DFS, BFS,
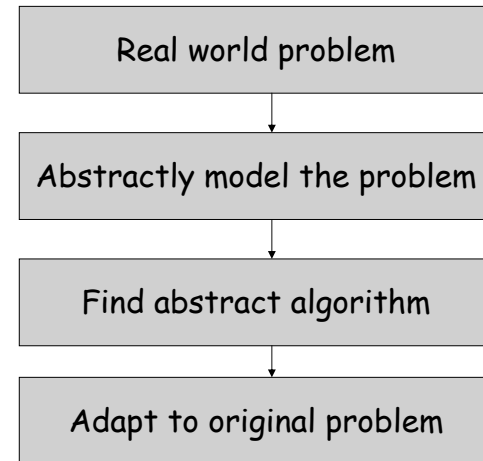# Shortest Path Problems

CSE 373
Data Structures
Unit 13

Reading: Sections 9.3, 9.6, 10.3.4

Applied Algorithm Scenario

Real world problem

↓

Abstractly model the problem

↓

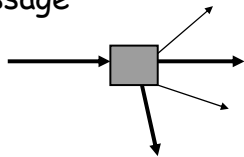Find abstract algorithm

↓

Adapt to original problem

Broadcasting in a Network

- Network of Routers
  - Organize the routers to efficiently broadcast messages to each other.
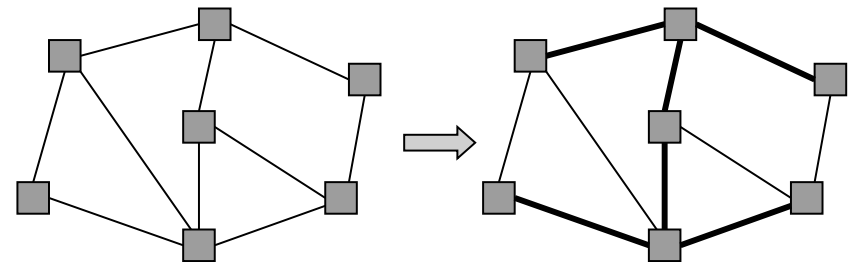
Incoming message

- Duplicate and send to some neighbors.
- Eventually all routers get the message

Goal: Minimize the number of messages.

Spanning Tree in a Graph

Vertex = router
Edge = link between routers

Spanning tree
 - Connects all the vertices
 - No cycles

Spanning Tree Problem

- Input: An undirected graph G = (V,E).
  G is connected.
- Output: T contained in E such that
  - (V,T) is a connected graph
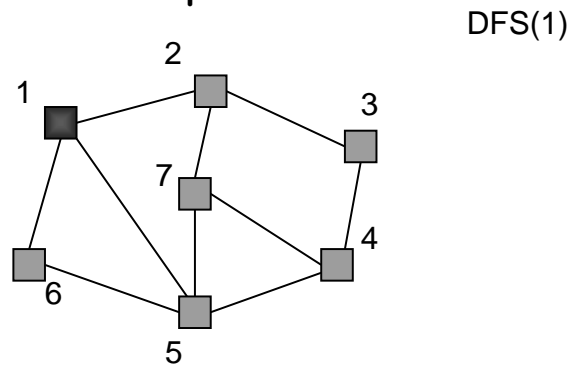 - (V,T) has no cycles

Depth First Search Algorithm

- Recursive marking algorithm
- Initially every vertex is unmarked

```
DFS(i: vertex)
   mark i;
   for each j adjacent to i do
      if j is unmarked then DFS(j)
end{DFS}
```
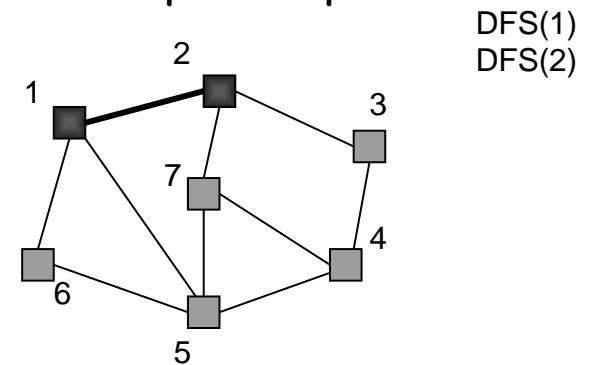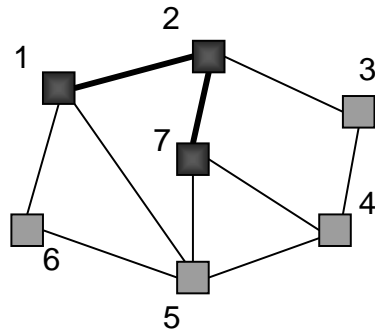
Example of Depth First Search

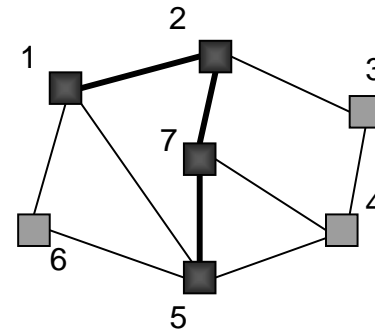DFS(1)

Example Step 2

DFS(1)
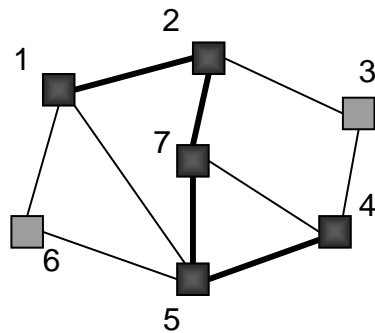DFS(2)

Example Step 3

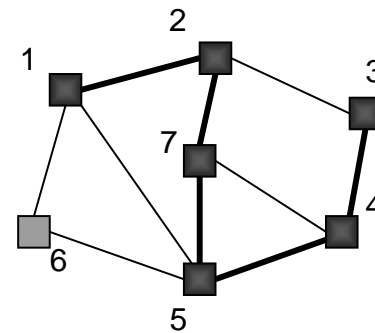DFS(1)
DFS(2)
DFS(7)

Example Step 4

DFS(1)
DFS(2)
DFS(7)
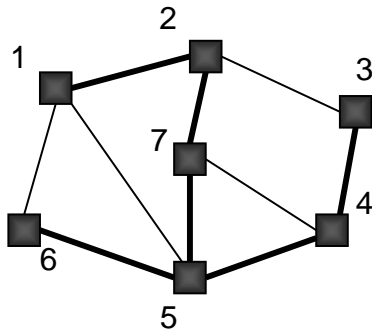DFS(5)

Example Step 5

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)

Example Step 6

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
DFS(3)

Example Step 7

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
DFS(3)
DFS(6)

Note that the edges traversed in the depth first search form a spanning tree.

Spanning Tree Algorithm

```
Main
T := empty set;
ST(1);
end{Main}
```

```
ST(i: vertex)
    mark i;
    for each j adjacent to i do
        if j is unmarked then
            Add {i,j} to T;
            ST(j);
end{ST}
```

The addition to DFS

Applied Algorithm Scenario

Real world problem

Wrong problem

Abstractly model the problem

Wrong model

Find abstract algorithm

Incorrect algorithm
poor performance

Adapt to original problem
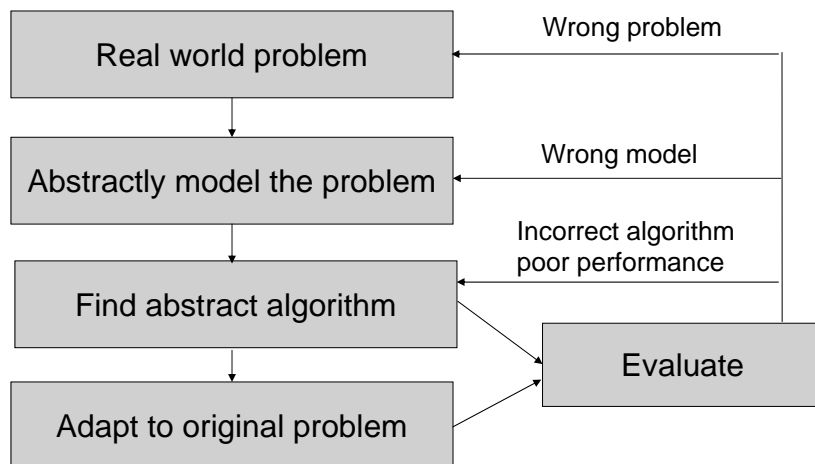
Evaluate

Evaluation Step Expanded

Algorithm Correct? — no →
- New algorithm
- New model
- New problem

yes

Choose Data Structure

Performance? — unsatisfactory →
- New data structure
- New algorithm
- New model

satisfactory

Implement
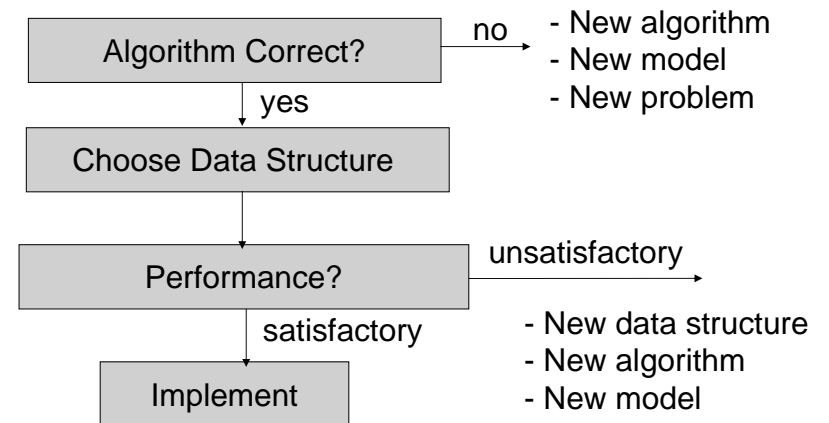
Correctness of ST Algorithm

- There are no cycles in T
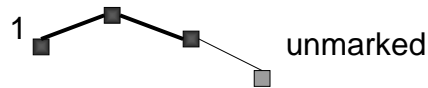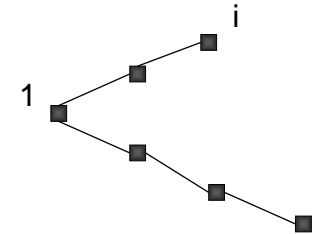 - This is an invariant of the algorithm.
  - Each edge added to T goes from a vertex in T to a vertex not in T.
- If G is connected then eventually every vertex is marked.

1   unmarked

Correctness (cont.)

- If G is connected then so is (V,T)

i

1

j

Data Structure Step

Algorithm Correct?   no   - New algorithm
- New model
- New problem

yes

Choose Data Structure

Performance?   unsatisfactory

satisfactory   - New data structure
- New algorithm
- New model

Implement
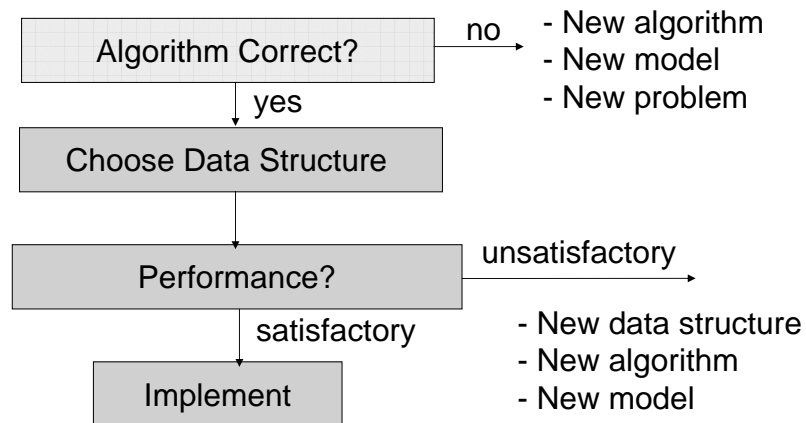
Data Structure Choice

- Adjacency lists
  - Good for sparse graphs
  - Supports depth first search
- Adjacency matrix
  - Good for dense graphs
  - Supports depth first search

Spanning Tree with Adjacency Lists

```
Main
    G is array of adjacency lists;
    M[i] := 0 for all i;
    T is empty;
    Spanning_Tree(1);
end{Main}
```

M is the marking array
(entry for each vertex).

Node of linked list:
vertex | | next

```
ST(i: vertex)
    M[i] := 1;
    v := G[i];
    while (v ≠ null)
        j := v.vertex;
        if (M[j] = 0) then
            add {i,j} to T;
            ST(j);
        v := v.next;
end{ST}
```

Performance Step

Algorithm Correct? →no→
- New algorithm
- New model
- New problem

↓ yes

Choose Data Structure

↓

Performance? →unsatisfactory→

↓ satisfactory

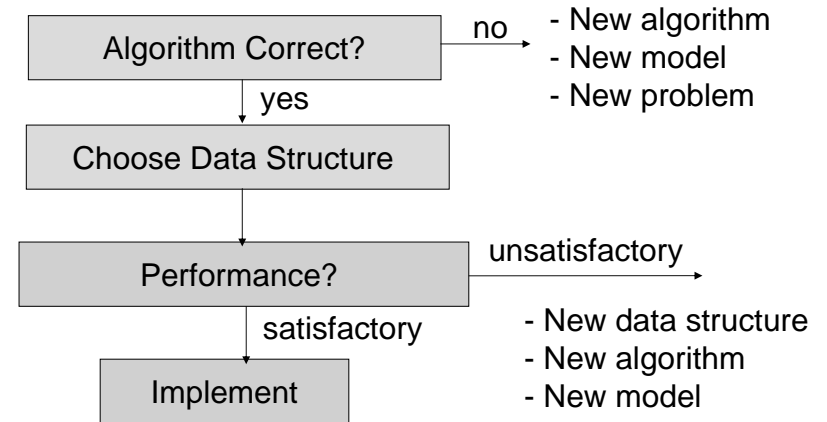- New data structure
- New algorithm
- New model

Implement

Performance of ST Algorithm

- n vertices and m edges
- Connected graph ($m \geq n-1$)
- Space complexity $O(m)$
- Time complexity $O(m)$ - for each edge we perform $O(1)$ operations in each of the two endpoints.
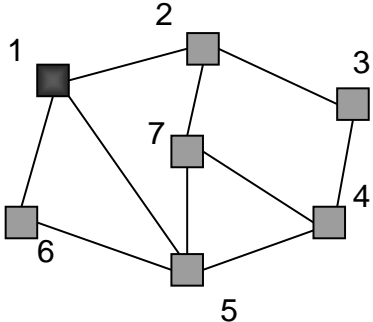
Other Uses of Depth First Search

- Popularized by Hopcroft and Tarjan 1973
- Connected components
- Strongly connected components in directed graphs
- Topological sorting of a acyclic directed graphs
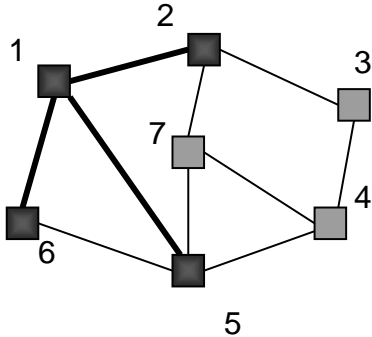- Maze solving

ST using Breadth First Search 1
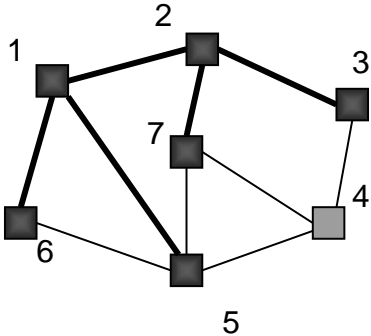
- Uses a queue to order search
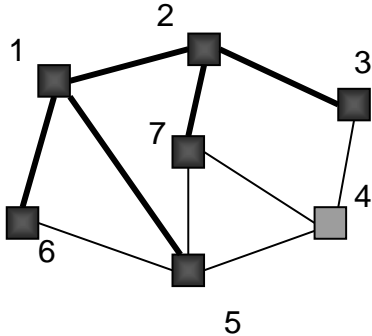

Queue = 1

Breadth First Search 2

Queue = 2,6,5

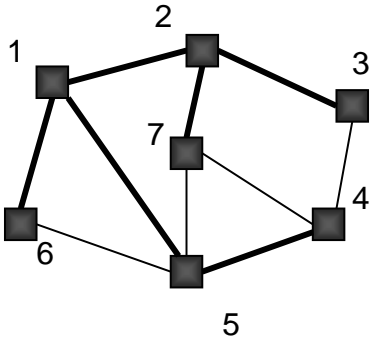## Breadth First Search 3

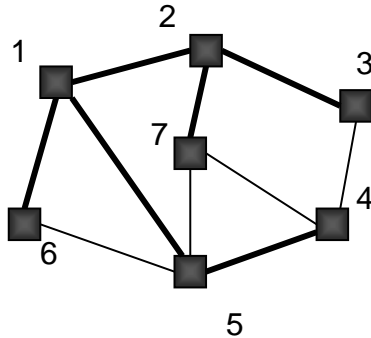Queue = 6,5,7,3

Breadth First Search 4

Queue = 5,7,3

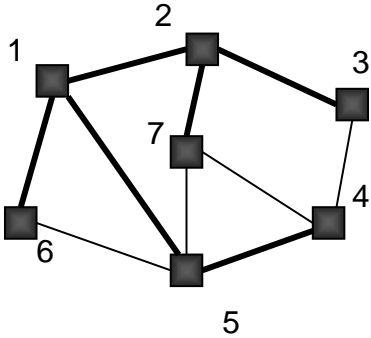# Breadth First Search 5

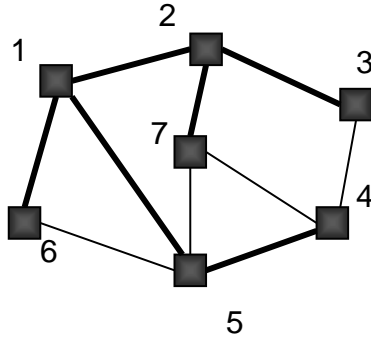Queue = 7,3,4

Breadth First Search 6

Queue = 3,4

Breadth First Search 7

Queue = 4

Breadth First Search 8

Queue =

Spanning Tree using Breadth First Search (BFS)

```
Initialize T to be empty;
Initialize Q to be empty;
Enqueue(1,Q) and mark 1;
while (Q is not empty) do
    i := Dequeue(Q);
    for each j adjacent to i do
        if j is not marked then
            add {i,j} to T;
            mark j;
            Enqueue(j,Q);
```

Depth First vs Breadth First

- Depth First
  - Stack or recursion
  - Many applications
- Breadth First
  - Queue (recursion no help)
  - Can be used to find shortest paths from the start vertex
- Both are $O(|E|)$

Shortest-path Algorithms

- Scenario: One router creates messages (source). Each message needs to reach other routers (one or more) along the shortest possible path.
- Abstraction: given a vertex s, find the shortest path from s to any other vertex of G.
- Other shortest path problems:
  - Different edges have different lengths (delay, cost, etc.)
  - All-pair shortest path problem: no specific source.

Using BFS for Shortest-path

- Given a vertex s, find the shortest path from s to any other vertex of G.

A 'centralized' version of BFS:
1. Label vertex s with 0.
2. $i \leftarrow 0$
3. Find all unlabeled vertices adjacent to at least one vertex labeled i. If none are found, stop.
4. Label all the vertices found in (3) with $i + 1$.
5. $i \leftarrow i + 1$ and go to (3).

BFS for Shortest Path (i=0)
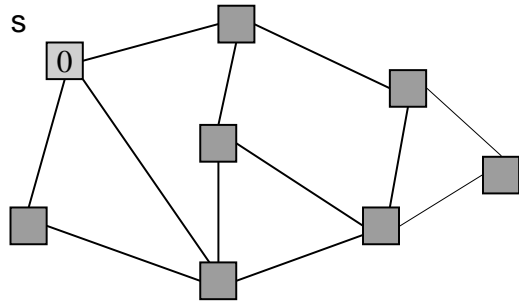
Vertices whose distance from s is 0 are labeled.

BFS for Shortest Path (i=1)

Vertices whose distance from s is 1 are labeled.

BFS for Shortest Path (i=2)

Vertices whose distance from s is 2 are labeled.

BFS for Shortest Path (i=3)

Vertices whose distance from s is 3 are labeled.

In the next iteration we find out that the whole graph is labeled and we stop.

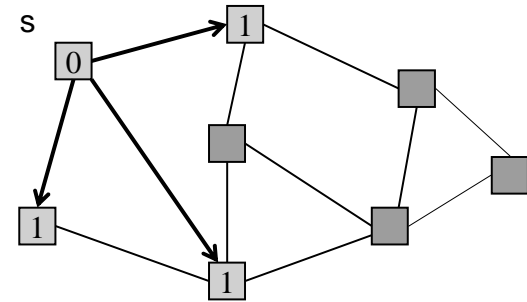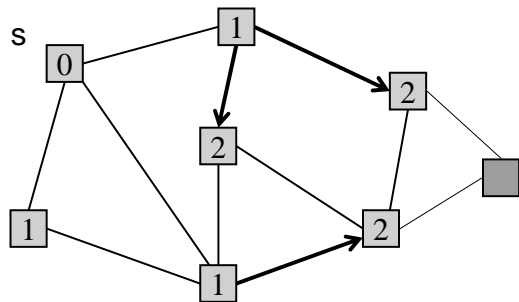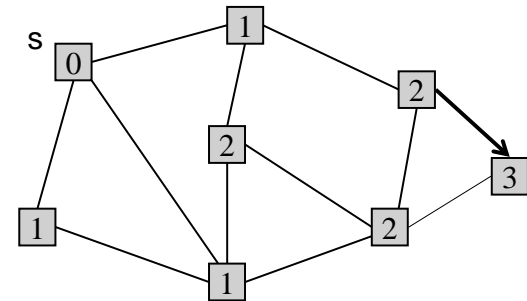# The BFS Tree

Theorem: Each vertex is labeled by it its length from s.

Proof: By induction on the label.

For any $v \neq s$, let p(v) be the vertex that 'discovered' v in BFS.

Then T={(p(v),v)} is a directed spanning tree rooted in s, and for each vertex v, the path from s to v in T is a shortest path from s to v in G.

Note: the 'centralized' version is for simplification only. When implemented, we need the queue as before.
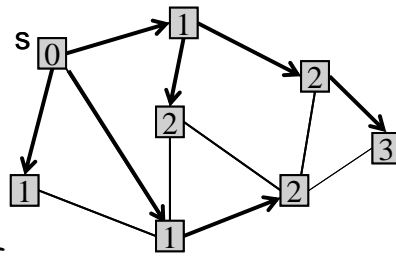
Single-Source Shortest Paths (Dijkstra's algorithm)

- Using BFS, we solve the problem of finding shortest path from s to any vertex v.
- What if edges have associated costs or distances? (BFS assumes edge costs are all 1.)
- Assume each edge (u,v) has non-negative weight c(u,v).
- A weight of a path = total weights of all edges on path.
- Problem: Find, for each vertex v, a shortest (minimum weight) path from s to v.

Idea of Dijkstra's Algorithm:

- Maintain:
  - $\lambda[0..n-1]$ where $\lambda(v)$ is the cost of best path from s to v found so far, and
  - T, set of vertices v for which $\lambda(v)$ is not yet known to be optimal.
- Initially:
  - $\lambda(s) = 0$; $\lambda(v) = \infty$ for all v other than s.
  - T = V.
- In each step:
  - remove that v in T with minimum $\lambda(v)$
  - update those w in T s.t. (v,w) in E and $\lambda(w) > \lambda(v) + c(v,w)$.
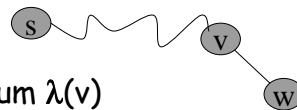
Dijkstra's Algorithm

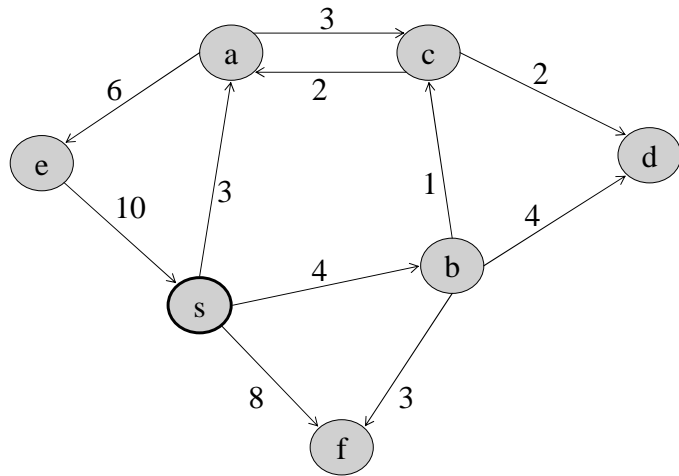Assumption: $c(u,v) = \infty$ if (u,v) not in E.

1. $\lambda(s) \leftarrow 0$ and for all $v \neq s$, $\lambda(v) \leftarrow \infty$.
2. $T \leftarrow V$.
3. Let u be a vertex in T for which $\lambda(u)$ is minimum.
4. For every edge, if $v \in T$ and $\lambda(v) \geq \lambda(u) + c(u,v)$ then $\lambda(v) \leftarrow \lambda(u) + c(u,v)$.
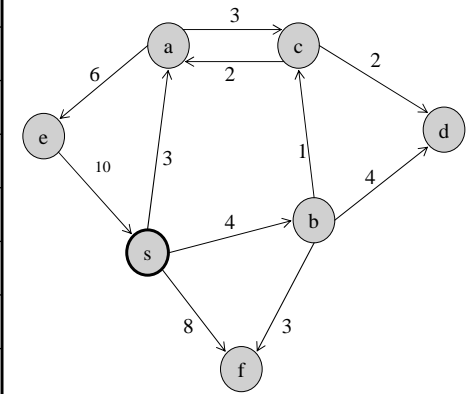5. $T \leftarrow T - \{u\}$, if T is not empty go to step 3.

Dijkstra's Algorithm - Example

Dijkstra's Algorithm - Example

|	init	u=s	u=a
s	0	0 *	0 *
a	∞	3	3 *
b	∞	4	4
c	∞	∞	6
d	∞	∞	∞
e	∞	∞	9
f	∞	8	8

In class exercise: complete the execution.

* non-T vertices.

Why is this Algorithm Correct?

- **Theorem:** At the termination of the algorithm, $\lambda(v)$ is the length of the shortest path from s to v for each vertex v of G.

- **Proof:** by induction on |V-T|.

- **Inductive hypothesis:** Let |V-T|=k.
  - $\forall v$ in V-T, $\lambda(v)$ is the length of the shortest path from s to v.
  - the vertices in V-T are the k closest vertices to s.
  - $\forall v$ in T, $\lambda(v)$ is the length of the shortest path from s to v that only goes through vertices in V-T.

k = 4
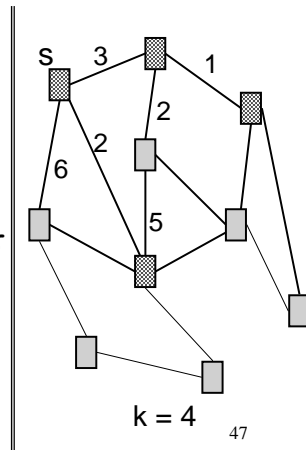
Why is this Algorithm Correct?

- **Base case: |V–T|=1, T=V–{s}.**
  - for every v in V-T, $\lambda(v)$ is the length of shortest path from s to v.
  - ✓ we init $\lambda(s)$ =0.
  - the vertices in V-T are the k closest vertices to s.
  - ✓ V-T={s}. s is surely the closest to s.
  - for every v in T, $\lambda(v)$ is the length of shortest path from s to v that only goes through vertices in V-T.
  - ✓ At this stage, $\lambda(v) = \infty$ for all v in V-T.

The λ values of vertices in V-T are correct and for each such v, the shortest path from s to v only goes through vertices in V-T

- Induction Step: Suppose true for first k steps. The SP to the $(k+1)^{st}$ closest vertex, say w, can go through only vertices in V-T, otherwise, there would be a closer vertex. Therefore, when selecting the min, we select the $(k+1)^{st}$ closest vertex to s.

  Say w is added.

  New λ value for a vertex x is min of old λ value and λ(w) + c(w,x)

Dijktra's Algorithm - Run Time Analysis

Implementation 1:
  - Adjacency lists.
  - An array for the λ values.

Complexity:

In each iteration:

1. Finding a vertex u in T with minimal λ

   In the whole execution: $n+(n-1)+(n-2)+…+1 = O(n^2)$

2. Updating the λ-values of u's neighbors:

   In each iteration we check degree(u) values.

   The total sum of the degrees in 2m à O(m)

All together: $O(m+n^2)= O(n^2)$ (remember, $m≤n(n-1)$)

Dijktra's Algorithm - Run Time Analysis

- Implementation 2: data structure: priority queue
- Stores set S (in our case, this is T) such that there is a linear order on key values (in our case the key is the λ value).
- Supports operations:
  - Insert(x) - insert element with key value x into set.
  - FindMin() - return value of smallest element in set.
  - DeleteMin() - delete smallest element in set.
- and usually:
  - Lookup(x), Delete(x)

Priority-Queue Implementations

- Priority-Queue can be implemented such that each of these operations takes O(log n) time for sets of size n.

Running time of Dijkstra's algorithm:

We need to consider insertions, delete Mins, lookups, modifying λ values.

Running Time of Dijkstra's Algorithm:

- n insertions: $O(n \log n)$ time
- n deleteMins: $O(n \log n)$ time
- m lookups: $O(m \log n)$ time
- m λ value mods: $O(m \log n)$ time

- Running time: $O((n + m) \log n)$

- The $O(n^2)$ is better for dense graphs

## Single-Source Shortest Paths (Bellman-Ford's algorithm)

- each edge (u,v) has a weight c(u,v).
- c(u,v) might be negative, but there are no negative cycles.

1. $\lambda(s) \leftarrow 0$ and for every $v \neq s$, $\lambda(v) \leftarrow \infty$.
2. As long as there is an edge such that $\lambda(v) > \lambda(u) + c(e)$ replace $\lambda(v)$ by $\lambda(u) + c(e)$.

For our purposes ∞ is not greater than $\infty + k$, even if k is negative.

Bellman-Ford algorithm

- How do we implement this algorithm?
- Order the edges: $e_1, e_2, ..., e_{|E|}$.
- Perform step 2 by first checking e_1, then e_2, etc., After the first such sweep, go through additional sweeps, until an entire sweep produces no improvement.

- Running Example:

e3, 4
e9, 5   a   c
e1, -3
e   e5, 4   e4, -1
e8, 10
s   e2, 5   b
e6, 3   e7, -3
d
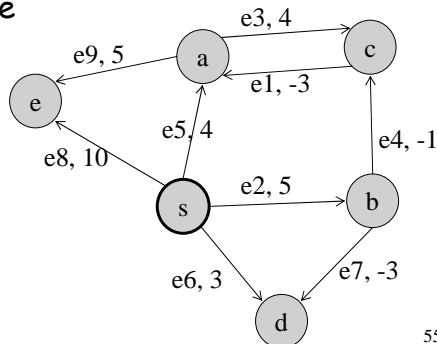
BF algorithm – correctness and run time analysis

- Theorem: if a shortest path from s to v consists of k edges, then by the end of the kth sweep v will have its final label.
- Proof: induction on k (not here).
- Since k is bounded by $|V|$ (remember, no negative cycles), step 2 is performed at most $|E| \cdot |V|$ times.
- Each comparison in step 2 can takes $O(1)$ if the graph is kept in an Adjacency Matrix (with the weights) and an array with the $\lambda(v)$ values.
- à The time complexity of BF is $O(|E| \cdot |V|)$.