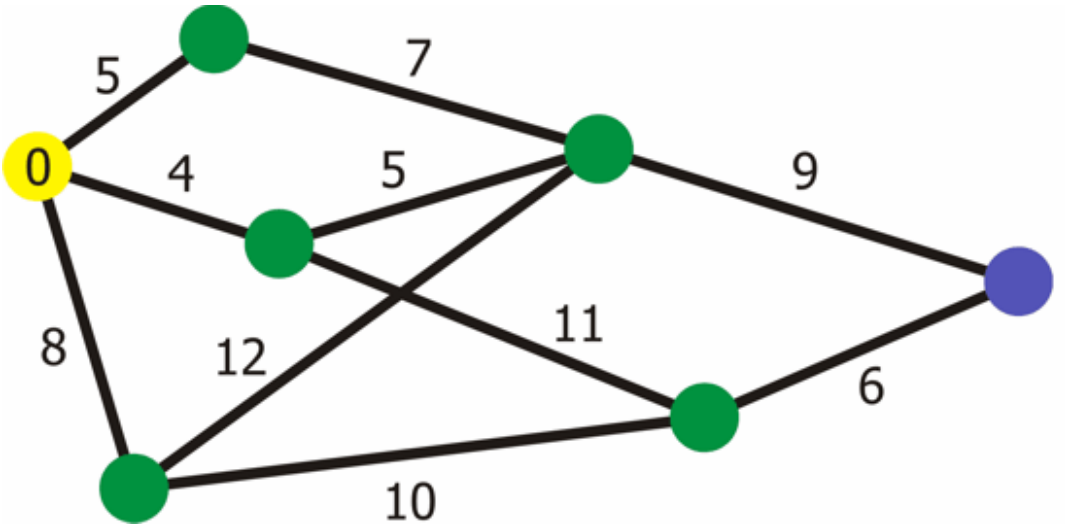
Dijkstra's Shortest-Path Algorithm

COMP215: Design & Analysis of Algorithms

KALAMAZOO COLLEGE

Today



- The Single-Source Shortest Path Problem
- Dijkstra's Algorithm
- Implementation and Running Time

The Single-Source Shortest Path Problem

- The **Single-Source Shortest Path** (SSSP) problem consists of finding the shortest paths between a given vertex v and all other vertices in the graph.
- The shortest path problem can be defined for graphs whether undirected, directed, or mixed.

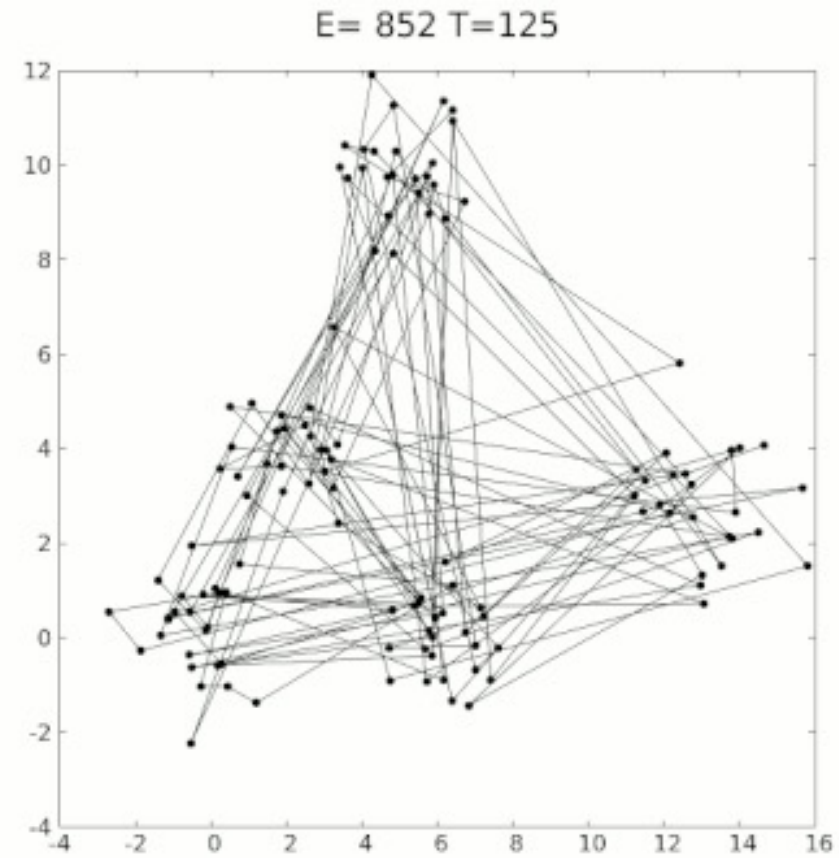
Problem: Single-Source Shortest Paths

Input: A directed graph $G = (V, E)$, a starting vertex $s \in V$, and a nonnegative *length* ℓ_e for each edge $e \in E$.

Output: $dist(s, v)$ for every vertex $v \in V$.

The Single-Source Shortest Path Problem

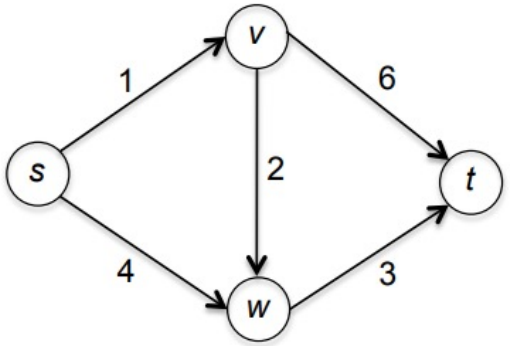
- Recall:
  - The notation dist(s, v) denotes the length of a shortest path from s to v.
  - If there is **no path** at all from s to v, then **dist(s, v) is +∞.**
  - By the length of a path, we mean the **sum** of the lengths of its edges.
  - For instance, in a graph in which every edge has length 1, the length of a path is just the number of edges in it.
  - A shortest path from a vertex v to a vertex w is one with minimum length (among all v-w paths).



The Single-Source Shortest Path Problem

Quiz 9.1

Consider the following input to the single-source shortest path problem, with starting vertex s and with each edge labeled with its length:



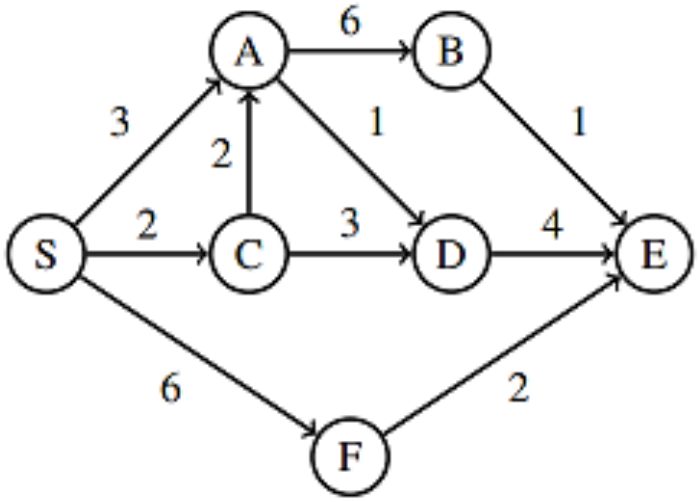
What are the shortest-path distances to s, v, w, and t, respectively?

a) $0, 1, 2, 3$

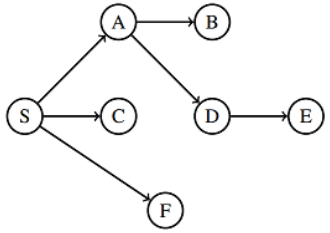
b) $0, 1, 3, 6$

c) $0, 1, 4, 6$

d) $0, 1, 4, 7$

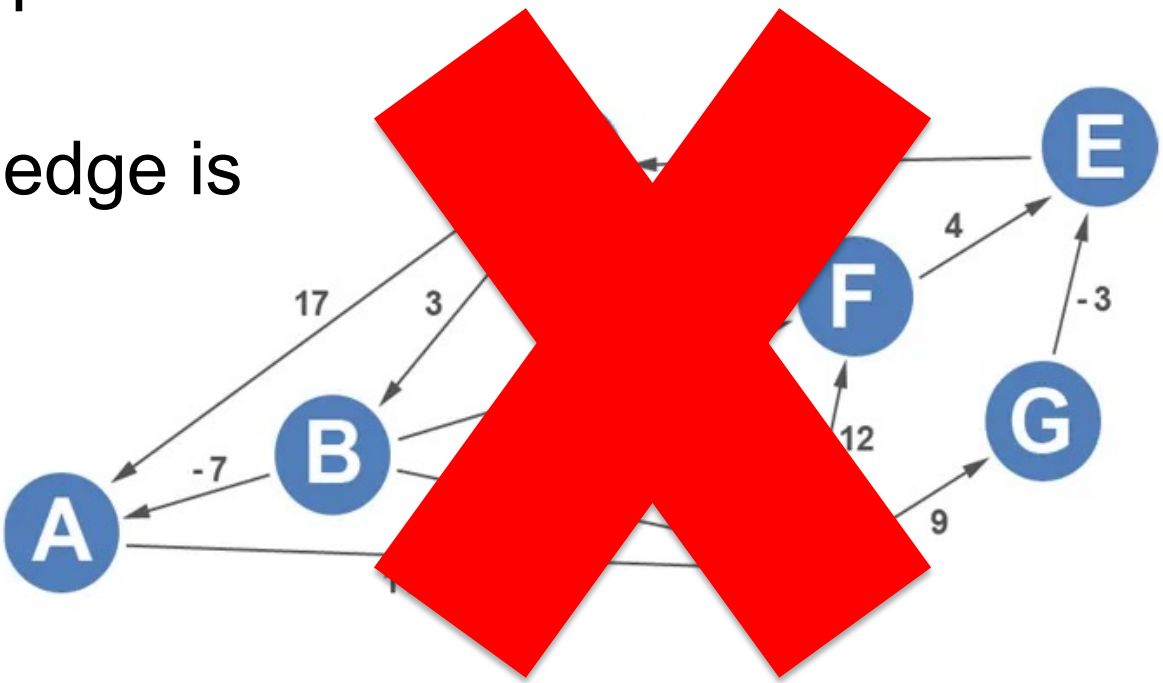


What are the shortest-path distances to S,C,D, and E, respectively?



The Single-Source Shortest Path Problem

- Assumptions:
  - For concreteness, we assume throughout input graph is directed.
  - The length of every edge is nonnegative.

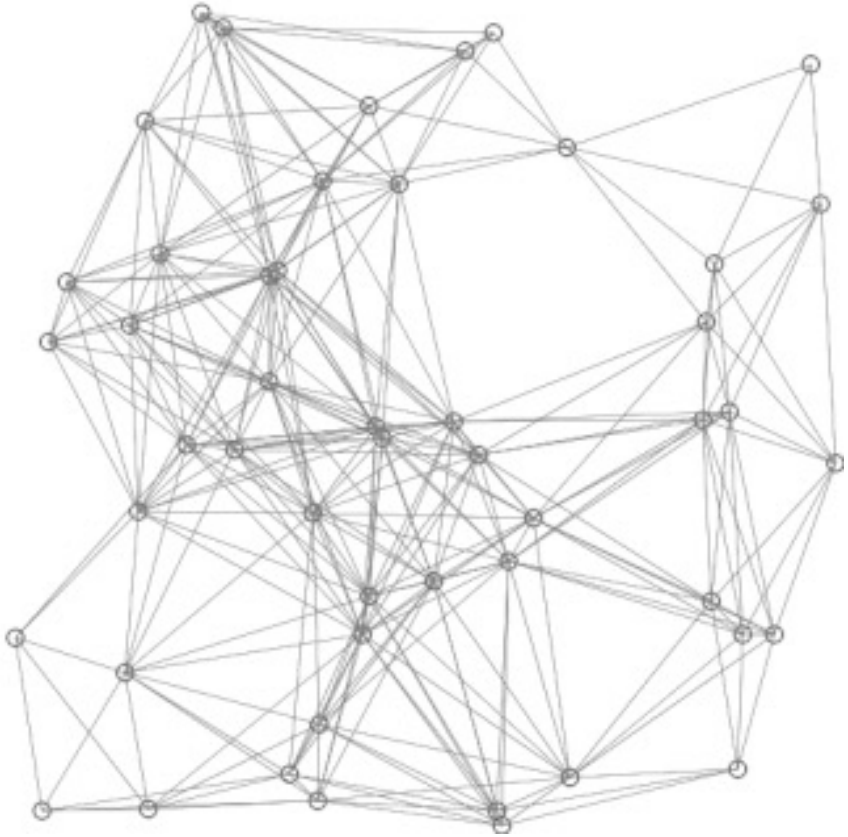


The Single-Source Shortest Path Problem

- **Why Not Breadth-First Search?**
  - Breadth-first search computes the minimum number of edges in a path from the starting vertex to every other vertex.
  - This is the special case of the single-source shortest path problem in which every edge has length 1.
  - With general nonnegative edge lengths, a shortest path need not be a path with the fewest number of edges.

Dijkstra's Algorithm

- Dijkstra's is an algorithm for finding the shortest paths between nodes in a weighted graph.
- Discovered by Edsger W. Dijkstra in 1956 ("in about twenty minutes," he said in an interview many years later).
- Each iteration of its main loop processes **one** new vertex.
- The algorithm's sophistication lies in its clever rule for selecting which vertex to process next: the not-yet-processed vertex that appears to be closest to the starting vertex



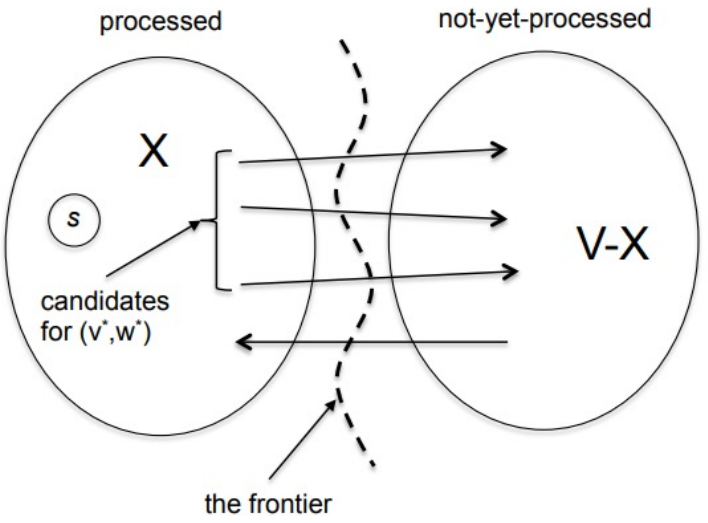
Dijkstra's Algorithm

Dijkstra

Input: directed graph $G = (V, E)$ in adjacency-list representation, a vertex $s \in V$, a length $\ell_e \geq 0$ for each $e \in E$.

Postcondition: for every vertex v, the value $len(v)$ equals the true shortest-path distance $dist(s, v)$.

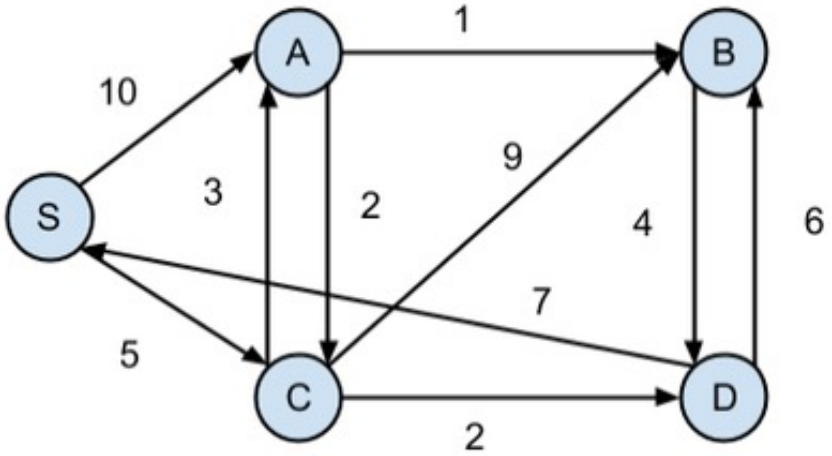
```
// Initialization
1 X := {s}
2 len(s) := 0, len(v) := +∞ for every v ≠ s
// Main loop
3 while there is an edge (v, w) with v ∈ X, w ∉ X do
4     (v*, w*) := such an edge minimizing len(v) + ℓ_vw
5     add w* to X
6     len(w*) := len(v*) + ℓ_v*w*
```

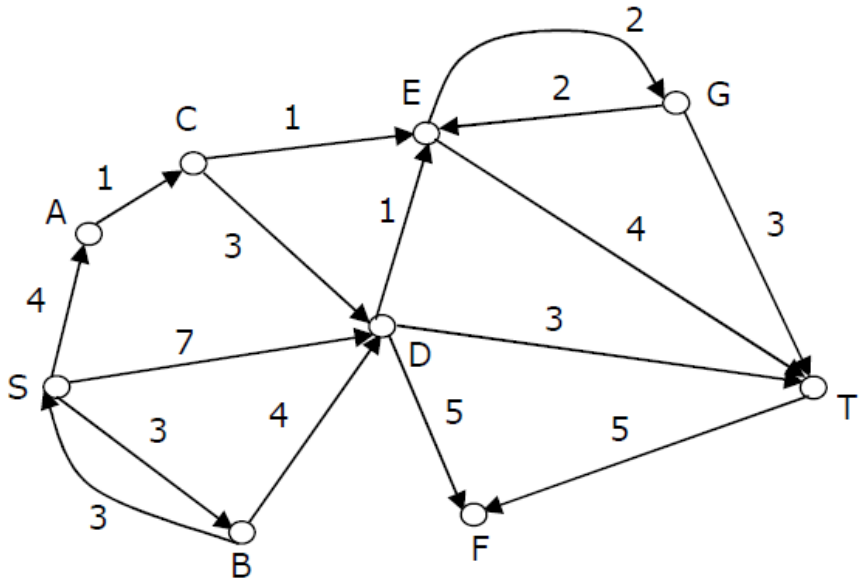


Dijkstra score

Dijkstra's Algorithm

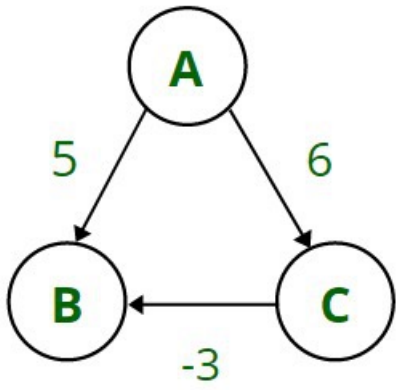
- Examples

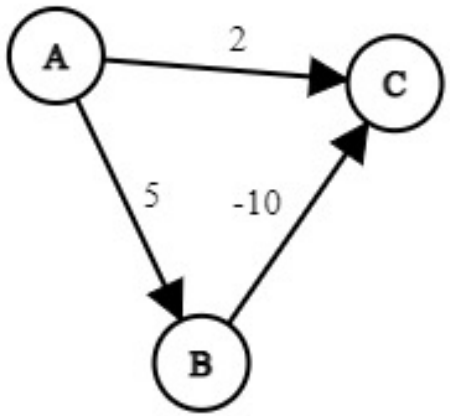




Dijkstra's Algorithm

- Dijkstra's algorithm applies equally well to undirected graphs after small cosmetic changes
- You should not use Dijkstra's algorithm in applications with negative edge lengths. **Why?**





Dijkstra's Algorithm

Problem 9.1 Consider a directed graph G with distinct and non-negative edge lengths. Let s be a starting vertex and t a destination vertex, and assume that G has at least one s-t path. Which of the following statements are true? (Choose all that apply.)

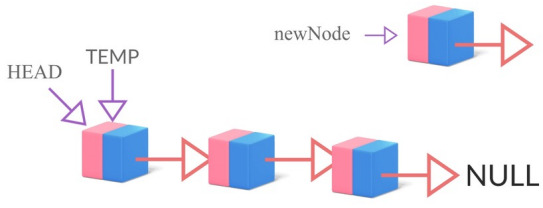
a) The shortest (meaning minimum-length) s-t path might have as many as $n - 1$ edges, where n is the number of vertices.

b) There is a shortest s-t path with no repeated vertices (that is, with no loops).

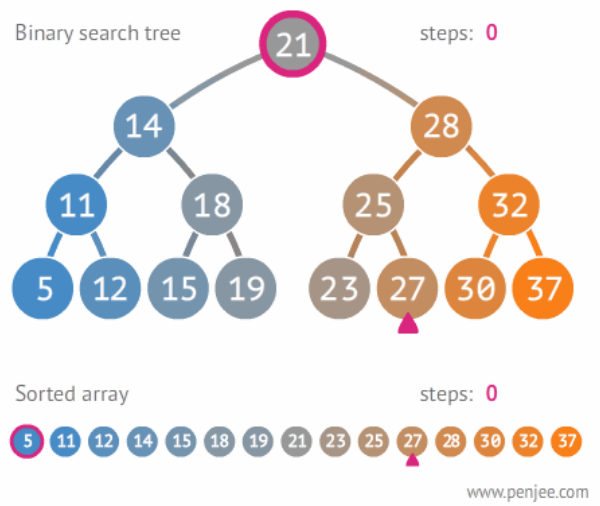
c) The shortest s-t path must include the minimum-length edge of G.

d) The shortest s-t path must exclude the maximum-length edge of G.

Data Structure



- The reason of using a data structure is to **organize** data so you can access it quickly and usefully.
  - The queue data structure, used in our linear-time implementation of breadth-first search
  - The stack data structure, which was crucial in our iterative implementation of depth-first search

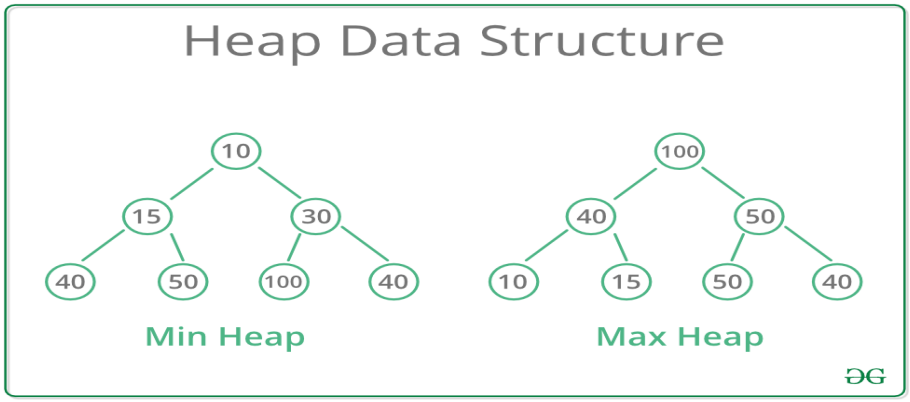


Data Structure

- What are the pros and cons of different data structures, and how should you choose which one to use in a program?
  - In general, the **more operations** a data structure supports, the **slower** the operations and the **greater** the space overhead

- **Principle of Parsimony:** Choose the simplest data structure that supports all the operations required by your application.
- Cleverly organizing your data can dramatically improve a program's running time

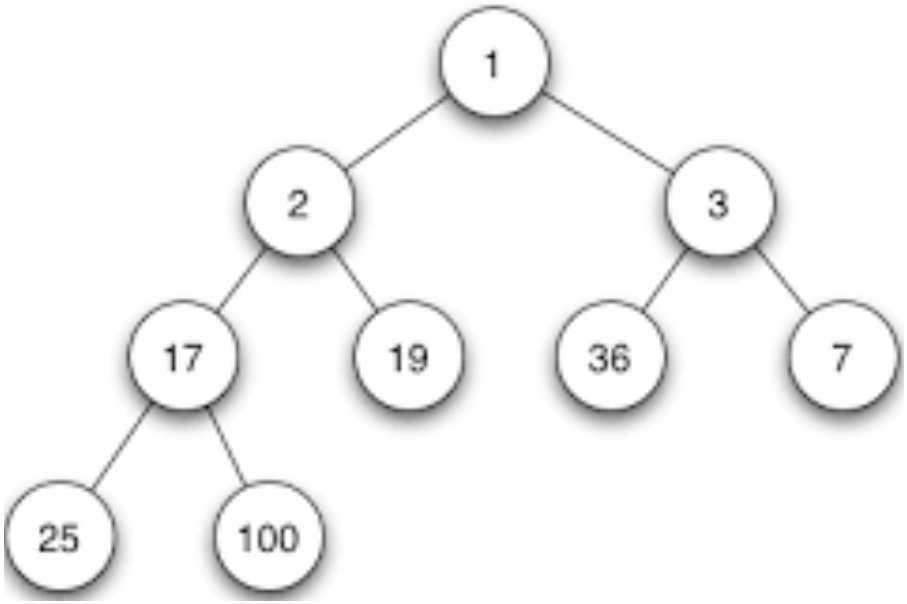
Heap



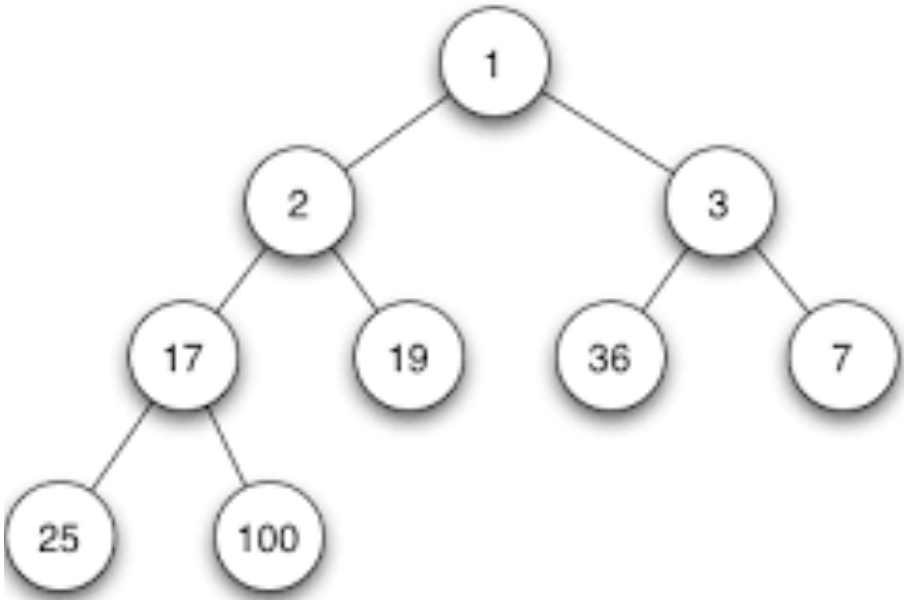
- A **heap** is a data structure that keeps track of an evolving set of objects with keys and can quickly identify the object with the smallest key.
- Examples:
  - objects might correspond to employee records, with keys equal to their identification numbers.
  - They might be the edges of a graph, with keys corresponding to edge lengths.
  - Events scheduled for the future, with each key indicating the time at which the event will occur

Heap

- Basic Operations
  - Insert: given a heap H and a new object x, add x to H.
  - ExtractMin: given a heap H, remove and return from H an object with the smallest key (or a pointer to it).



Heap



- Other operations:
  - FindMin: given a heap H, return an object with the smallest key (or a pointer to it).
  - Heapify: given objects x1, . . . ,xn, create a heap containing them.
  - Delete: given a heap H and a pointer to an object x in H, delete x from H.

Heap

Operation	Running time
INSERT	$O(\log n)$
EXTRACTMIN	$O(\log n)$
FINDMIN	$O(1)$
HEAPIFY	$O(n)$
DELETE	$O(\log n)$

- **When to Use a Heap?**
  - If your application requires fast minimum (or maximum) computations on a dynamically changing set of objects, the heap is usually the data structure of choice.

Heaps

- Add the following elements to a Minheap:

  15, 100, 67, 45, 12, 40, 7, 30, 4

Heap (Applications)

- Sorting:

HeapSort

Input: array A of n distinct integers.
Output: array B with the same integers, sorted from smallest to largest.

$H :=$ empty heap
for $i = 1$ to n **do**
    INSERT $A[i]$ into H
for $i = 1$ to n **do**
    $B[i] :=$ EXTRACTMIN from H

Quiz 10.1

What's the running time of `HeapSort`, as a function of the length n of the input array?

a) $O(n)$

b) $O(n \log n)$

c) $O(n^2)$

d) $O(n^2 \log n)$

Speeding Up Dijkstra's Algorithm

- The concrete plan is to store the as-yet-unprocessed vertices (V – X in the Dijkstra pseudocode) in a heap, while maintaining the following invariant.

Invariant

The key of a vertex $w \in V - X$ is the minimum Dijkstra score of an edge with tail $v \in X$ and head w, or $+\infty$ if no such edge exists.

$$key(w) = \min_{(v,w) \in E \,:\, v \in X} \underbrace{len(v) + \ell_{vw}}_{\text{Dijkstra score}}$$

Speeding Up Dijkstra's Algorithm

Dijkstra (Heap-Based, Part 1)

Input: directed graph $G = (V, E)$ in adjacency-list representation, a vertex $s \in V$, a length $\ell_e \geq 0$ for each $e \in E$.

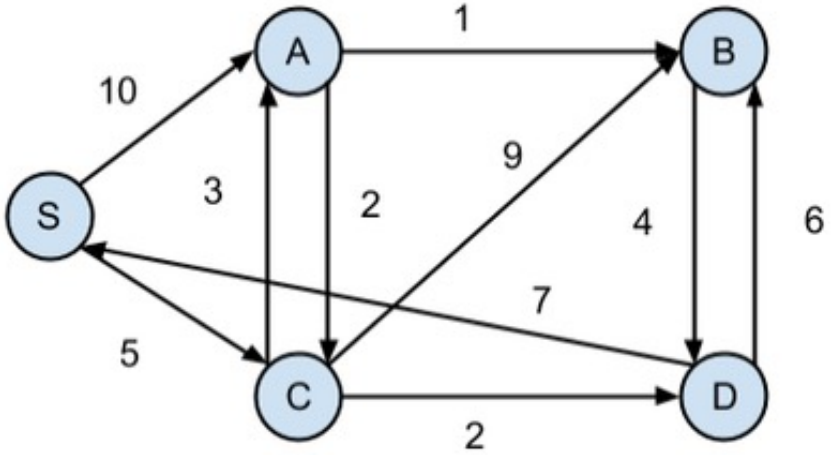
Postcondition: for every vertex v, the value $len(v)$ equals the true shortest-path distance $dist(s, v)$.

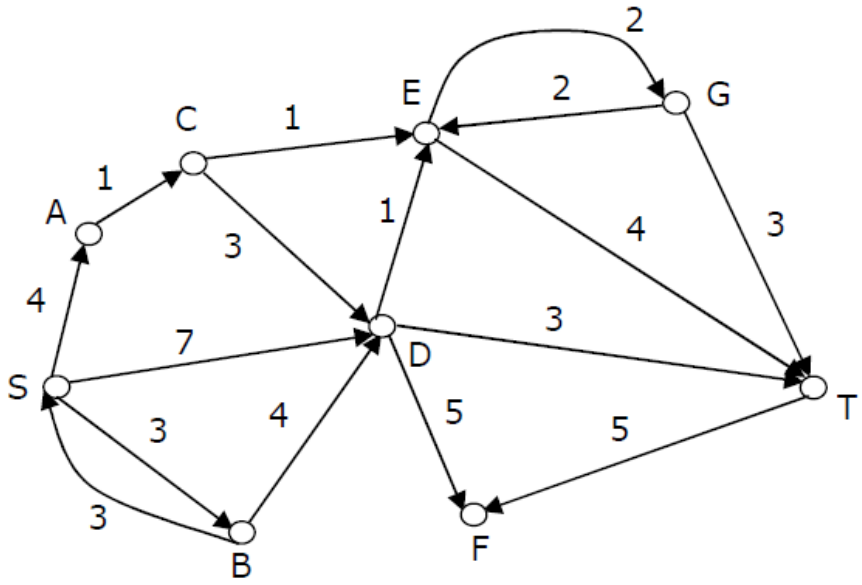
```
   // Initialization
 1 X := empty set, H := empty heap
 2 key(s) := 0
 3 for every v ≠ s do
 4     key(v) := +∞
 5 for every v ∈ V do
 6     INSERT v into H              // or use HEAPIFY
   // Main loop
 7 while H is non-empty do
 8     w* := EXTRACTMIN(H)
 9     add w* to X
10     len(w*) := key(w*)
       // update heap to maintain invariant
11     (to be announced)
```

Dijkstra (Heap-Based, Part 2)

```
       // update heap to maintain invariant
12     for every edge (w*, y) do
13         DELETE y from H
14         key(y) := min{key(y), len(w*) + ℓ_{w*y}}
15         INSERT y into H
```

Speeding Up Dijkstra's Algorithm





Running Time

The overall running time is O((m+n) log n)