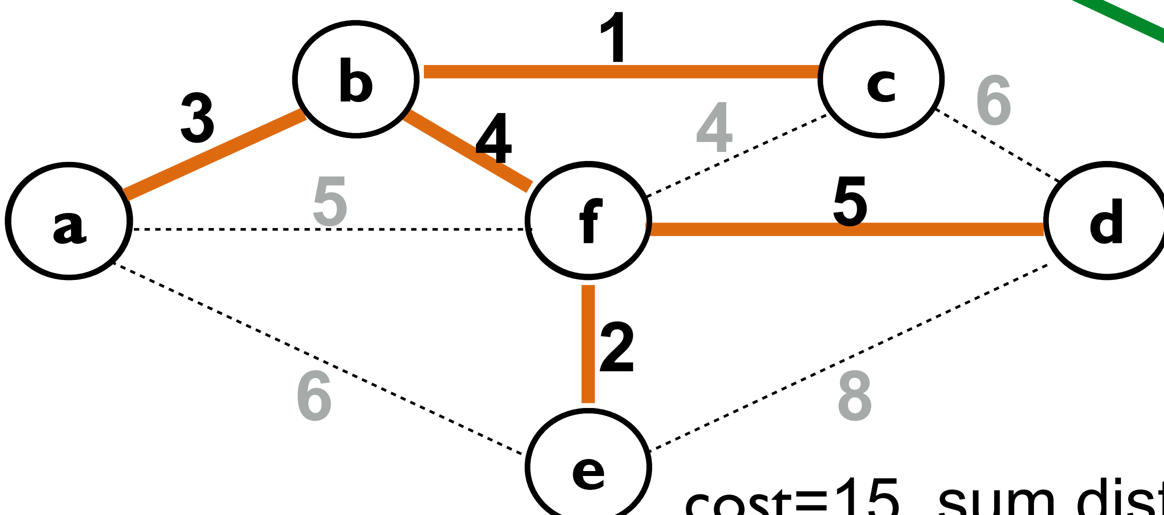


Dijkstra algo



cost=20, sum distance=28

Prim's algo



cost=15, sum distance=38

Summary

- Dijkstra's algorithm
  - Keeps shortest length for each vertex from source `s`
  - Keep predecessor with each vertex towards `s`
  - Different from Prim's algorithm
    - Dijkstra: Chooses vertex with min shortest length from source
    - Prim: chooses edges with minimum weight from curren spanning tree (and not from src)