Maharishi University of Management

FINAL - ANSWERS

Course Title and Code: *CS 435 - Design and Analysis of Algorithms*
Instructo: *Dr. Emdad Khan*
Date: *Friday 07/16/2015*
Duration: *10am - 12 pm*

Student Name:

Student ID:

Total Mark
_____
46 + 4 (Bonus)

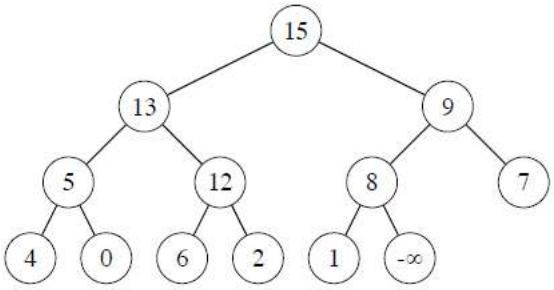
1. This is a closed book exam. Do not use any notes or books!
2. Show your work. Partial credit will be given. Grading will be based on correctness, clarity and neatness.
3. We suggest you to read the whole exam before beginning to work any problem.
4. There are 4 questions worth a total of 46 points.
5. There is an additional question as bonus (4 points).
6. There are TOTAL **10 pages** (including this page) - **first 7 pages are for questions and answers.** Please write answers in the space provided and the back side of the pages. This is the preferred option. There are 3 Extra blank pages stapled. You can use these pages as scratch paper or to write answers when back side of each pages is full. Do NOT REMOVE any Pages. **Please write your Student ID on the 3 extra blank pages.**
7. WE STRONGLY RECOMMEND TO COMPLETE YOUR WORK IN THE PAGES PROVIDED. ADDITIONAL BLANK PAGES ARE NOT RECOOMENDED AS IT WILL NOT BE NEEDED.
8. Please do NOT use Cell phones or Calculators.

Question 1: 11 points (5 + 3 + 3)

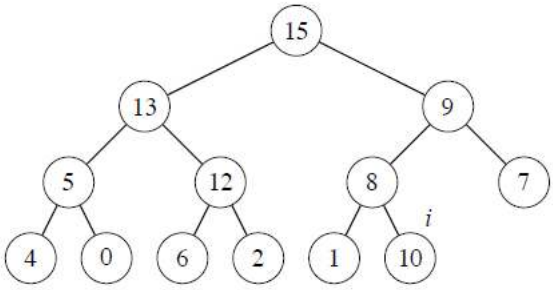
a. Use the following Heap-Insert algorithm to insert 10 in the array A = {15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2 ,1}. Make sure you take care of needed heapification as appropriate. Show all the steps starting at i and moving i to the final step.

MAX-HEAP-INSERT(A, key)
1 $A.heap\text{-}size = A.heap\text{-}size + 1$
2 $A[A.heap\text{-}size] = -\infty$
3 HEAP-INCREASE-KEY$(A, A.heap\text{-}size, key)$

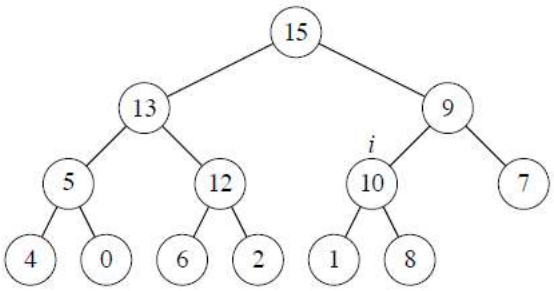
Ans. Starting point is i=10 as after executing lines 1 and 2, the heap size becomes 10. The steps are shown below:



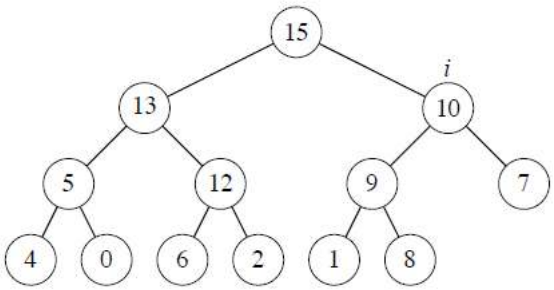
(a)



(b)



(c)



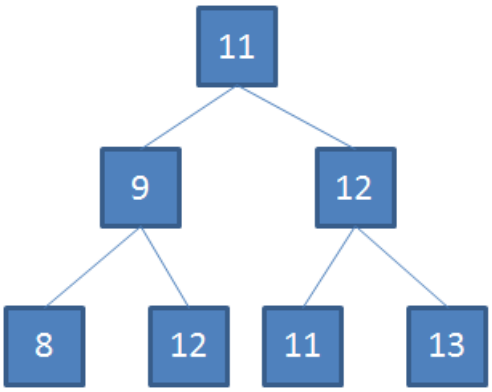
(d)

b. Insert the following sequence of numbers 23, 46, 12, 21, 75, 5, 3 into a hash table of size 9 using h(x) = x%9 as a hash function, where % mean "mod". Use Chaining with Linked List to avoid collision.

Ans.

0	1	2	3	4	5	6	7	8
	46		12,21, 75,3		23,5			

c. Use Pre-Order, In-Order and Post-Order Tree-Walk to print all keys in the following Tree:



Ans.

pre-order tree walk: 11,9,8,12,12,11,13
in-order tree walk: 8,9,12,11,11,12,13
post-order tree walk: 8,12,9,11,13,12,11

Question 2: 11 points (6 + 5)

a. Consider a 0-1 Knapsack problem with 4 items and with a max weight of **8**. The benefits and weight values are shown below.

Item	2	3	4	5
Benefit	3	4	5	6
Weight	2	3	4	5

Find an optimal solution using Dynamic Programming. Show all Table values. Be sure to state

both the value of the maximum benefit as well as the item(s) selected. Explain how you determine the items selected (you can also use a column for Keep-items / selected items).

Ans.

We use dynamic programming to resolve this problem. So we're going to fill in a table that has 4 rows (as there's r items in our problem instance) and 7 columns (since Wmax = 7). Recall by the time we have filled in all of row i, we have found the maximum benefit possible using items {1,…,i} (or the equivalent alphabetic characters as I'm using for the items here). Also recall that to find B[i, w] (the maximum benefit possible using items {1,…,i}, with maximum total weight w, we use the equation:

$B[i, w] = \max\{B[i-1;w]; w_i + B[i-1,w-w_i]\}$.

The table consisting of the B[i, w] values is given below:

	1	2	3	4	5	6	7	8	Items Selected
2	0	3	3	3	3	3	3	3	2
3	0	3	4	4	7	7	7	7	2,3
4	0	3	4	5	7	8	9	9	3,4
5	0	3	4	5	7	8	9	10	3,5

(For example, the third line in this table is the result we get when we try to find the maximum benefit that is achievable using the set of items {2, 3, 4}.) We see from the table that the maximum benefit obtainable (over the set of all 4 items) is 10. Tracing backwards through the table (e.g 10 -6 = 4 in row 2), we can find that the set of items that gives us this benefit of 10 is the set (in the order we discover them) {3,5}.

b. Show that a Red-Black tree with n-internal nodes has height at most 2 lg(n+1).

Ans.

We start by showing that the subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes. We prove this claim by induction on the height of x. If the height of x is 0, then x must be a leaf ($T.nil$), and the subtree rooted at x indeed contains at least $2^{bh(x)} - 1 = 2^0 - 1 = 0$ internal nodes. For the inductive step, consider a node x that has positive height and is an internal node with two children. Each child has a black-height of either $bh(x)$ or $bh(x) - 1$, depending on whether its color is red or black, respectively. Since the height of a child of x is less than the height of x itself, we can apply the inductive hypothesis to conclude that each child has at least $2^{bh(x)-1} - 1$ internal nodes. Thus, the subtree rooted at x contains at least $(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$ internal nodes, which proves the claim.

Let h be the height of the tree. According to property 4 of Red Black tree, at least half the nodes on any simple path from the root to a leaf, not including the root must be black. Thus, the black height of the root must be at least h/2. Hence, we have,

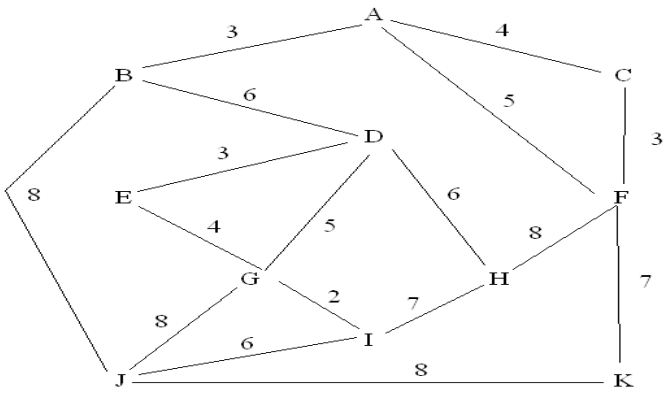
$$n \geq 2^{h/2} - 1 .$$

Moving the 1 to the left side, and then taking log on both sides, we get,

$$\lg(n + 1) \geq h/2, \text{ or } h \leq 2\lg(n + 1).$$

Question 3: 12 points (6 + 4 + 2)

a. Consider the following graph -



Using Dijkstra's algorithm, determine the shortest path from node A to I. Show the steps, your tables and the resulting path.

Ans.
The min path is A->B->D->G->I for a cost of 16. Start at node A. Then MinQ operation will yield node B instead of C as the enxt node as $|AB| < |AC|$. Using the same method, the next node will be D etc.

b. Let G = (V, E) be a weighted, directed graph with exactly one negative weight edge and no negative-weight cycles. Give an algorithm to find the shortest distance from s to all other vertices in V that has the same running time as Dijkstra.

Ans.
Let's say the negative-weight edge is (u; v). First, remove the edge and run Dijkstra from s. Then, check if $d_s[u] + w(u,v) < d_s[v]$. If not, then we're done. If yes, then run Dijkstra from v, with the negative-weight edge still removed. Then, for any node t, its shortest distance from s will be $\min(d_s[t], d_s[u] + w(u, v) + d_v[t])$.

c. Can you use Bellman-Ford shortest path algorithm on undirected graph? Explain your answer. If your answer is yes, will you be able to deal with negative weight?

Ans.
Yes, we can use Bellman-Ford shortest path algorithm on undirected graph. This is because an undirected graph is a directed graph with arrows in both direction. However, we need to ensure that there is a no negative cycle. A negative cycle in a directed graph means a negative weight in an undirected graph. So, to use Bellman-Ford algorithm on an undirected graph, we need to make sure all weights are >= 0.

Question 4: 12 points (7 + 5)

a. Assume that Subset-Sum problem is NP-Complete. Show that the Knapsack problem is NP-Complete by reducing the Subset-Sum problem to Knapsack problem. Show all the key details. Use W for the weight and V for values for the Knapsack problem. Use {si} as the set, and "t" as the total desired sum for the Subset-Sum problem.

You just need to explain the reduction concept well with specifics - no need to do a formal proof. Assume that Knapsack belongs to NP. [Hint: see that both problems are very similar].

Ans.

The first step is to show that Knapsack belongs to NP. Given an input set, it is easy to see if the total weight is at most W, and if the corresponding profit is at least V. It takes only linear time to

add all profits and weights to find true / false result of the decision problem. Thus, we can verify a solution in polynomial time.

The 2nd step is to use Subset-sum problem and reduce it to Knapsack problem.

Knapsack problem

Instance: *Non-negative weights w*1*, w*2*, , , wn, W, and profits v*1*, v*2 *, , vn, V.*

Question: *Is there a subset of weights with total weight at most W, such that the corresponding profit is at least V?*

Subset Sum problem

Instance: *Non-negative integer numbers s*1*, s*2*, , , , sn and t.*

Question: *Is there a subset of these numbers with a total sum t?*

It is very easy to reduce an instance of Subset Sum problem to an instance of Knapsack problem.

We just create such a Knapsack problem that has the following:

wi = ci = si and

W = V = t

NOTE: We can give full points if students just do up to above line.

The *Yes*/*No* answer to the new problem corresponds to the same answer to the original problem. This can be shown with a few extra lines of reasoning (omitted for simplicity as we are just testing whether the students could map the Subset-sum problem to Knapsack problem).

b. Define NP problems, NP-Complete problems and NP-Hard problems? Can all NP-Hard problems be reduced to NP problems? Can all NP problems be reduced to NP-Hard problems? Explain your answers.

Ans.

NP Problems: The class **NP** consists of all those decision problems whose positive solutions can be verified in polynomial time given the right information, or equivalently, whose solution can be found in polynomial time on a non-deterministic machine.

NP-complete problems are a set of problems that any other **NP**-problem can be reduced

to in polynomial time, but retain the ability to have their solution verified in polynomial time. Alternatively, A problem (in NP) is NP-complete if any problem in NP is reducible to it

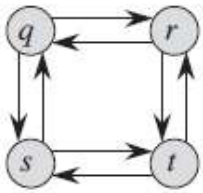
NP-Hard Problem: **NP-hard** problems are those at least as hard as **NP**-complete problems, meaning all **NP**-problems can be reduced to them, but not all **NP-hard** problems are in **NP**, meaning not all of them have solutions verifiable in polynomial time.

No, not all NP-Hard problems can be reduced to NP problems.

Yes, all NP problems can be reduced to NP-Hard problems. This is because NP-Hard problems are at least as hard as the NP-Complete problems (where all NP problems can be reduced to).

Question 5: 4 points [Bonus Question]

Consider the unweighted directed graph shown below. Does this graph has optimal substructure to find the longest simple path, say between q and r or q and t? Explain your answer. Can you use Dynamic Programming for this problem. Explain why or why not.



Ans.

No, this graph does not have optimal substructure. Consider

path $q \to r \to t$, which is a longest simple path from q to t. Is $q \to r$ a longest simple path from q to r? No, for the path $q \to s \to t \to r$ is a simple path that is longer. Is $r \to t$ a longest simple path from r to t? No again, for the path $r \to q \to s \to t$ is a simple path that is longer.

Since the graph does not have optimal substructure, we cannot use Dynamic Programming for this problem.