CS 58000\_01 Algorithm Design Analysis & Implementation(3 cr.)

Assignment As\_04 (Exam 02)

Student Name: **Harsh Sharma**

This assignment As\_04 is due 11:59 p.m., Tuesday, November 28, 2023. Please submit your assignment to Brightspace (purdue.brightspace.com). No late turn-in is accepted. Please write your name on the first page of your assignment. (Take 10 off without giving your proper name on the first page). Your file name should be the first character of your first name followed by your last name such as PNg.docx. Please number your problem-answer clearly such as Problem 1a, 1b, 1c, 1d, Problem 2a, 2b, …. The problems’ answers must be arranged in order. Please answer your questions using only a Word file (.docx file). No pdf file will be accepted. Without using a Word file (.docx file) the submitted problems’ answers would not be graded, or take 10 points off. If you attach a pdf page of the solution to your Word file, please leave a few lines blank before your solution page; this allows us to place the cursor for writing comments on your Word file.

The total number of points for this Assignment\_04 (Exam 02) is 200 points.

Problem 1 [40 points]

Given the following array A[0..15] contains 13 elements.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3 | 14 | 27 | 31 | 39 | 42 | 55 | 70 | 74 | 81 | 85 | 93 | 98 |  |  |

When answering these questions 1a through 1d, it is helpful to recognize what an equivalent binary search tree looks like:



1a. What is the largest number of key comparisons made by binary search in searching for a given key from the given array A[0..15]?

Ans: Given a binary tree with a total of N nodes we need **log2(N)** comparisons since it’s like a binary search, dividing the search space in half in every iteration.

Here we have a binary tree with a total of 16 elements and therefore the maximum number of comparisons we need to perform are log2(13) ~ 4 comparisons.

1b. List all the keys (elements) of this given array A[0..15] that will require the largest number of key comparisons when searched for using the binary search.

Ans: The largest number of comparisons will be required by all the elements that are present in the **last level** of binary tree. In the given binary tree, the elements present in the **fourth level** will need the largest number comparisons. And they are: **14, 31, 42, 74, 85, 98.**

1c. Find the average number of key comparisons made by binary search in a successful search in this array. (Assume that each key is searched for with the same probability.)

Ans: There are a total of 13 keys. Lets see how much comparisons these keys require at different levels.

Level 1 (root) : 55 : total of 1 comparisons (1 key requires 1 comparison)

Level 2: (27, 81) : Total of 2 comparisons. (2 keys require 2 comparisons)

Level 3: (3, 39, 70, 93) : Total of 3 comparisons. (4 keys require 3 comparisons)

Level 4: (14, 31, 42, 74, 85, 98) : Total of 4 comparisons. (6 keys require 4 comparisons)

So total comparisons = (6 \* 4) + (4 \* 3) + (2 \* 2) + (1 \* 1) = 24 + 12 + 4 + 1 = 36 + 5 = 41

Total number of keys: 13

Therefore, average number of comparisons required are: 41/13

= 3.1538 comparisons.

1d. Find the average number of key comparisons made by binary search in an unsuccessful search in this array. (Assume that searches for keys in each of the 14 intervals formed by the array’s elements are equally likely.)

Ans:

For unsuccessful search, we have 14 intervals between 13 keys.

Level 1 (root, 55): 1 interval requires checking 1 key

Level 2: (27, 81): 3 intervals require checking 2 keys.

Level 3: (3, 39, 70, 93): 4 intervals require checking 3 keys.

Level 4: (14, 31, 42, 74, 85, 98): 6 intervals require checking 4 keys.

So total comparisons = (6\*4) + (4\*3) + (3\*2) + (1\*1) = 24 + 12 + 6 + 1

= 36 + 7

= 43

With 14 total intervals, the average number of comparisons is:

Average comparisons = Total comparisons/Total intervals = 43/14 = 3.0714

Problem 2 [ 30 points]

Approaches to obtaining solutions: Either use “programming”, “hands (manual) computation without writing program code”, or “a mix of programming and hands computation” to yield the solutions for this problem. Comments on your program code and program output are required.

Given the following array A[0..15] contains 13 elements.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3 | 14 | 27 | 31 | 39 | 42 | 55 | 70 | 74 | 81 | 85 | 93 | 98 |  |  |

2a. Use Max Heapify(A, i), 0 < i to maintain the given array A[0..15] a max-heap. Show step-by-step in terms of the intermediate resulting arrays

2b. Using the resultant max-heap array obtained from 2a, apply the heapsort algorithm to obtain a sorted array A[0..15] in descending order. (No max-heapify() can be further applied.) Show step-by-step in terms of the intermediate resulting arrays.

2c. State time efficiency of both Max-Heapify and Heapsort. And state space efficiency of both Max-Heapify and Heapsort.

Solution:

Time Efficiency of Max-Heapify:

To create a Heap from some array with the N numbers of element in it, we would require to insert every single element into the Heap.

To insert an element in Heap, the best case is if we had to only check once that child and parent satisfy the property and second best case is if we have to shift the child once, and then the number of checks will increase until we reach the root that is logN of checks and shifts

So, as we have seen that the inserting of an element in heap, takes **O(logN)** inserting for all elements will go from bottom to up, therefore we will be considering the worst-case time complexity **O(logN)** and now since, we are performing insertion on N number of elements then the whole time complexity will be: N \* log N

**O(NlogN)**

Space Efficiency of Max-Heapify:

We need to store the heap in a array and the size of the array will be determined by the number of elements. Therefore for N elements the space complexity will be O(N).

**O(N)**

Time Effciency of Heap-Sort:

Sorting in a Heap is done by continuously performing the Delete operation for all the available elements, and there are actually, in total N number of elements, The deleted element gets stored at the end of the array, in the memory that we are not using anymore for the Heap.  
Completing this whole procedure produces a sorted array.

Since, the complexity of deleting an element from any Heap is O(logN), therefore, the time complexity for sorting goes to N times of O(logN).  
That is,

**O(NlogN)**

Space efficiency for Heap Sort:

Since the sorting can be achieved inplace we don’t need extra space. Therefore, it is **O(1).**

Problem 3 [ 50 points]

Approaches to obtaining solutions: Either use “programming”, “hands (manual) computation without writing program code”, or “a mix of programming and hands computation” to yield the solutions for this problem. Comments on your program code and program output are required.

Given a weighted undirected graph G, which is as follows:

22

36

3

1

10

10

3

3

16

18

1

10

4

1

16

4

21

16

4

13

3

8

9

6

17

4

27

4

1

1

130

16000000

8

8

9

3a. Construct

1. a weighted adjacency list representation and

Solution: Weighted adjacency list representation,

A = B(10), K(3)

B = C(36), A(10)

C = B(36), D(3)

D = K(16), X(1), C(3)

E = X(22), I(3), H(10)

F = X(10)

G = t(9)

H = E(10), Z(13), O(4)

h = R(27), P(17), O(13)

I = N(4), n(1), E(3)

J = K(18), n(1), L(16)

K = D(16), A(3), J(18)

L = J(16), Y(3)

M = n(4), N(21)

N = I(4), M(21), Z(16)

n = J(1), I(1), M(4)

O = T(8), H(4), h(13)

P = h(17), Y(6), S(4)

Q = Z(4)

R = h(27)

S = P(4), Y(8), U(9)

T = O(8), V(16), U(1)

t = W( 8), G(9), U(1)

U = T(1) , t(1), S(9)

V = T(16)

W = t(8)

X = D(1), F(10), E(22)

Y = S(8), L(3), P(6)

Z = Q(4), N(16), H(13)

1. a weighted adjacency matrix representation

Ans:

[0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[10, 0, 36, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 36, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 10, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9]

[0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0]

[0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

[3, 0, 0, 16, 0, 0, 0, 0, 0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 13, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 6, 0, 17, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 27, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 9, 0, 0, 0, 8, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 1, 16, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8]

[0, 0, 0, 1, 22, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 6, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 0, 0, 16, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 17, 0, 27, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 8, 0, 0, 0, 0, 0, 0]

Both representations will be used to traverse the given weighted undirected graph

Write a program or use hands to traverse the given weighted undirected graph, based on its **weighted adjacency list** representation obtained in problem 3a(i). Construct its **depth-first tree** (also called depth-first search tree) forest **starting from vertex A.** In your obtained DFS tree forest, show the **tree edges** (indicated as solid lines) and **back edges** (indicated as dotted lines) for your trees. For programming, use the form (A 3(K … K) … A), which means that there is a tree edge from vertex A to vertex K with weight 3, to display the parentheses structure linear representation to represent the program output tree forest, and the form (X, Y) to describe each of the back edges. **Traversal’s stack** contains symbols (such as V(i, j) or Vi, j, the first subscript number indicates the order in which a vertex V was first visited, say at i, (push V onto the stack), where 0 < i n; the second one indicates the order in which it became a dead-end, say at j (pop V off the stack), where 0 < j n. n is the total number of vertices for the given graph. For simplicity’s sake, please **use two time-stamps**: one is 0 < i n, the order for **pushing a vertex onto the stack** counting from 1 through n. The other one is 0 < j n, the order for **popping off a vertex from the stack** counting from 1 through n. For this problem, you need to answer 3b through 3e, which are as follows:

3b. Show the traversal’s stack (FILO Queue) with time-stamp, and give the orderings of vertices yielded by the DFS.

3c. Construct the corresponding depth-first (DFS) tree forest, with indications of tree edges and back edges. Find the weight for each path starting with source A?

3d. What is the given graph called? Is the given graph acyclic? What is the topological sort ordering for the graph, if it exists? State all articulation points of the given graph, if they exist.

3e. What are the time efficiency and space efficiency of the DFS?

Problem 4 [ 40 points]

Approaches to obtaining solutions: Either use “programming”, “hands (manual) computation without writing program code”, or “a mix of programming and hands computation” to yield the solutions for this problem. Comments on your program code and program output are required.

Write a program or use hands to traverse the weighted undirected graph given in Problem 3, based on its **weighted adjacency list** representation obtained in Problem 3a(i). Construct its **breath-first tree** (also called depth-first search tree) forest **starting from vertex A.** In your obtained BFS tree forest, show the **tree edges** (indicated as solid lines), **back edges, forward edges, and cross edges** (indicate the last three types of edges as dotted lines) for your trees. For programming, use the form (A 3(K … K) … A), which means that there is a tree edge from vertex A to vertex K with weight 3, to display the parentheses structure linear representation to represent the program output tree forest, and the form (X, Y) to describe each of the back edges, forward edges, and cross edges.

For constructing its **breath-first (BFS) tree** forest **starting from vertex A**, you need to use a FIFO **queue** (note the *difference from DFS*) to trace the operation of breadth-first search, indicating the order in which the vertices {…, V’, V”, … } were visited. i.e., the order of the operation of **adding several vertices to**, or **removing a vertex from the queue** {,…, , , …}. The order in which vertices are added to the queue (i.e., enqueue operation) is the same order in which they are removed from it (i.e., dequeue operation). For this problem, you need to answer 4a through 4d, which are as follows:

4a. Show the traversal’s queue with a time-stamp indicating the order in which the vertices were visited (i.e., discover). What is the ordering of vertices yielded by the BFS?

4b. Construct the corresponding breadth-first search (BFS) tree forest, with an indication of tree edges and back edges, cross edges and forward edges.

4c. From the obtained BFS tree forest, compute the *shortest* *distance (smallest number of edges) from* A to vertex G. State all the paths with weights for each path generated by BFS.

4d. What are the time efficiency and space efficiency of the BFS?

Problem 5 [40 points]:

Approaches to obtaining solutions: Either use “programming”, “hands (manual) computation without writing program code”, or “a mix of programming and hands computation” to yield the solutions for this problem. Comments on your program code and program output are required.

Construct the string-matching automaton for the pattern P = aaabbaba over the alphabet Σ = {a, b, x |x is any letter other than a and b}; and illustrate its operation on the text string T = aaaabbabaaabbaaabbabaab.

5a. Construct the string-matching automation for the pattern P over the alphabet Σ = {a, b, x} in terms of the state transition table (Complete the state transition table)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | input | | | P | |
| state | a | b | x | |  |
| 0 |  |  |  | |  |
| 1 |  |  |  | |  |
| 2 |  |  |  | |  |
| 3 |  |  |  | |  |
| 4 |  |  |  | |  |
| 5 |  |  |  | |  |
| 6 |  |  |  | |  |
| 7 |  |  |  | |  |
| 8 |  |  |  | |  |

5b. Show the operation on the text string T, computed by the state transition table.

Complete the following table, in which T[i] is the letter at the position of i of the text string; and State Φ(T[i]) stands for the state transition Φ (s, T[i]) = s’.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | - | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| T[i] | - |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| State Φ(T[i]) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

5c. Complete the following sentence.

The result is \_\_\_\_\_\_ matches of the pattern at shift = \_\_\_\_ (i = \_\_\_) and shift = \_\_\_ (i = \_\_\_\_\_). Note that shift = i – 1.

5d. Draw a state transition diagram for a string-matching automaton for the pattern P over the alphabet Σ = {a, b, x |x is any letter other than a and b}.

**Do your own work. Provide enough detailed work to show how you get the solutions.**