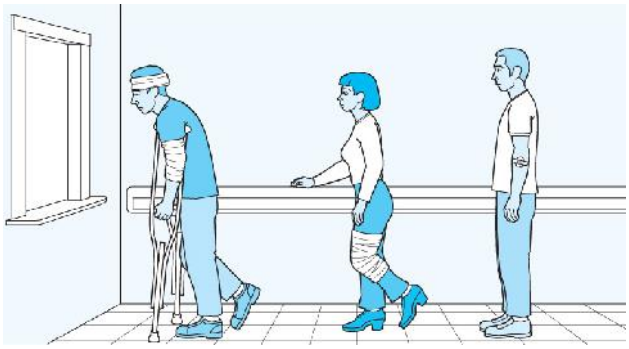
Chapter 9

Priority Queues, Heaps, Graphs

Priority Queue



Priority Queue
An ADT in which only the item with the highest priority can be accessed

Priority

- Depends on the Application
  - Telephone Answering System
    - The length of waiting (FIFO queue)
  - Airline Check-In System
    - Travel Class
  - Hospital Emergency Room
    - Severeness of Injury

Implementation Level

- How items are enqueued ?
- How items are dequeued ?
- How to sort the items ?
  - Based on their priorities
  - ComparedTo( ) function

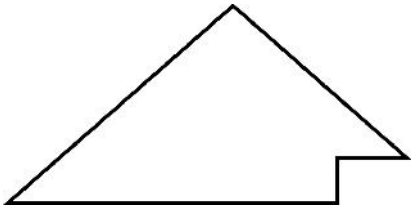
Implementation Level

- There are many ways to implement a priority queue
  - **An unsorted List-** dequeuing would require searching through the entire list
  - **An Array-Based Sorted List-** Enqueuing is expensive
  - **A Linked Sorted List-** Enqueuing again is 0(N)
  - **A Binary Search Tree-** On average, $0(\log_2 N)$ steps for both enqueue and dequeue
  - ***Any other choice?***

Heaps

•Heap

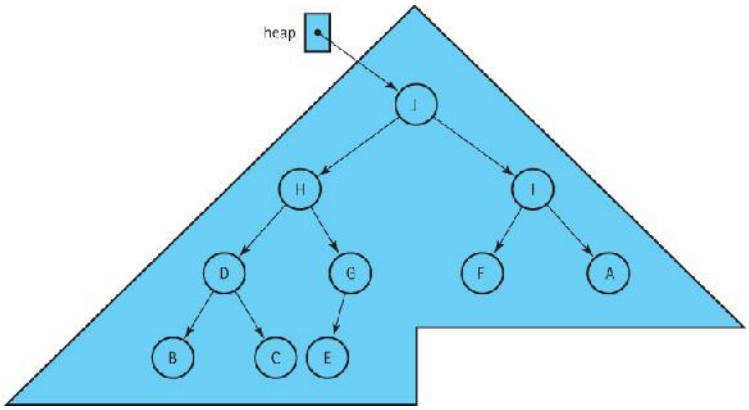
A ***complete binary tree***, each of whose elements contains a value that is ***greater than or equal to the value*** of each of its children



Complete tree

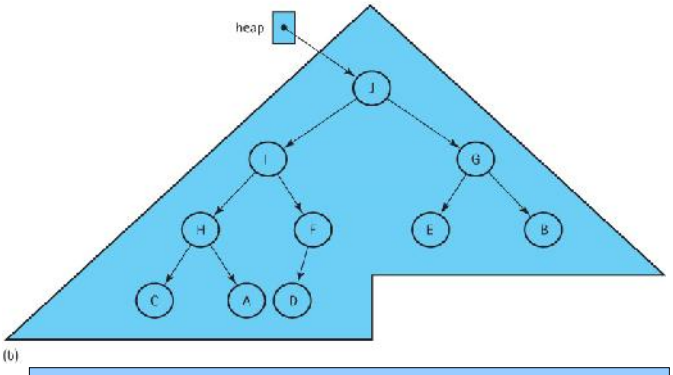
Remember?

Heaps



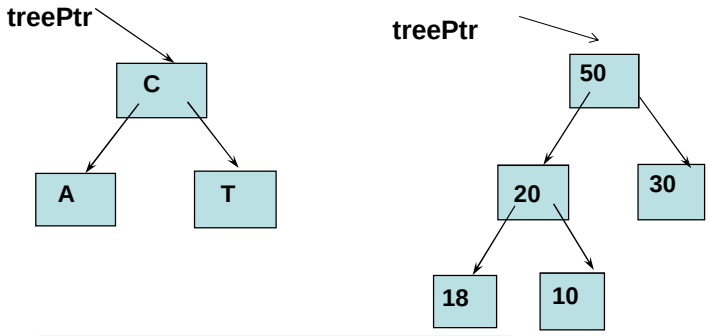
Heap with letters 'A' .. 'J'

Heaps



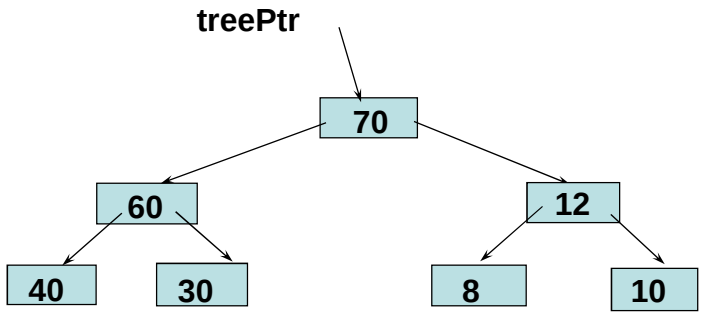
Another heap with letters 'A' .. 'J'

Heaps



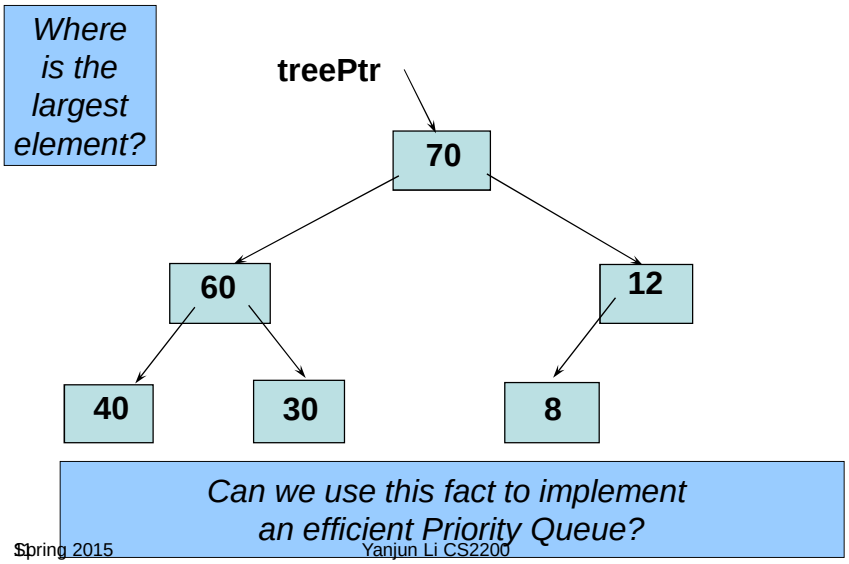
Are these heaps?
Shape & Order Properties

Heaps



Is this a heap? Shape & Order Properties

Heaps

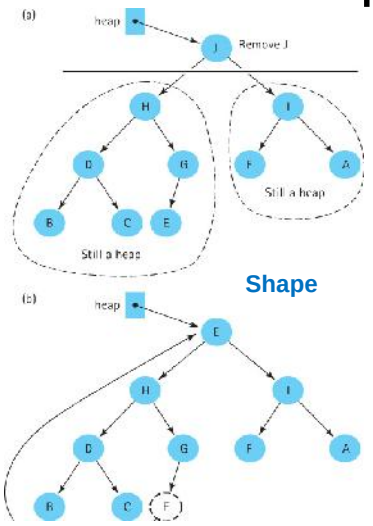


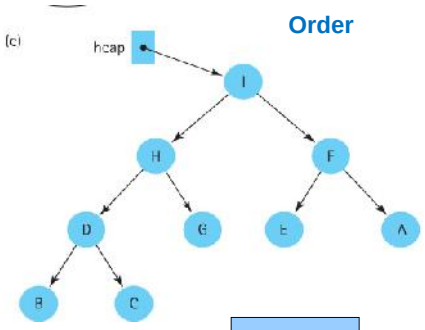
Heaps

- Heap is good for Priority Queue.
- We have immediate access to highest priority item BUT if we remove it, the structure isn't a heap

- *Can we "fix" the problem efficiently?*

Heaps



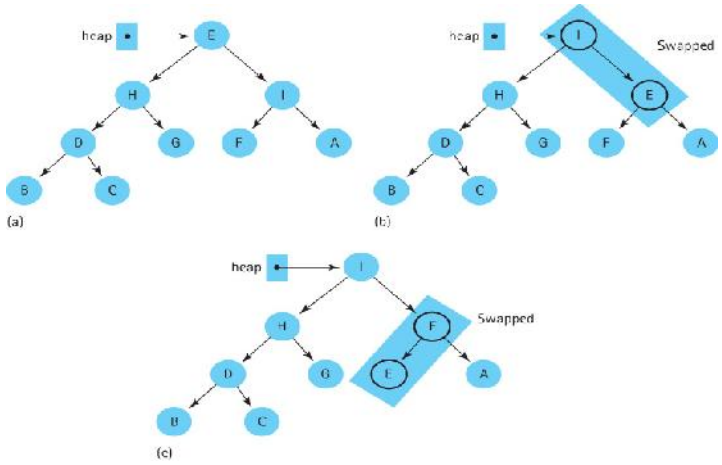


How?

ReheapDown( )

- Function: Restores the order property of heaps to the tree between root and bottom.
- Precondition: *The order property of heaps may be violated only by the root node of the tree.*
- Postcondition: The order property applies to all elements of the heap.

Heaps



ReheapDown()

If the root is not a leaf node

If the value of the root is smaller than its ***largest*** *child*

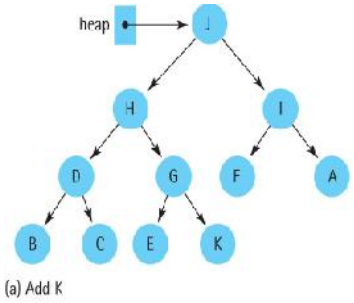
Swap the value of the root with the value of this child.

ReheapDown the subtree with the new root value.

- Why just compare the root with its children? How about the rest of the tree?
- Base Cases
  - The root of the current subtree is a leaf node.
  - The value of the root is greater or equal to the values in both its children.

Heaps

Add 'K': Where must the new node be put?

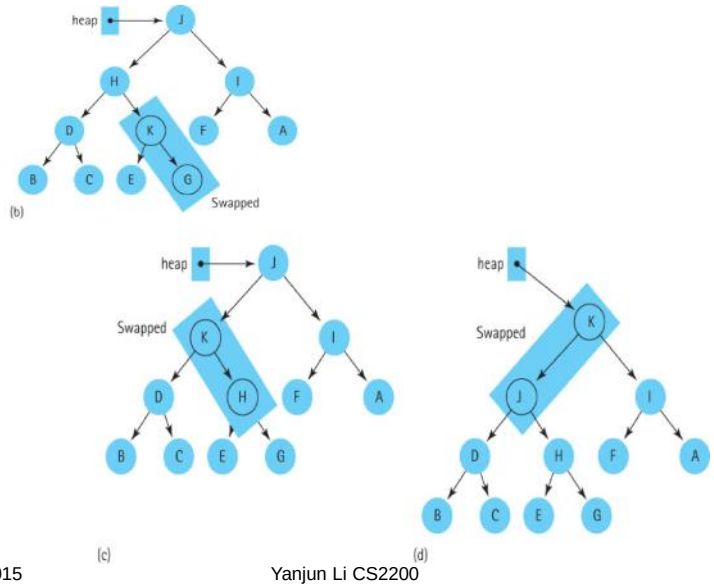


(a) Add K

Shape Property is kept, but Order Property?

ReheapUp

- Function: Restores the order property to the heap between root and bottom.
- Precondition: The order property is satisfied from the root of the heap through the next-to-last leaf node; *the last (bottom) leaf node may violate the order property.*
- Postcondition: The order property applies to all elements of the heap from root through bottom.



ReheapUp()

If the root is not reached

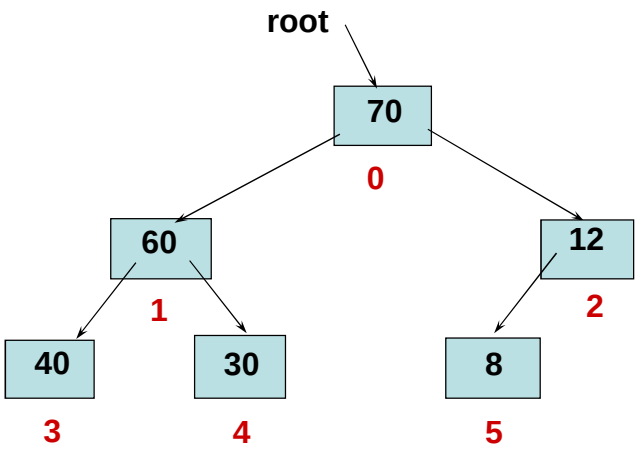
If the value of the bottom node is larger than the value of its parent

Swap the value of the bottom node with the value of its parent

ReheapUp the subtree with the new bottom node.

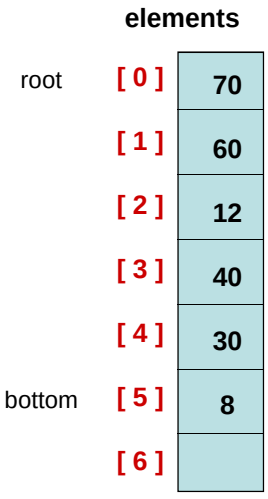
- Why just compare the root with its parent? How about the rest of the tree?
- Base Cases
  - Reach the root of the tree
  - The value of the bottom node is smaller or equal to the values of its parent.

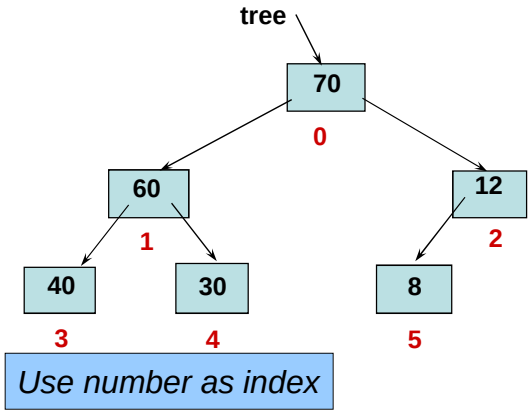
Heaps



Number nodes left to right by level

Implementation of Heaps





Use number as index

Why use array, not linked list?

Review

- For any node **`tree.nodes[index]`**
  its left child is in **`tree.nodes[index*2 + 1]`**
  right child is in **`tree.nodes[index*2 + 2]`**
  its parent is in **`tree.nodes[(index - 1)/2]`**

Heaps

```
struct HeapType
{
  void ReheapDown(int root, int bottom);
  void ReheapUp(int root, int bottom);
  ItemType* elements;  // Dynamic array
  int numElements;
};
```

Recursive ReheapDown

```
void HeapType::ReheapDown(int root, int bottom)
{
   int  maxChild,  rightChild, leftChild ;

    leftChild  =  root * 2 + 1 ;
    rightChild  =  root * 2 + 2 ;
    if ( leftChild  < =  bottom)        //root is not a leaf node
    {    //find the largest child
         if (leftChild == bottom)     // only one child
                maxChild = leftChild;
         else {                                    //find the largest child
                if (elements[leftChild] .ComparedTo(elements[rightChild])!= GREATER)
                     maxChild  =  rightChild;
                else      maxChild  =  leftChild;
         }
         if (elements[root] .ComparedTo(elements[maxChild])== LESS)   //violate the order property
         {
              Swap(elements[root], elements[maxChild]);
              ReheapDown(maxChild, bottom);
          }
     }
}
```


Recursive ReheapUp

```
void HeapType::ReheapUp(int root,  int bottom)
{
  int parent;
  if (bottom > root)  //not reach the root
  {
    parent = ( bottom - 1 ) / 2;
    //violate the order property
    if (elements[parent].ComparedTo(elements[bottom])== LESS)
    {
      Swap(elements[parent], elements[bottom]);
      ReheapUp(root, parent);
     }
  }
}
```

Heaps/Priority Queues

- *How can heaps be used to implement Priority Queues?*
  - *Enqueue()*
    - *Insert a node as the right-most leaf node*
    - *Apply the order property by calling ReheapUp()*
  - *Dequeue()*
    - *Remove the root node*
    - *Move the right-most node to the root position*
    - *Apply the order property by calling ReheapDown()*

Why Heap is Good

- Heap's Order Property:
  - Root has the largest value / highest priority
- Heap's Shape Property:
  - Min. number of levels with N nodes of a binary tree: $log_2N +1$
  - A heap guarantees $0(\log_2 N)$ steps, even in the worst case
    - A complete tree is of minimum height.
    - At most log_2N levels exist above the leaf node.
    - At most log_2N levels exist below the root.

Comparison of Priority Queue Implementations

	Enqueue	*Dequeue*
Heap	**O($\log_2 N$)**	**O($\log_2 N$)**
Linked List	**O(*N*)**	**O(*N*)**
Binary Search Tree		
Balanced	**O($\log_2 N$)**	**O($\log_2 N$)**
Skewed	**O(*N*)**	**O(*N*)**

Graphs

Graph

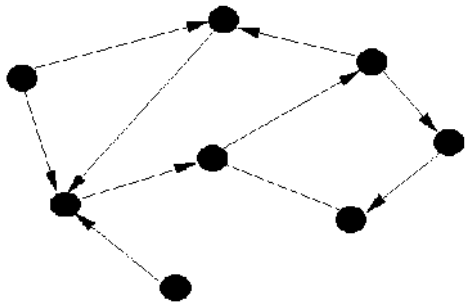
A data structure that consists of a set of nodes and a set of edges that relate the nodes to each other

Vertex

A node in a graph

Edge (arc)

A connection betweer two nodes in a graph, represented by a pair of vertices.



Graphs

Undirected graph

A graph in which the edges have no direction

Directed graph (digraph)

A graph in which each edge is directed from one vertex to another (or the same) vertex

Graphs

Formally a graph G is defined as follows

G = (V,E)

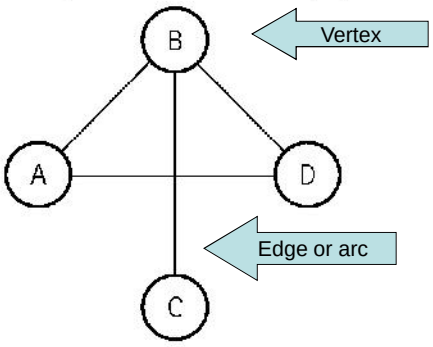
where

V(G) is a finite, nonempty set of vertices

E(G) is a set of edges (written as pairs of vertices)

Graphs

(a) Graph1 is an undirected graph.

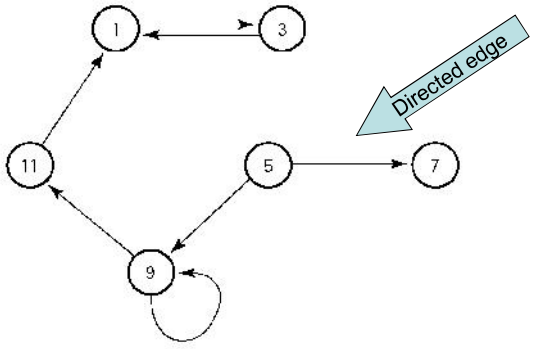


V(Graph1) = {A, B, C, D}
E(Graph1) = {(A, B), (A, D), (B, C), (B, D)}

Undirected Graphs have no arrows on the edges

Graphs

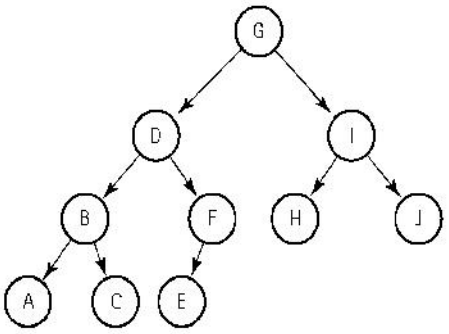
(b) Graph2 is a directed graph.



V(Graph2) = {1, 3, 5, 7, 9, 11}
E(Graph2) = {(1, 3), (3, 1), (5, 7), (5, 9), (9, 11), (9, 9), (11, 1)}

Graphs

(c) Graph3 is a directed graph.



V(Graph3) = {A, B, C, D, E, F, G, H, I, J}
E(Graph3) = {(G, D), (G, I), (D, B), (D, F), (I, H), (I, J), (B, A), (B, C), (F, E)}

What other structure is this ?
A tree is a acyclic graph.

Graphs

Adjacent vertices

Two vertices in a graph that are connected by an edge

Path

A sequence of vertices that connects two nodes in a graph

Complete graph

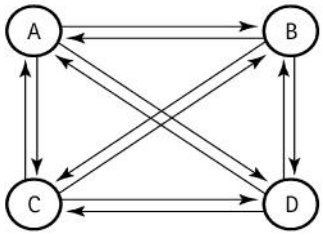
A graph in which every vertex is directly connected to every other vertex; For a graph with N nodes,

- N*(N-1) edges in a complete directed graph
- N*(N-1)/2 edges in a complete undirected graph

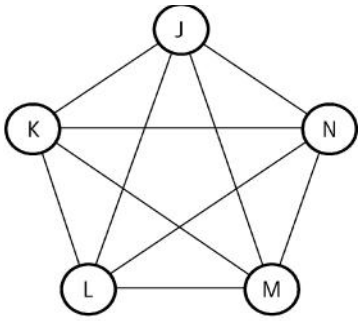
Weighted graph

A graph in which each edge carries a value

Graphs



(a) Complete directed graph.

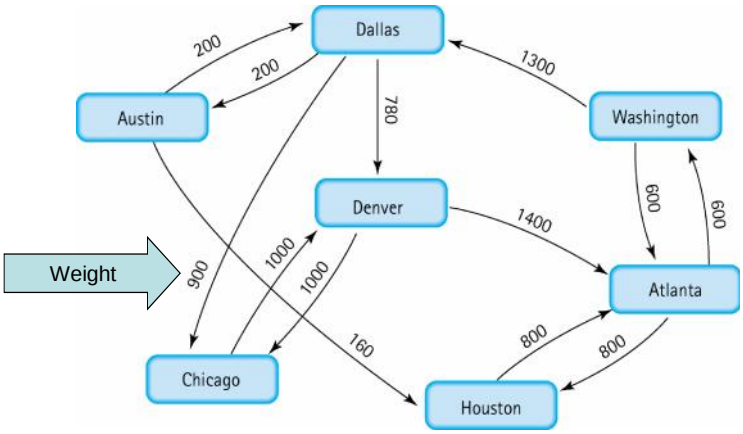


(b) Complete undirected graph.

How many edges in a directed graph with N vertices?

How many edges in an undirected graph with N vertices?

Graphs



Array-Based Implementation

Adjacency Matrix

For a graph with N nodes, an N by N table that shows the existence (and weights) of all edges in the graph

Mapping Array

An array that maps vertex names into array indexes

Marking

Associate a Boolean variable with each vertex: true if visited, false if not yet visited

Adjacency Matrix for Flight Connections

graph
.num Vertices 7
.vertices

[0]	"Atlanta"
[1]	"Austin"
[2]	"Chicago"
[3]	"Dallas"
[4]	"Denver"
[5]	"Houston"
[6]	"Washington"
[7]	
[8]	
[9]	

.edges

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[0]	0	0	0	0	0	800	600	•	•	•
[1]	0	0	0	200	0	160	0	•	•	•
[2]	0	0	0	0	1000	0	0	•	•	•
[3]	0	200	900	0	780	0	0	•	•	•
[4]	1400	0	1000	0	0	0	0	•	•	•
[5]	800	0	0	0	0	0	0	•	•	•
[6]	600	0	0	1300	0	0	0	•	•	•
[7]	•	•	•	•	•	•	•	•	•	•
[8]	•	•	•	•	•	•	•	•	•	•
[9]	•	•	•	•	•	•	•	•	•	•

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

(Array positions marked '•' are undefined)

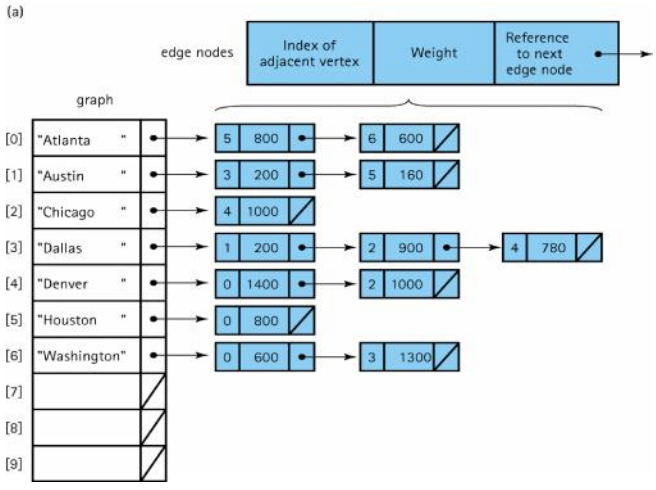
Where could marking variable be kept?

Linked Implementation

Adjacency List

A linked list that identifies all the vertices to which a particular vertex is connected; each vertex has its own adjacency list

Adjacency List Representation of Graphs



Where could a marking variable go ?

Graph Algorithms

Breadth-first search algorithm

Visit all the nodes on one level before going to the next level

Depth-first search algorithm

Visit all the nodes in a branch to its deepest point before moving up

Single-source shortest-path algorithm

An algorithm that displays the shortest path from a designated starting node to every other node in the graph

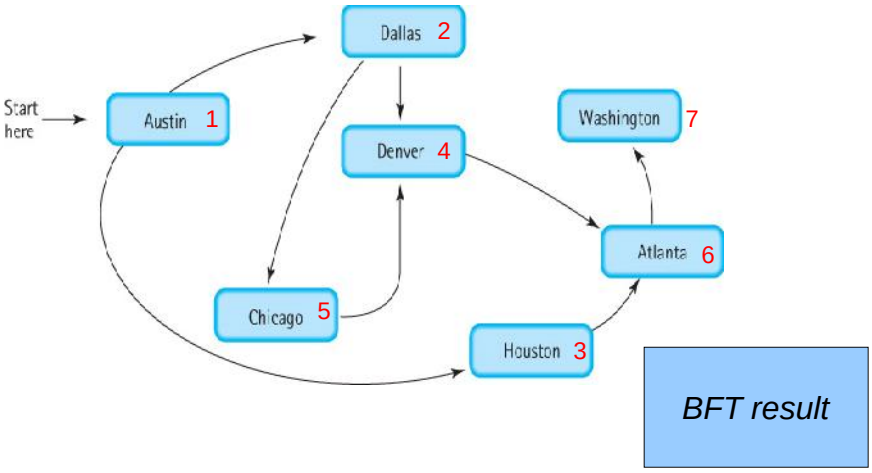
Graph Traversal

- Visits all the vertices, beginning with a specified start vertex.
- No vertex is visited more than once, and vertices are only visited if they can be reached – that is, if there is a path from the start vertex.

Breadth-First Traversal with Queue

- “Neighbors-First”.
- A queue data structure is needed. It holds a list of vertices which have not been visited yet but which should be visited soon.
- While visit a vertex involves, adding its neighbors to the queue. Neighbors are not added to the queue if they are already in the queue, or have already been visited.
- Queue : FIFO.

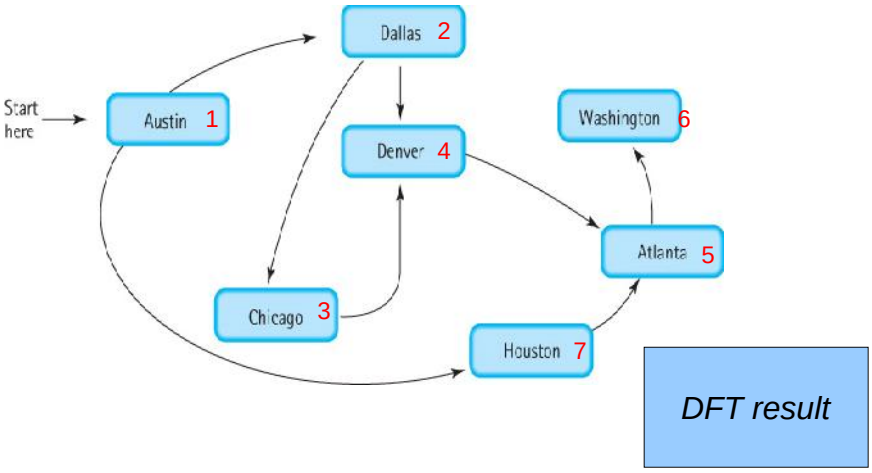
Graphs



Depth-First Traversal with Stack

- “Neighbors-First”.
- A stack data structure is needed. It holds a list of vertices which have not been visited yet but which should be visited soon.
- While visit a vertex involves, adding its neighbors to the stack. Neighbors are not added to the stack if they are already in the stack, or have already been visited.
- Stack: LIFO.

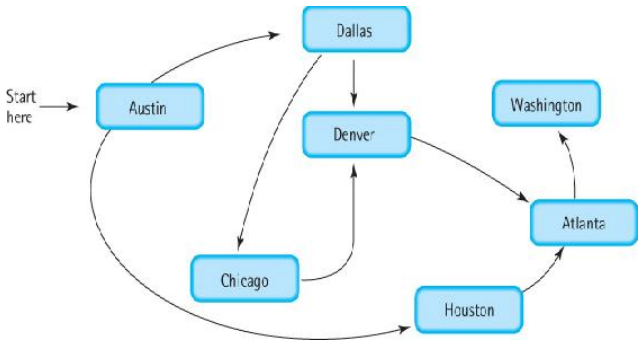
Graphs



Breadth-First vs. Depth-First

- Breadth-first Traversal: *all one-flight solutions, then all two-flight solutions, etc.*

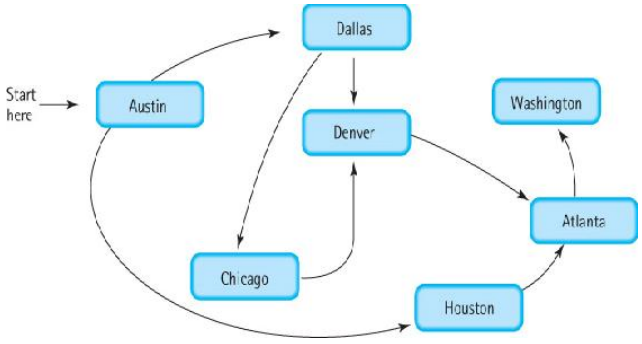
Austin
Dallas
Houston
Denver
Chicago
Atlanta
Washington



Breadth-First vs. Depth-First

- Depth-first Traversal: *keep going forward in one direction; backtrack only when reach a dead end.*

Austin
Dallas
Chicago
Denver
Atlanta
Washington
Houston



Single-Source Shortest-Path

- Shortest-Path
  - A **path** through the graph is a sequence (*v*1*, ..., vn*) such that the graph contains an edge *e*1 going from *v*1 to *v*2, an edge *e*2 going from *v*2 to *v*3, and so on.
  - The path whose weights, when added together, have the smallest sum.
- If the weight is the distance between two cities
  - The shortest path between two cities is the route which has the minimum travelling distance.
- If the weight is the travelling time between two cities
  - The shortest path between two cities is the route which has the shortest travelling time.

Applications of Shortest Path Traversal

- Google Map finds the route with the shortest travelling time.
- Networking Routing Protocols
- Game

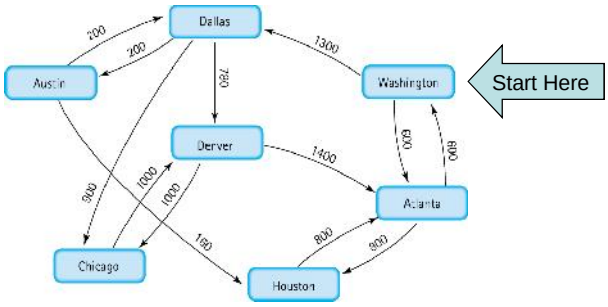
Dijkstra's Algorithm

- Similar to Breath-First Traversal
- A FIFO queue for unvisited neighbors of a visited vertex.
- A priority queue for the possible path
  - Each item of the priority queue has three data members
    - fromVertex: last visited vertex on the path from the starting vertex
    - toVertex: next visiting vertex
    - Distance : the min. distance from the starting vertex to next visiting vertex.

Dijkstra's Algorithm

- The priority-queue implemented with a minimum heap
- Starts from the Starting vertex, the first edge is itself (distance is 0).
- For each visited vertex, put its unvisited neighbors into FIFO queue.
- Dequeue the FIFO queue and put the possible path from the starting vertex to the neighbor into the priority queue.
- Dequeue the priority queue one by one.

Graphs



Washington -> Atlanta	**600**
Washington ->Dallas	**1300**
Washington -> Atlanta -> Houston	**1400**
Washington -> Dallas -> Austin	**1500**
Washington -> Dallas -> Denver	**2080**
Washington -> Dallas -> Chicago	**2200**

Reference

- Reproduced from C++ Plus Data Structures, 4th edition by Nell Dale**.**
- **Reproduced by permission of Jones and Bartlett Publishers International.**