# 241 Exam Review

## Algorithms, recursion and algorithmic analysis

What is an algorithm?

Algorithm is a description of a series basic steps which achieve a specified result.

How do we describe it?

Plain English, Pseudocode, computer program implementation.

What (and why) is recursion?

Who is the big-O? What does he mean?

It is a representation of the time efficiency of some algorithm.

Definition: we write f(n) = O(g(n)) if, for some Constant C, f(n) <= Cg(n) for ALL sufficiently large C.

Common efficiency analyses: nested loop, divide and conquer!

## Array, sorting and searching

Using subarrays (particularly in recursive method for array processing)

Searching in unsorted array and in sorted array

Sorting method (selection, insertion, Heap sort)

Selection sort:

Pick the 1st smallest element at the first position

Pick the 2ed smallest element at the second position

…

Pick the Nth element at the Nth position

Insertion sort:

The insertion algorithm sorts an array of values by repetitively inserting a particular value into a subset of the array which is already sorted.

Heap sort (see Heap section)

## Random number generator and their use

Why are random numbers important in computation?

What is the difference between truly random and pseudo-random numbers?

Truly random number come from by observing unpredictable physical process. Pseudo-random numbers are generated mathematically, even they appear to be random, but if we know the seed and the algorithm they use, then the random number can be fully predicted.

How are pseudo-random numbers generated?

S <- (s \* a + b) % c

How can we pick a winner? Shuffle a deck? Choose from a collection of know or unknown size? What are some of the efficiency issue?

## ADT principles, data structures

What are the advantages of using ADTs?

ADT is a data type whose intended use is specified by its interface, so this abstraction can hide the detail implementation from user, and user can focus on important issue.

What is the relationship between ADT and data structure?

A data structure is the collection of programming structure used to implement an ADT.

Common ADTs: stack, list, queue – similarities and differences

## Stack and queue ADTs, linked data structures

What is a stack? What is a queue? How are they similar? How are they different?

How are lined data structures implemented in Java?

It is implement with recursive data structure in which, one of the attribute of the class, its type is the class itself.

## Tree:

* A balanced n-ary tree with m elements will have a height of lognm.
* A full balanced binary tree with depth = d, can have at most 2^[d+1] – 1 nodes and at least 2^[d] nodes.

If the binary tree is poor balanced, then it can have at least d + 1 nodes.

* the meaning of full and complete
* Traversal:

1. Preorder: visit root, then traverse the subtree from left to right.
2. Inorder: travers the left subtree, then root, then the remaining subtrees from left to right.
3. Postorder: travers the subtree from left to right, then visit root.

* Binary Search Tree

1. The definition of BST: A binary search tree is a binary tree that, for each node n, the left subtree of n contains element less than the element stored in n, and the right subtree of n contains element that are greater than or equal to the element stored in n.
2. Adding element into binary search tree
3. If root is empty, then root = element.
4. If element is less then root, then add to left subtree.
5. Else add to right subtree.
6. Deleting node from binary search tree
7. If node has no children, just delete it.
8. If node has only one child, replace the deleted node with its child.
9. If the node has two children, an appropriate node is founded from lower in the tree and used it to replace the node, the children of the removed node become the children of replacement node.

A good choice for the appropriate node is the deleted node’s inorder successor (the next highest value).

1. The height () or depth () of Binary search tree.

## Heap and Priority Queue

* Definition of Heap: A heap is a complete binary tree in which every element is greater than or equal to both of its children.
* A heap keeps the largest value of a set of elements readily available. This is useful in any situation in which the value is the dominant criteria.
* Pseudo-code for adding and removal algorithms.
* Using heaps of sorting, compare with other sorting?

Have the advantage of in place sort (insertion sort) and the O(nlogn) time efficiency (merge sorting).

* Heap operation:

1. Adding an element into the heap

The strategy to add an element is to add the element in the heap as a new leaf, keep the tree complete; then moving the element up toward the root.

1. Finding the maximum value in the heap

It is easy, it is just the root.

1. Remove the maximum value

The strategy to delete maximum value, first delete the root, then reconstruction the tree. During the reconstruction, move the “last leaf” of the tree (rightmost leaf on the last level) to the root position, then move it down through the tree as needed until the relationships among the elements are appropriate.

* Heap sort: A heap sort sorts a set of elements by adding each of them into a heap, then removing them one at a time.
* Two ways of organizing heaps:

1. Top down approach: the efficiency is O (nlogn)
2. Bottom up approach: the efficiency is O (n)

* Priority Queue definition: A priority queue is a collection that follows two rules:

1. Item with higher priority go first
2. Items with same priority use first in, first out method to determine their ordering.

* Priority Queue Implementation:

1. Could be implemented by using a list of queues where each queue represents items of a given priority.
2. Using heap to implement Priority Queue, all we need to do is define the priority on the priority queue node, then apply the heap operation based on priority.

Define the PriorityQueueNode that stores, what is the priority:

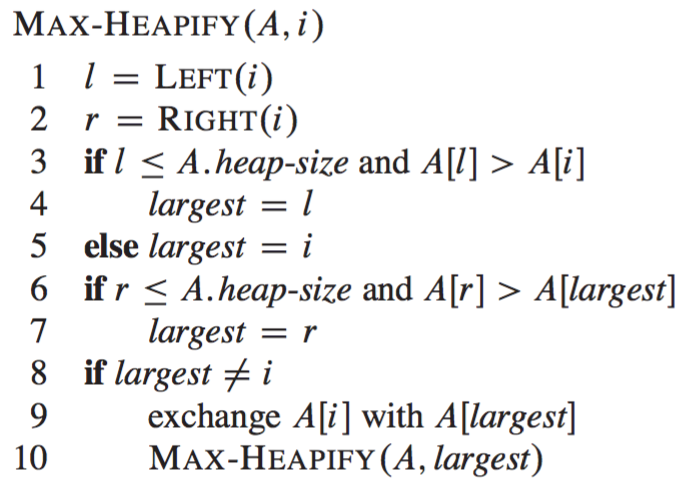
1. the element to be placed on the queue.
2. the priority of the element.
3. the order in which element are placed on the queue.

Then define compareTo () method for PriorityQueueNode class to compare priorities first and then compare the order if there is a tie.

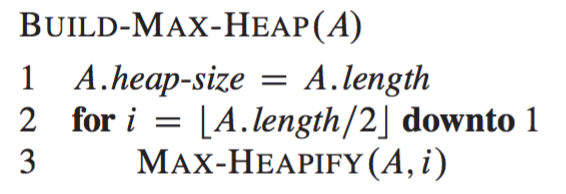
* Implementing heap using array:

A[i] => its ParentIndex = i/2, LeftChildIndex = 2\*i, RightChildIndex = 2\*i + 1

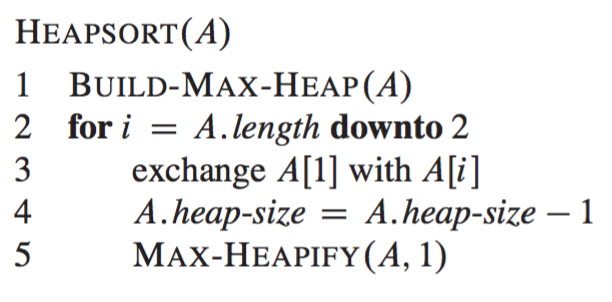
* Core method: Max-Heapify(A, i)



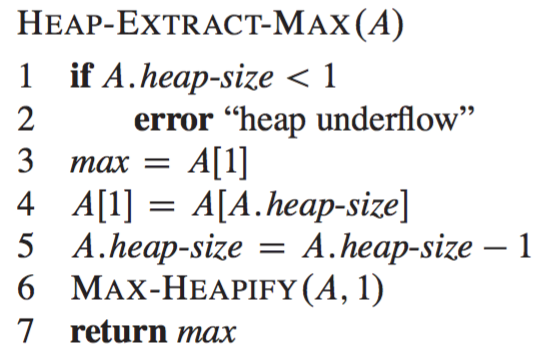
* Build-Max-Heap(A)



* HEAPSORT(A)



* HEAP-EXTRAC-MAX(A)



* MAX-HEAP-INSERT(A, KEY)

A.heap-size = A.heap-size + 1

i = A.heap-size

A[i] = key

While i>1 and A[Parent(i)] < A[i]

Exchange A[i] with A[Parent(i)]

i = Parent(i)

* Two approach to implement heap
  1. Top-down approach

Treat a growing initial sub-array as heap and add element onto it one at a time, and let it float as high as necessary.

* 1. Bottom-up approach

Image there is already a tree structure over the array, and fix the non-leaf node which violate the heap property from bottom to top.

## Graph

* An undirected graph is considered complete if it has the maximum number of edges connecting vertices.
* A path is a sequence of edges which connects two vertices in a graph.
* The length of a path is the number of edges in the path (or the number of vertices – 1).
* An undirected graph considered to be connected if for any two vertices there is a path between them.
* A cycle is a path in which the first and last vertices are the same and none of the edges are repeated.
* An undirected tree is a connected, acyclic, undirected graph with one element designated as root.
* **The traversal of graph**

1. Breadth first

Use a queue to manage the traversal nodes, and use an iterator to build the result

1. Enqueue the starting vertex into the queue and mark the enqueued vertex as visited.

Begin a loop that will continue until the queue is empty.

1. Within this loop, dequeue a vertex from the queue, and add that vertex into the iterator.
2. Enqueue each of the vertices that are adjacent to the current one, and have not already been marked as visited, into the queue. And mark they as visited vertices.

After the loop, the iterator contains the vertices in breath-first order from the giving starting vertex.

1. Depth first

Similar to breath first search with two differences:

1. Replace the queue with stack
2. Mark the vertex as visited not when we push it into stack, but when it has been added into the iterator.

* Testing for connectivity

A graph is connected if and only if for each vertex v in a graph containing n vertices, the size of the result of a breadth-first traversal starting at v is n.

* Minimum Spanning Tree
* Determining the shortest path
* Distance Algorithm: Determine the distance from a given vertex to any other vertices

**d[home] <- 0;**

**enqueue home**

**while queue is not empty do:**

**v <- dequeue**

**for w, each of the neighbors of v:**

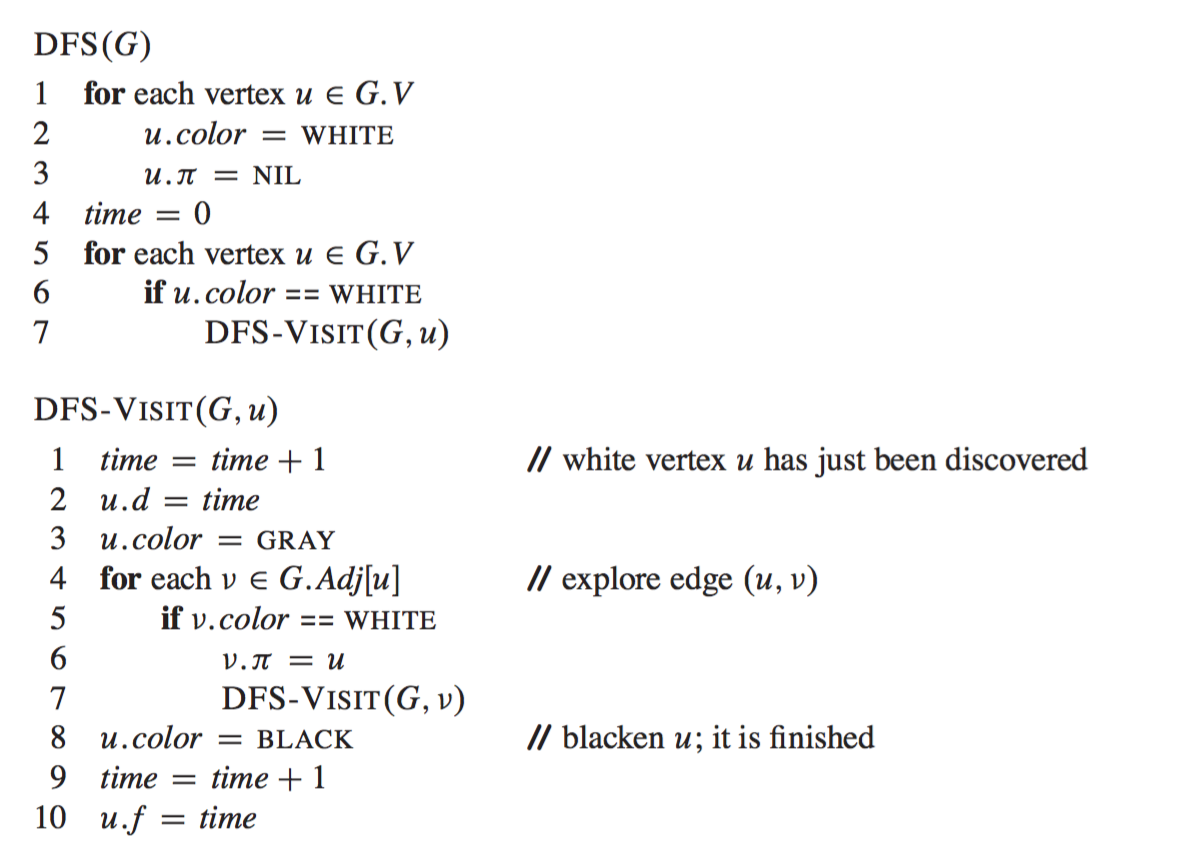
**if(d[w] is not seen before)**

**d[w] = d[v] + 1;**

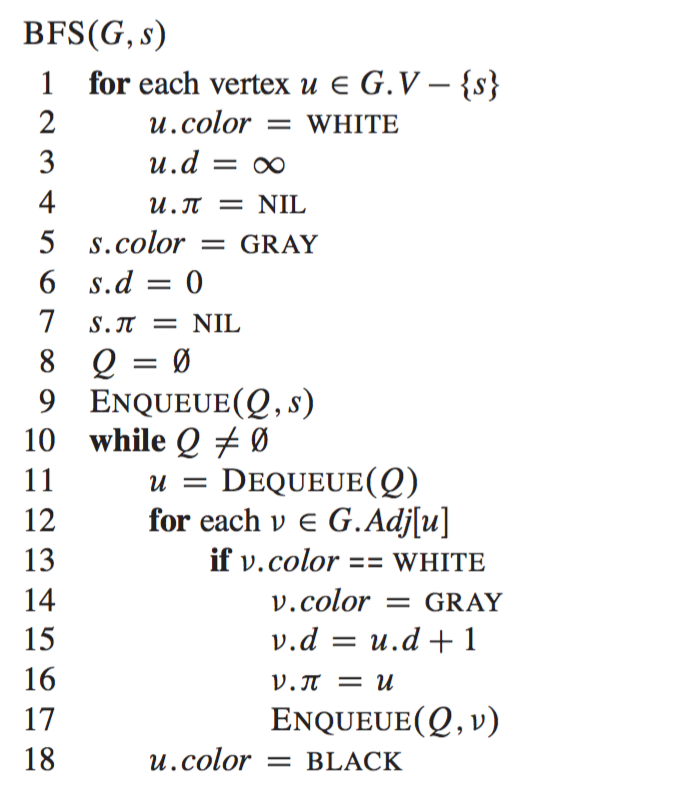
**enqueue w;**

* **Pseudo Code For Traversal of Graph:**

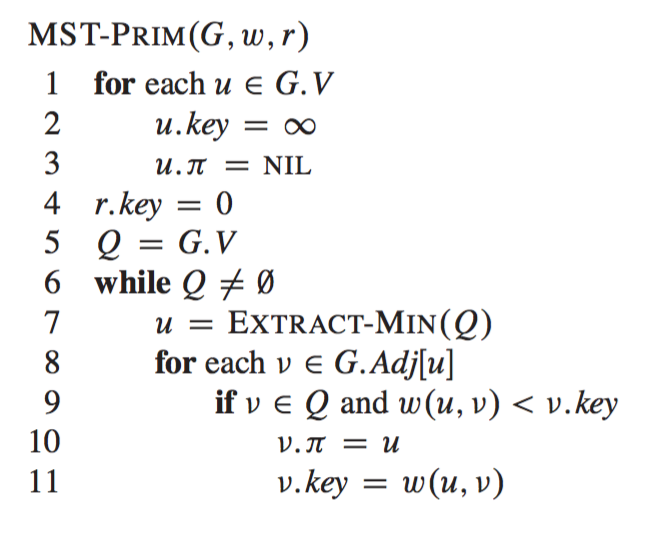
1. **Depth First Search**

****

1. **Breath First Search**

****

* **Minimum Spanning Tree: For each edge (u, v) belongs to E, there is a cost or weight associated with it, w(u, v). The MST is an acyclic subset T belongs to E that connect all the vertices and whose total weight w(T) = Total w (u, v) where (u, v) belongs to T, is minimized.**
* **Pseudo code for Prim’s algorithm**

****

# Common definition questions in 241

1. Give the formal definition of the notion f(n) = O(g(n))
2. What does it mean to say that a method is recursive? Give one possible advantage and one possible disadvantage of using recursive methods.
3. What is the fundamental characteristic of a recursive data structure?
4. How does insertion sort, selection sort and heap sort?
5. What is the difference between truly random numbers and pseudo-random numbers?
6. Describe the algorithm of shuffling an array.
7. What are the advantages of designing programs using ADTs? (give three advantages)
8. How do Java interfaces correspond to abstract data type (ADT)?
9. What are generic types and what is their importance in ADT for collections?
10. What is the difference between a static and a dynamic data structure? How an Array to be used to implement dynamic data structure? What accommodations need to be made to allow for fully dynamic behaviour?
11. How are the ADT’s and behaviour of stacks and queues similar? How to they differ?
12. Give a recursive definition of binary search tree.
13. What are the two conditions that distinguish a heap from a general binary tree?
14. How do the input and output operations of a priority queue differ from those of a normal queue?
15. Why is a heap a natural data structure for representing a priority queue ADT (include a consideration of efficiency)?
16. What is a breadth first traversal of a graph, and how can it be used to compute the distance from a vertex of the graph to each other vertex?
17. What is the difference between breadth first and depth first traversals in linked data structures?
18. What is a spanning tree in a graph? Spanning trees can be constructed using either breadth-first or depth first traversals. How do the two types differ in a qualitative sense?
19. In a weighted graph, describe the algorithm
20. shortest distances between one vertex and all the remaining vertices
21. finding a minimum spanning tree.
22. Barabasi-Albert model for graph, why this is an attractive model of the development of some social network graphs. What are some of the most common features observed in graphs constructed in this way?

# Common definition questions in 241

(Question you want to ask please append at the end of this list.)

**Kurt**

**LM**

**Fergus**

Frank

1.     Give the formal definition of the notion f(n) = O(g(n))

We write f(n) = O(g(n)), if for some constant A, such that f(n) <= A\*g(n), for ALL sufficiently large n.

f(n) = O(g(n)) means that f(n) grows no more quickly that g(n) does. If for some constant A, f(n) <= Ag(n) for ALL sufficiently large n.

2.     What does it mean to say that a method is recursive? Give one possible advantage and one possible disadvantage of using recursive methods.

A recursive method is a method which calls itself. The main advantage of recursion is more elegant and readable code. A disadvantage could be software performance, loops have been heavily optimized and in general perform better.

**A disadvantage of recursive methods is the possibility for stack overflow.**

3.     What is the fundamental characteristic of a recursive data structure?

A recursive data structure is a method which contains a reference to other objects of the same type or closely related type?

4.     How do you do insertion sort, selection sort and heapsort?

Insertion sort algorithm sorts an array by repetitively selection each element from the second to the last and insert they each at a time into the subarray which is already sorted at an appropriate position. The subarray at the beginning just contain the first element of the array which by its natural is sorted.

  Insertion Sort: O(n^2)

a {8, **1**, 5, 3}      k = null              1. Starting at index i = 1, compare value to index i-1

a {8, **1**, 5, 3}      k = 1                  2. If(a[i] < a[i-1]) { assign a[i] to temporary variable k}

a {**8, 8**, 5, 3}      k = 1                  3. Copy a[i-1] to a[i] and compare to decremented indexes (while>0)

a {**1**, 8, 5, 3}      k = 1                  4. Insert k into correct index where a[i-1] < a[i].

a {1, 8, **5**, 3}      k = 5                  5. Repeat by incrementing i.

a {1, **8, 8**, 3}      k = 5

a {1, **5**, 8, 3}      k = 5

a {1, 5, 8, **3**}      k = 3

a {1, 5, **8, 8**}      k = 3

a {1, **5, 5**, 8}      k = 3

a {1, **3**, 5, 8}      k = 3

Selection sort works by repetitively selecting the ith element, and swap it with the element with index i.

Selection Sort: O(n^2)

Underlined values indicate ‘Search Range’

a {8, **1**, 5, 3}                      1. Perform a linear search to find the smallest value in the search range

a {**1, 8**, 5, 3}                      2. Swap the first index of the search range with the smallest no.

a {1, 8, 5, 3}                      3. Decrement search range

a {1, 8, 5, **3**}                      4. Repeat steps 1 – 3 while search range > 1

a {1, **3**, 5, **8**}

a {1, 3, 5, 8}

a {1, 3, **5**, 8}                      note : 5 is the smallest AND is already at the lowest index of the search range so                   does not get swapped

a {1, 3, 5, 8}                      search range == 1, return.

Heap Sort works by representing a heap using array, then repetitively swap the first element in the array which is the root of the heap which the last element of the heap. During each time, heapsize-1, and call heapify method at index 0 which maintain the heap property.

Heapify(A, i), this method take a array and an index as parameter. It assumes the binary tree rooted at index 2\*i and 2\*i + 1 are heaps, but the A[i] might smaller than its children, thus violate the heap property. Heapify let the value A[i] sink down so the subtree rooted at index i obeys heap property.

Heapify(A, i)

L = 2\*2;

R = 2\*i + 1

largest = i

If L <= heapSize and  A[L] > A[i] // decide if left child or right child is the largest

largest = L

If R <= heapSize and A[R] > A[largest]

largest = R

If i != largest

Swap A[i] with A largest

Heapify(A, largest)

5.     What is the difference between truly random numbers and pseudo-random numbers?

Truly random numbers are based on observations of unpredictable physical occurrences eg. roll of dice, flip of coins.

Pseudo-random numbers are calculated using an algorithm with an input seed. I both the algorithm and the seed are known, the output numbers can be predicted.

6.     Describe the algorithm of shuffling an array.

Random\_Shuffle(A)

N = A.length;

For i from 1 to A.length:

Sway A[i] with A[Random(i, n)]

for(int i = array.length - 1; i > 0; i--){

Random r = new Random();

swap(array, i, r.nextInt(i + 1));

}

7.     What are the advantages of designing programs using ADTs? (give three advantages)

8.     How do Java interfaces correspond to abstract data type (ADT)?

9.     What are generic types and what is their importance in ADT for collections?

10.  What is the difference between a static and a dynamic data structure? How an Array to be used to implement dynamic data structure? What accommodations need to be made to allow for fully dynamic behaviour?

A static data structures size is set upon its creation. It can not grow or shrink.

Dynamic data structures grow and shrink as items are added and removed from them.

For an array to be used to implement a dynamic data structure it needs to be able to grow when an item is added to it EVEN IF the array is full. To achieve this, a new array(a2) is created with a size double that of the initial array(a). All items from a are then copied to a2. This takes O(n).

11.  How are the ADT’s and behaviour of stacks and queues similar? How to they differ?

Both are linear data structures with interfaces that allow for the adding(push, enqueue) and removing(pop, dequeue) of items.

The major difference is that while stacks add/remove from the same side giving it FILO behaviour, queues add/remove from opposite ends giving it FIFO behaviour:

Input: A, B, C, D, E

Stack Queue

in/out(push/pop) in(enqueue)

E E

D D

C C

B B

A A

out(dequeue)

Output: E, D, C, B, A Output: A, B, C, D, E

12.  Give a recursive definition of binary search tree.

In binary search tree, each node is bigger than any element in its left subtree, and smaller than any element in its right subtree.

13.  What are the two conditions that distinguish a heap from a general binary tree?

1. A heap is a complete binary tree
2. Every node must be bigger than both of children

14.  How do the input and output operations of a priority queue differ from those of a normal queue?

In priority queue, each element associated with a priority, and when dequeue, it is always the highest priority element got dequeue, when there is a tie, the element enqueued first, deque first.

15.  Why is a heap a natural data structure for representing a priority queue ADT (include a consideration of efficiency)?

If we associated each heap node with a priority and let the heap node implement the comparable interface according to the priority, then by the nature of heap, it matches definition of a priority queue.

16.  What is a breadth first traversal of a graph, and how can it be used to compute the distance from a vertex of the graph to each other vertex?

Breadth first search, given Graph G(V, E), given starting source vertex s.

BFS(G, s):

For each vertex u belongs to V except s

u.parent = nil

u.color = white

U.distance = infinite

s.distance = 0

s.parent = nil

Queue = empty

Enqueue(Q, s)

While Q is not empty:

u <- Q.dequeue

                                   u.color = black

for each v belongs to u’s neighbors:

If v.color == white

v.parent = u

v.distance = u.distance + 1

Enqueue(Q, v)

After using breadth first search to discover the property of Graph G(V, E), the distance from a vertex s to each other vertex v, is the length of path from s to v which formed by the relationship defined by property v.parent.

17.  What is the difference between breadth first and depth first traversals in linked data structures? (this question worth 2 points, it probably has simpler answer)

BFS see above.

DFS(G):

for each vertex u in G.V

u.color = white

u.parent = nil

for each vertex u in G.V

If u.color == white:

DFS\_VISIT(G, u)

DFS\_VISIT(G, u):

u.color = black

for each v in u’s neighbours:

If v.color == white:

v.parent = u

DFS\_VISIT(G, v)

BFS produce a subgraph which is a tree corresponding to the parent property. The subgraph contains a unique simple path from s to v that is also a shortest path from s to v in G.

DFS produce a depth-first forest comprising several trees, because the search may repeat from multiple sources.

18.  What is a spanning tree in a graph? Spanning trees can be constructed using either breadth-first or depth first traversals. How do the two types differ in a qualitative sense?

A spanning tree is a tree that include all the vertices of a graph, and some, but possibly not all, of the edges.

Using BFS, can form this tree, see above.

Using DFS, is a forest comprising several tree.

Qualitative sense?

19.  In a weighted graph, describe the algorithm

a.     shortest distances between one vertex and all the remaining vertices

b.     finding a minimum spanning tree.

A. (G, s, target):

for each vertex u in G.V:

= nil

= infinite

= white

= 0

(Q, s) // Q is a min-priority queue dominated by the distance property

While Q is not empty:

u <- Dequeue(Q)

== black

If (u == target)

Return

for each v in u’s neighbors

If == white

v.parent = u

= + )

After the running the method, the path from s to target, created by the property parent is the shortest path.

B. For finding a minimum spanning tree in a weighted graph, using Prim’s method

MST-PRIM(G, w, r)

for each vertex u in G.V:

u.distance = infinite

u.parent = nil

r.distance = 0

Q is a min-priority queue which the priority is v.distance = w(u, v) and contains all the vertices

While Q is not empty:

u <- dequeue Q

for each v in u’s neighbors

If v is in Q and w(u, v) < v.distance

v.distance = w(u, v)

v.parent = u

G means graph, w is a function given to which gives the weights between vertex u and v, r is the arbitrary starting vertex.

The minimum spanning tree must contain one of the edges of of least weight, because the Q is a min-priority queue which dominated by the property of w(u, v), so during the discovery process the vertex v  in which the w(u, v) is the minimum value, will be dequeued and the edge formed by v and its parent u, will be contained in the edges of MST.

20.  What is the Barabasi-Albert model? Why this is an attractive model of the development of some social network graphs. What are some of the most common features observed in graphs constructed in this way?

The Barabasi-Albert model displays preferential attachment behaviour - vertices with more neighbours have a higher probability of attaching to newly added vertices according to (simplified) n/t where n = vertices no. of neighbours and t = total no. of vertices in graph.

Due to this creation behaviour, we see that ‘clumping’ has a high probability of occurring. Where a handful of vertices are connected to the majority of vertices.

This is a good model for social networking graphs as it makes sense that individuals that have been using the network for a long time (or are awesomely popular) will have more neighbours and individuals that have recently joined will have very few but will probably be connected to an individual with maybe neighbours.

21. Given the BST:

50

- -

34 83

     -      -      -             -

18 42 67 95

   -   -                -      -

     25     47     62     73

i. add 36

ii. remove 50

i. 50(34(18[25])[42(36)[47]]) [83(67(62)[73])[95]]

ii. 62(34(18[25])[42(36)[47]]) [83(67[73])[95]]

22. Given the Heap:

93

            - -

67 83

* -       -       -

51 31 67 12

   -     -               -     -                 -

25     30     29        2         62

i. add 85

ii. remove 93

i.  93 (67(51(25)[30])[31(29)[2]]) [85(83(62)[67])[12]

ii. 85 (67(51(25)[30])[31(29)[2]]) [83(67[62])[12]

23. What are the minimum and maximum possible depths for a binary tree with n nodes?

Minimum is log(n), maximum is n-1