Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2024

Problem Set 6

Due: 4:00 PM, Friday, April 12, 2024

You may work alone or in a group of size 2 or 3 on this assignment. However, in order to make sure you learn the material and are well-prepared for the exams, you should work through the problems on your own before discussing them with your partner, should you choose to work with someone. In particular, the "you do these and I'll do these" approach is sure to leave you unprepared for the exams.

All GitHub repositories must be created with all group members having write access and all group member names specified in the `README.md` file by 4:00 PM, Friday, April 5, 2024. This applies to those who choose to work alone as well!

There is a significant amount of work to be done here, and you are sure to have questions. It will be difficult if not impossible to complete the assignment if you wait until the last minute. A slow and steady approach will be much more effective.

Submitting

Please submit a hard copy (typeset preferred, handwritten OK but must be legible) for all written questions. Only one submission per group is needed.

Dynamic Programming Practice

Read Section 8.1 of Levitin to learn all about the robot coin collection problem. There, you will find a bottom-up dynamic programming solution to this problem.

Question 1: Complete the pseudocode below that uses a **recursive exhaustive search algorithm** (**not** dynamic programming) to solve the robot coin collection. (6 points)

```
ALGORITHM RobotCoinCollection($r, c, C$)
    //Input: $r, c$, starting row and column position
    //Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
    //Output: the max number of coins collected when the robot reaches $(n, m)$
```

Question 2: Rewrite your algorithm from above so that it uses **top-down dynamic programming** to get a recursive algorithm that is more efficient. (4 points)

```
ALGORITHM RobotCoinCollection($r, c, C, sols$)
    //Input: $r, c$, starting row and column position
    //Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
    //Input: $sols[1..n, 1..m]$ initialized to all -1
    //Output: the max number of coins collected when the robot reaches $(n, m)$
```

Question 3: Give the Θ efficiency class of your top-down dynamic programming algorithm and briefly explain how you arrived at this bound. (2 points)

Other Dynamic Programming Algorithms

Read about Warshall's algorithm and Floyd's algorithm in Levitin Section 8.4, then complete the two questions below.

Question 4: Complete Levitin Exercise 8.4.1. At each iteration of Warshall's Algorithm for this graph, show the graph both in adjacency matrix form and visually (vertices and edges). (5 points)

Question 5: Complete Levitin Exercise 8.4.7. At each iteration of Floyd's Algorithm for this graph, show the graph both in adjacency matrix form and visually (vertices and edges). (5 points)

Refresher on Heaps and Heapsort

For this part of the problem set, you will review and expand your understanding of the heap data structure and the heapsort algorithm. If you have not studied or do not recall the details of heaps and heapsort, please review Levitin Section 6.4 and/or the topic notes on Heaps linked from the schedule page.

Question 6: Consider the construction of a min-heap by starting with an empty heap, then inserting values in a given order. Show the min-heap after each of the values 2, 8, 97, 31, 12, 3, and 39 are added to an initially empty heap. (5 points)

Question 7: Consider the construction of a min-heap as done in the first phase of a heapsort. Start with an array containing the values 2, 8, 97, 31, 12, 3, and 39. Show the contents of the array (in array or tree form, your choice) after each non-trivial iteration of the first phase of the heapsort (which goes from an unsorted array to a min-heap). (6 points)

Question 8: Finally, show the transformation from the min-heap you just constructed to a sorted array, completing the heapsort procedure. (6 points)

Question 9: Consider a binary min-heap like the ones in Levitin Section 6.4. Which locations in a binary min-heap of n elements could possibly contain the third-smallest element? (2 points)

Question 10: Which locations in a binary min-heap of n elements could possibly contain the largest element? (1 point)

Generalized Heaps and Heapsort

The heaps you know from data structures and class, where the heap is represented by a binary tree stored in an array, are one specific case of a more general structure called a *d-heap*. In a d-heap, each node has up to d children. So the binary heaps you worked with in the previous section would be considered "2-heaps". For the questions below, assume that the minimum value is stored at the root node (*i.e.*, that it is a min-heap).

Question 11: Show the construction of a 2-heap that results from the following values being inserted: 18, 9, 23, 17, 1, 43, 65, 12. How many comparisons are needed? (4 points)

Question 12: Show the construction of a 3-heap that results from the following values being inserted: 18, 9, 23, 17, 1, 43, 65, 12. How many comparisons are needed? (4 points)

Question 13: For the heap element at position i in the underlying array of a 3-heap, what are the positions of its immediate chidren and its parent? (Give formulas in terms of i.) (1 point)

Question 14: For the heap element at position i in the underlying array of a d-heap, what are the positions of its immediate chidren and its parent? (Give formulas in terms of i and d.) (1 point)

Question 15: Show the construction of a 1-heap that results from the following values being inserted: 18, 9, 23, 17, 1, 43, 65, 12. How many comparisons are needed? (4 points)

Question 16: Show the construction of a 7-heap that results from the following values being inserted: 18, 9, 23, 17, 1, 43, 65, 12. How many comparisons are needed? (4 points)

You also reviewed heapsort in the previous section. In a sense, heapsort uses a 2-heap as an intermediate representation to sort the contents of an array. Let's consider a generalization of the heapsort idea. It's not exactly the same, since it uses a secondary data structure, but the general behavior is similar.

- First, insert the elements to be sorted into a priority queue (PQ).
- Then, remove the elements one by one from the PQ and place them, in that order, into the sorted array.

For heapsort, the PQ is a 2-heap, but any PQ implementation would work (naive array- or list-based with contents either sorted or unsorted, a d-heap, or even a binary search tree). Depending on which underlying PQ is used, the sorting procedure will proceed in a manner similar, in terms of the order in which comparisons occur, to one of the other sorting algorithms we have studied (*e.g.*, selection sort, quicksort, *etc.*).

Question 17: For each of the following underlying PQ structures, state which sorting algorithm proceeds in the manner most similar to the PQ-based sort using that PQ structure, and explain your answer. Each response should be at least a few sentences long, and should discuss how the pattern of comparisons and swaps, and the resulting efficiency relates to the sorting algorithm. (15 points)

1. 1-heap
2. 3-heap
3. (n-1)-heap
4. binary search tree
5. balanced binary search tree

Grading

This assignment will be graded out of 75 points.

Feature	Value	Score
Question 1	6	
Question 2	4	
Question 3	2	
Question 4	5	
Question 5	5	
Question 6	5	
Question 7	6	
Question 8	6	
Question 9	2	
Question 10	1	
Question 11	4	
Question 12	4	
Question 13	1	
Question 14	1	
Question 15	4	
Question 16	4	
Question 17	15	
Total	75	