CSUS Section\_03­\_

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE Name\_Santiago Bermudez\_\_

Department of Computer Science

CSc 130

Jinsong Ouyang

# DATA STRUCTURES AND ALGORITHM ANALYSIS

#### Final

Total Time – 120 Minutes

Closed Book – Closed Notes

### Total: 100

1. **(30%) Graph**

Part I (15 %)

Given the following network topology, fill out the table for constructing the least-cost paths to all network destinations from V3.

2

4

V13

V23

V73

V63

V53

V43

3

2

8

2

5

2

5

3

1

9

V33

|  |  |  |  |
| --- | --- | --- | --- |
| V | Known | Dv | Pv |
| V1 | ~~F~~T | ~~Inf~~ 0+4 | V3 |
| V2 | ~~F~~T | ~~Inf~~ 2+3 | V4 |
| V3 | ~~F~~T | ~~Inf~~ 0 |  |
| V4 | ~~F~~T | ~~Inf~~ 0+2 | V3 |
| V5 | ~~F~~T | ~~Inf~~ ~~2+5~~ 4+2 | ~~V4~~ V7 |
| V6 | ~~F~~T | ~~Inf~~ 0+3 | V3 |
| V7 | ~~F~~T | ~~Inf~~ 2+2 | V4 |

Part II (15 %)

Using V1 as the starting point, use Prim’s algorithm to fill out the table and draw the minimum spanning tree (MST).

2

4

V13

V23

V73

V63

V53

V43

3

2

8

2

5

2

5

3

1

9

V33

|  |  |  |  |
| --- | --- | --- | --- |
| V | Known | Dv | Pv |
| V1 | ~~F~~T | ~~Inf~~ 0 |  |
| V2 | ~~F~~T | ~~Inf~~ 0+2 | V1 |
| V3 | ~~F~~T | ~~Inf~~ ~~0+4~~ 1+2 | ~~V1~~ V4 |
| V4 | ~~F~~T | ~~Inf~~ 0+1 | V1 |
| V5 | ~~F~~T | ~~Inf~~ ~~1+5~~ 3+2 | ~~V4~~ V7 |
| V6 | ~~F~~T | ~~Inf~~ ~~1+8~~ 4+3 | ~~V4~~ V3 |
| V7 | ~~F~~T | ~~Inf~~ 1+2 | V4 |

1. **(20 %) Heap**
2. (5) Show the steps to add the value “16” to the following heap:

12

25 19

87 57 44 23

1. 1st, we add 16 as the left child of 87.
2. Then, we swap 16 with 87.
3. Then, we swap 16 with 25.

We should end up with:

12

/ \

16 19

/ / \

25 44 23

/

87

1. (5) How can the given binary heap including “16” be represented in an array?

0 1 2 3 4 5 6 7 8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 16 | 19 | 25 | 57 | 44 | 23 | 87 |  |

\*Just a formatting error above. I don’t have time to fix it. 12 is at index 0, 16 is at index 1, 19 is at index 2, and so on.

1. (10) Perform heapsort within the same array including “16” (without using additional array).

0 1 2 3 4 5 6 7 8

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Original array | 12 | 16 | 19 | 25 | 57 | 44 | 23 | 87 |  |
| After delete 12 | 16 | 25 | 19 | 87 | 57 | 44 | 23 |  |  |
| After delete 16 | 19 | 25 | 23 | 87 | 57 | 44 |  |  |  |
| After delete 19 | 23 | 25 | 44 | 87 | 57 |  |  |  |  |
| After delete 23 | 25 | 57 | 44 | 87 |  |  |  |  |  |
| After delete 25 | 44 | 57 | 87 |  |  |  |  |  |  |
| After delete 44 | 57 | 87 |  |  |  |  |  |  |  |
| After delete 57 | 87 |  |  |  |  |  |  |  |  |
| After delete 87 |  |  |  |  |  |  |  |  |  |

1. Write pseudocode for the mergesort algorithm (10%) Explain how you figure out the time complexity of the “conquer” part of the code? (5%)

Mergesort(leftend, rightend, array.length){

If(array.length <= 1){

ret

}

Mergesort(0, array.length/2)

Mergesort(array.length/2+1, array.length)

If(array[leftend]<array[rightend]){

Insert value from left array into array

Index++

}else{

Insert value from right array into array

Index++

}

}

For the “conquer” part of the code, we are not dividing the problem, but using instructions to put values into an index. All of the instructions for this that are used are constant, so we say that the time complexity of “conquer” is O(1).

1. Write pseudocode for the quicksort algorithm (10%) Accordingly to the algorithm, show how you select a pivot and do the partitioning till you get S1 and S2 (**only for the top level**) (5%) Explain how you figure out the time complexity of the partitioning of an array of size N? (5%)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 1 | 4 | 9 | 6 | 3 | 5 | 2 | 7 | 0 |

Quicksort(leftend,rightend){

If(array.length > 1){

Int pivot = array.length/2

Int leftend = 0

Int rightend = array.length

Int temp = array(array.length-1)

array(array.length-1) = pivot

array(array.length/2) = temp

}

While(array(leftend) <= array(pivot))

Leftend++

}

While(array(rightend) >= array(pivot))

Rightend--

}

If (array(leftend) <= array(pivot) && array(rightend) >= array(pivot)){

Int temp = array(leftend)

array(leftend) = array (rightend)

array(rightend) = temp

Leftend++

Rightend--

}

Quicksort(leftend,rightend)

}

When we select our first pivot, we divide the array length by 2 and use our result as the pivot. We divide 9 by 2, get 4 as the result and use the value at index 4 (which is 6 in this case) as our pivot. This is assuming we start counting by 0. If we divide 3 by 2, we use 1, ignoring the remainder. First, we move pivot 4 to the end and depending on the array and values in it, we move the left pointer to the right as long as the value it is pointing to is less than the pivot. We then move the right pointer to the left as long as the value it is pointing to is greater than the pivot. When we find a left-sided value that is greater than the pivot and a right-sided value that is less than the pivot, we swap those two values in the array and repeat the process described above. We stop when left pointer crosses over the right pointer and vice-versa. Once we stop, we swap 6 with an appropriate value in the array, which would be the value of 9 in this case to help keep our array “sorted”. We mark 6 as “done” and repeat for the subarray left of 6. If there is no left subarray, then we look at the right and do this recursively until all values are marked as done.

For the partitioning of an array of size N, we sort the array based on a pivot which gets marked as done afterwards and the next pivot we choose will be based of a smaller array to sort. Commands for choosing the pivot and swapping values are constant and thus negligible in the long run. We recursively choose pivots and swap values, ignoring values that have been marked as done and so the problem gets smaller over time. As such, I want to say that the time complexity of partitioning is O(logn).

1. Given a hash table which has 10 slots [0, 1, 2, …, 9] and hash function h(x) = x mod 10, insert the sequence of numbers, {6, 16, 26, 36, 56, 76, 25, 86}, into the hash table using separate chaining (5%) and double hashing technique with h2(x) = 7 – (x mod 7) (10%) respectively.

Diagram

Description automatically generated