Date of Exam: Wednesday 11/18 Topics 9 - 17

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

TOPIC 9: Heaps, Heapsort and Priority Queues

As a Complete Binary Treees

- n: the number of nodes
- number of leaves: $\lceil \frac{n}{2} \rceil$

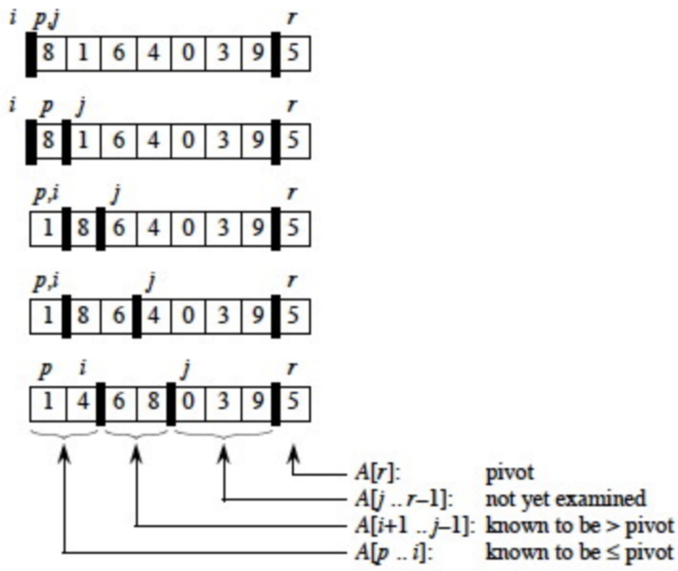
Array Representation

- root: A[1]
- parent of A[i]: $A[\frac{i}{2}]$
- left child: A[2i]
- right child: A[2i+1]

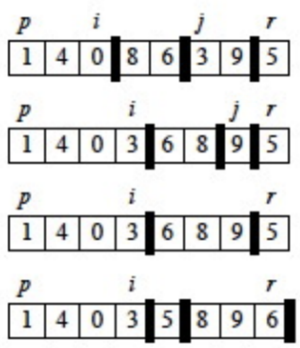
TOPIC 10: Quicksort, Theoretical Limits, and O(n) Sorts

Chapter 7 and 8 from CLRS Quick Sort
example on the homework to do

- Partition: $\Theta(n)$
- Worst Case: $\Theta(n^2)$
- Best Case: $\Theta(nlgn)$



Continuing ...



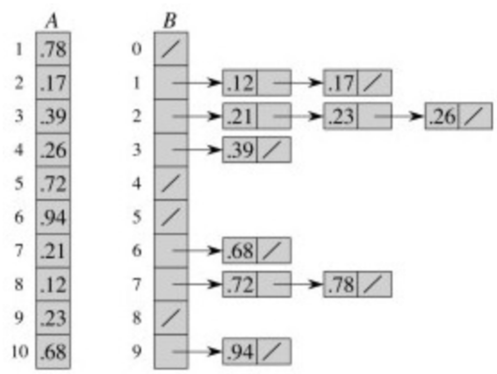
Counting Sort

- determines how many are less or greater than the input
- stable sort example
- requires $\Theta(n + k) \rightarrow \Theta(n)$ since $k \in \mathbb{R}$

Radix Sort

- sorting from least to most significant digit
- Run Time: $\Theta(d(n+k))$

Bucket Sort



- Worst Case Run-Time: $\Theta(n^2)$
- Average: $\Theta(n)$

TOPIC 11: Balanced Trees (2-3-4 and Red-Black)

2-3-4 Trees

- Node Size: internal node has 2-4 children
- Depth Property: external nodes have the same depth
- 2-4 Tree storing n items has a height of $\Theta(\lg n)$
- Insertion runs $\Theta(\lg n)$
- Overflow is handled with Split Operation
- Deletion runs $\Theta(\lg n)$

- Case 1: Fusion Operation
- Case 2: Transfer Operation

Red Black Trees

- color: either red or black
- root: always black
- external: leaf is black
- internal: if node is red then children are black
- depth: each node, all the paths from the node to descendant leaves contain the same number of black node

Insertion

- Case 1: Z.uncle is Red
  Solution recolor Z.parent, uncle, grandparent
- Case 2: Z.uncle is black (Δ)
  Solution Rotate Z.parent
- Case 3: Z.uncle is black with a line
  Solution rotate z.grandparent and recolor
- insert: O(log n)
- recolor:O(1)
- violation clean up: O(1)

Deletion

- runtime: O(log n)

Case 1

Node x is black and its sibling w is red

1. Color w black
2. Color $x.p$ red
3. Rotate $x.p$
   a. If x is the left child do a left rotation.
   b. If x is the right child do a right rotation.
4. Now we have to change w
   a. If x is the left child set $w = x.p.right$
   b. If x is the right child set $w = x.p.left$
5. With x and our new w, decide on case 2, 3, or 4 from here.

Case 2

Node x is black and its sibling w is black and both of w's children are black

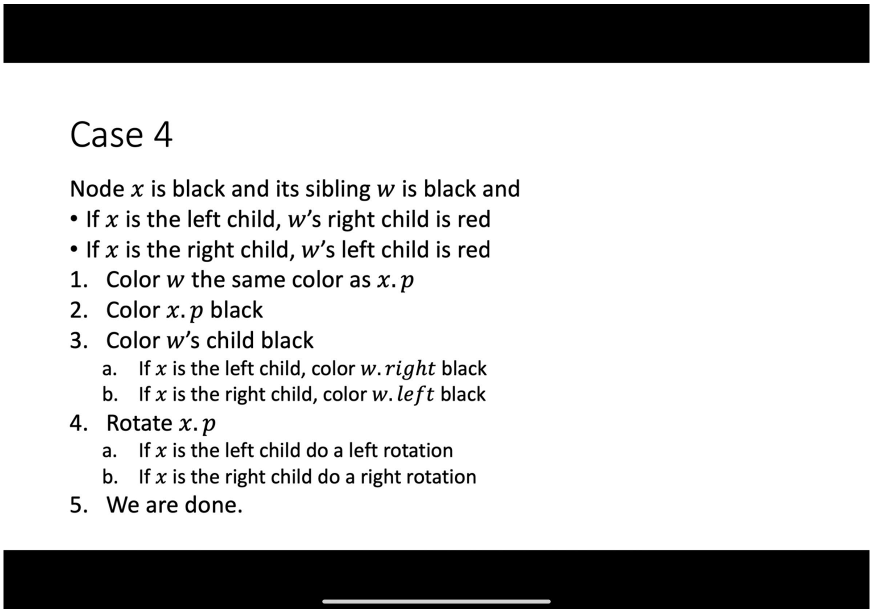
1. Color w red
2. Set $x = x.p$
   a. If our new x is red, color x black. We are done.
   b. If our new x is black, decide on case 1, 2, 3, or 4 from here. Note that we have a new w now.

Case 3

Node x is black and its sibling w is black and

- If x is the left child, w's left child is red and w's right child is black
- If x is the right child, w's right child is red and w's left child is black

1. Color w's child black
   a. If x is the left child, color $w.left$ black
   b. If x is the right child, color $w.right$ black
2. Color w red
3. Rotate w
   a. If x is the left child do a right rotation
   b. If x is the right child do a left rotation
4. Now we have to change w
   a. If x is the left child set $w = x.p.right$
   b. If x is the right child set $w = x.p.left$
5. Proceed to case 4.



TOPIC 12: Dynamic Programming

- Unlike D and Q applies to subproblems that overlap
- Solves each sub problems once and saves its answer on a table
- Examples: Rod Cutting
- Bottom Up
- Can handle interdependence

4 step for DP

1. characterize the optimal solution
2. recursively define the value of an optimal solution
3. compute the value of the optimal solution
4. construct an optimal solution from the computed information

TOPIC 13: Greedy Algorithms and Huffman Codes

- find the solution using top down
- assume that if the objective function is optimized locally it will do it globally
- cannot do interdependence

TOPIC 14:Graph Representations and Basic Algorithms

- G = (V,E)
- V the set of vertices
- E the set of Edges

Adjacency List

- Space needed: $\Theta(n+k)$
- List all vertices adjacent: Θ(degree(u))
- determine whether $(u,v) \in E : O$(degree(u)))

Adjacency Matrix

- Space Required: $\Theta(V^2)$
- Time to list all vertices adjacent: $\Theta(V)$
- determine whether $(u,v) \in E : O(1)$

Breadth First Search

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

- uses a queue
- WHITE: not encountered
- GRAY: have been encountered but processing

- BLACK: done being processed
- enqueue and dequeue takes O(1)
- total time to queue O(V)
- scanning adjacency list $\Theta(E)$
- running time of BFS: $O(V + E)$

Depth First Search

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE  // only explore from undiscovered vertices
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
1  time = time + 1         // u has just been discovered: record time
2  u.d = time
3  u.color = GRAY          // u is now on active path
4  for each v ∈ G.Adj[u]   // explore edge (u,v)
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  u.color = BLACK         // u finished: blacken and record finish time
9  time = time + 1
10 u.f = time
```

- operates in a stack like manner
- will search $\forall$ vertices until $\forall$ edges are discovered
- Line 1 to 5 runs $\Theta(V)$
- Line 4 runs $\Theta(E)$
- Total Runtime of DFS: $\Theta(V + E)$

Classification of Edges

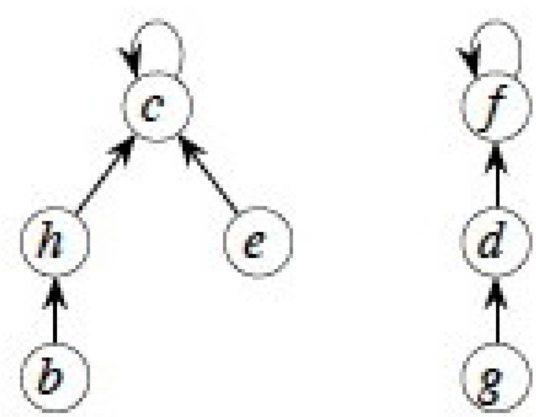
- Tree Edge
- Back Edge: (v,u): v is descendant of u
- Tree Edge: (v,u) v is a descendant of u but not a tree edge
- Cross Edge: any other edge that goes between vertices in the same or different depth first tree

White Path Theorem
Vertex v is a descendant of u iff at time u.d there is a path from u to v consisting of only white vertices
DFS Theorem
DFS of an undirected graph produces only tree and back edges: never forward or cross edges.
Topological Sort

```
TOPOLOGICAL-SORT(G)
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

- a linear ordering of vertices s.t. if $(u, v) \in E$ then u appears before v in that ordering
- runtime: $\Theta(V + E)$

Forest Representation of Disjoint Sets



- amortized cost of Make Set O(1)
- amortized cost of each link operation is $O(\alpha(n))$
- amortized cost of each find set operation is $O(\alpha(n))$
- sequence of m Make-Set, Union, and Find set can be performed on a disjoint set forest with union by rank and

TOPIC 15: Amortized Analysis

TOPIC 16:Sets and Union-Find:

Dynamic Disjoint Sets

- Make-Set(X)

- Union(X,Y)
  if $x \in S_x \land y \in S_y \rightarrow$ combine the two sets
  destroys the x and y sets since they must be disjoint
- Find-Set(x): return a rep containing x
- n = num of make set ops
- m = total number of ops
- $m \geq n$
- Union operation count at most n - 1

TOPIC 17: Minimum Spanning Trees

Properties of MST

- any tree has no cycles
- one path between vertices
- there might be more than one MST

Safe Edge Theorem

- A **cut** $(S, V - S)$ is a partition of vertices into disjoint sets S and $V - S$.
- Edge $(u,v) \in E$ **crosses** cut $(S, V - S)$ if one endpoint is in S and the other is in $V - S$.
- A cut **respects** A iff no edge in A crosses the cut.
- An edge is a **light edge** crossing a cut iff its weight is minimum over all edges crossing the cut. (There may be more than one light edge for a given cut.)

Let G = (V,E) and A $\subset$ G such that G is a MST.
(S,V-S) be a cut that respects A and (U,V) be a light edge crossing (S, V-S).
Then (u,v) is a safe for A.

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

- edges are processed greedy
- organizes in nondecreasing order

- initialize: O(1)
- First for loop: $|V|$
- Sort E: O(E lg E)
- Second For Loop: O(E)
- Run Time O(E lg V)

Prim's Algorithm

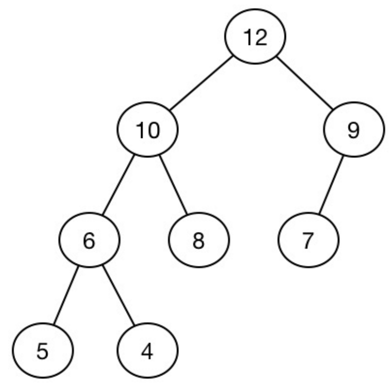
```
PRIM(G, w, r)
Q = ∅
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
    INSERT(Q, u)
DECREASE-KEY(Q, r, 0)        // r.key = 0
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            DECREASE-KEY(Q, v, w(u, v))
```

- first for loop to queue O(V lg V)
- decrease key of r: O(lg V)
- while loop: O(V lg V)
- decrease key: O(E lg V)
- Analysis: O(V lg V) + O(E Lg V)
- if G is connected then O(E lg V)

Review Questions

1. What is the expected runtime of randomized quicksort?

2. Is this a Max-Heap?



3. Why do we focus on the expected time of randomized algorithms such as randomized quicksort, and not the worst case?

4. What is the smallest possible depth of a leaf in a decision tree for a comparison sort of n items?

5. What is the worst-case runtime of randomized quicksort?

6. What is the best-case runtime of randomized quicksort?

7. Suppose that you knew that your array was sorted except for the possible misplacement of one or two elements: Which of the sorts that we have studied would be the fastest? What is its expected big-O runtime given your choice in the above question?

8. What is the smallest possible height of a decision tree for a comparison sort of n items?

9. The runtime of counting sort is $\Theta(n + k)$. What is k?

10. You will show the result of running Radix sort on the following words

BOW, DOG, FAX, DIG, BIG, COW

11. The runtime of radix sort using counting sort is $\Theta(d(n + k))$. What is d?

12. Which of the sorting algorithms are stable sorts?

13. In general, which version of Dynamic Programming is more appropriate to which situation?

    - only some sub-problems must be solved
    - all sub-problems must be solved

14. Write the name of the graph search to the characterization of its process.
    - Processes vertices in a queue-like manner
    - Processes vertices in a stack-like manner

15. Given a "classic" matrix representation of a directed graph (example in right hand figure below), how long does it take to compute and print a table of the in-degree of all the vertices in the graph?

16. Which of the following (listed randomly) are true for the aggregate method of analysis?

17. Given a "classic" adjacency list representation of a directed graph (example in middle figure below), how long does it take to compute and print a table of the in-degree of all the vertices in the graph

18. A minimum weight edge in a connected graph G must belong to some minimum spanning tree for G. (T/F)

19. Which of the following (listed in random order) are true of the accounting method of analysis?

20. Claim: Prim's algorithm correctly finds minimum spanning trees in connected graphs with negative weights.(T/F)

21. A minimum weight edge in a connected graph G must belong to every minimum spanning tree for G. (T/F)

22. If an edge (u,v) is contained in some minimum spanning tree for graph G then it is a light edge crossing some cut of G. (T/F)

23. Claim: Kruskal's algorithm correctly finds minimum spanning trees in connected graphs with negative weights (T/F)