CS 231, Spring 2024 A. Jamshidpey

ASSIGNMENT 2

Due date: June 19, 2024 16:00 (Waterloo time)

WARNING: To receive credit for this assignment, you checked "I agree" to the academic integrity declaration. Please keep all the rules in mind as you complete your work.

Coverage: Through Module 5

This assignment consists of a written component and a programming component. Please read the instructions on the reference page for assignments carefully to ensure that you submit each component correctly.

Please check the pinned FAQ in the discussion forum for corrections and clarifications.

Note: In assignment questions that specify the use of a particular paradigm, you are expected to come up with a new algorithm using that paradigm. It is not sufficient to implement a class example as a helper function and declare that the paradigm has been used. For example, using binary search or mergesort is not sufficient for a problem asking you to use divide-and-conquer.

Written component

For full marks, you are expected to provide a brief justification of any answer you provide.

W1. *[6 marks]* In this problem, you will develop a counterexample for a greedy algorithm for the problem Representative Sets, defined below:

Representative Sets

Input: A set $\mathcal{A}$ of sets of numbers

Output: A subset $\mathcal{B}$ of $\mathcal{A}$ such that

- the union of all the sets in $\mathcal{B}$ is equal to the union $\mathcal{U}$ of all the sets in $\mathcal{A}$, and
- the size of $\mathcal{B}$ is as small as possible

For example, if $\mathcal{A} = \{\{0,1\},\{2,3\},\{0\},\{1\},\{2\},\{0,3\},\{0,2\},\{1,2\}\}$, $\mathcal{B} = \{\{0,1\},\{2,3\}\}$ is a solution to the problem because $\mathcal{U} = \{0,1,2,3\}$ and the union of the sets in $\mathcal{B}$ is also $\{0,1,2,3\}$.

Consider the following algorithm for Representative Sets.

```
REPRESENTATIVE_SETS($\mathcal{A}$)
INPUT:     A set $\mathcal{A}$ of sets of numbers
OUTPUT:    A subset of $\mathcal{A}$
1  $\mathcal{B} \leftarrow \emptyset$
```

```
2  while there exists some $x \in \mathcal{U}$ not contained in the union of the sets in $\mathcal{B}$
3      Choose a set $S$ in $\mathcal{A}$ that contains $x$ and such that no larger set in
           $\mathcal{A}$ contains $x$
4      Add $S$ to $\mathcal{B}$
5  return $\mathcal{B}$
```

(a) *[3 marks]* Give an example of an input on which the algorithm produces the optimal solution, **for any order in which elements are considered in Line 2 and any way of breaking ties in Line 3**. Your example should contain at least four sets in $\mathcal{A}$ and at least four elements in $\mathcal{U}$. **Explain both how the algorithm obtains the output and why the output is an optimal solution.**

(b) *[3 marks]* Give an example of an input on which the algorithm **does not** produce the optimal solution, **for any order in which elements are considered in Line 2 and any way of breaking ties in Line 3**. Your example should contain at least four sets in $\mathcal{A}$ and at least four elements in $\mathcal{U}$. **Explain both how the algorithm obtains the output and why the output is not an optimal solution.**

W2. *[8 marks]* In this question, we consider a greedy algorithm for finding a *cluster* in a graph, as defined in Assignment 1. Here, we are ignoring the weights of the edges and simply trying to find a cluster containing the **largest number** of vertices possible.

Our algorithm will make use of the idea of k-*friendly* vertices, as defined in Assignment 1. We'll define the *friendliness* of a vertex as the maximum k for which it is k-friendly. For example, in Sample graph 3, the vertex with ID `"c"` has friendliness 2, since it is 2-friendly but not 3-friendly.

The algorithm will proceed as follows:

- Order the vertices in nonincreasing order (from large to small) by friendliness, breaking ties arbitrarily
- Initialize the cluster-so-far to the empty set
- For each vertex in order, add it only if it is a neighbour of all the vertices in the cluster-so-far
- After all vertices have been considered, return the cluster-so-far

In the algorithm, we consider a vertex to be a neighbour of all vertices in the empty set.

(a) *[2 marks]* What is the worst-case running time of the algorithm? Express your answer in Θ notation as a function of n, the number of vertices in the graph, m, the number of edges in the graph, and $g(n)$, the cost of the first step of the algorithm. Briefly justify your answer.

(b) *[2 marks]* Does the algorithm produce an optimal solution for Sample graph 4, no matter how ties are broken? Briefly justify your answer by showing how the algorithm behaves on the graph as well as discussing what the optimal solution is.

(c) *[4 marks]* Give an example of a graph on which the algorithm fails to produce the correct solution. Your graph must have an optimal solution of size at least 3, and cannot be any of the sample graphs. **Explain both how the algorithm obtains the output and why the output is not an optimal solution.**

W3. *[5 marks]* In this question, you will write a divide-and-conquer algorithm TOTAL that consumes a tree and the ID of a node in the tree and determines the sum of the weights of all nodes in the subtree rooted at the input node. For example, on Sample tree 3 and ID `"c"`, TOTAL will produce 15, summing the weights of the nodes with IDs `"c"`, `"e"`, and `"f"` . Using the tree operations provided on the reference page for `trees.py`, write a pseudocode function TOTAL that solves the problem.

W4. *[5 marks]* In this problem, you will develop recurrence relations for a divide-and-conquer algorithm MYSTERY written in pseudocode. The function call MYSTERY*(List, A, B)* processes the data items in positions A up to but not including B in the list *List*; the size of the instance should be calculated as B - A.

Provide recurrences using Θ notation, using $T(n)$ to represent the worst-case running time of MYSTERY on an input of size n (where $n = B$ - A). Refer to line numbers as needed, and make use the reference page on costs in your analysis.

Note: You are not required to solve the recurrences.

```
MYSTERY(List, A, B)
INPUT:      A nonempty list List of integers and positions
            A and B in List such that $A \leq B$
OUTPUT:     An integer
1   if A == B
2       return List[A]
3   else if A > B-5
4       return List[B]
5   else
6       FrontSplit ← A + (B - A) // 5
7       MidSplit ← A + 2 (B - A) // 5
8       BackSplit ← A + 4 (B - A) // 5
9       if List[A] == List[B]
10          return MYSTERY(List, A, FrontSplit)
11      else
12          Second ← MYSTERY(List, FrontSplit, MidSplit)
13          Third ← MYSTERY(List, MidSplit, BackSplit)
14          Fourth ← MYSTERY(List, BackSplit, B)
15          return Second + Third + Fourth
```

(a) *[1 mark]* Use a recurrence to express the base case or cases. Briefly justify your answer by referring to specific line numbers.

(b) *[4 marks]* Use a recurrence to express the general case. You may express your recurrence relation using $\leq$ instead of $=$, but be as precise as possible. Briefly justify your answer by referring to specific line numbers; explain which lines correspond to the divide, conquer, and combine stages of the algorithm.

W5. *[6 marks]* Use the substitution method to show that $T(n)$ is in $O(n^2 \log n)$ for $T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + 10n^2$ for $n \geq 4$, $T(2) = T(3) = 50$. *Hint: Be sure to come up with an explicit "guess" for the upper bound instead of trying to reason using order notation.*

Programming component

Please read the information on assignments and Python carefully to ensure that you are using the correct version of Python and the correct style. For full marks, you are required not only to have a correct solution, but also to adhere to the requirements of the assignment question and the style guide, including aspects of the design recipe.

Although submitting tests is not required, it is highly recommended that you test your code. For each assignment question, create a testing file that imports your submission and tests the code. Do not submit your testing file.

For any of the programming questions in this assignment, you may import any of the following files: `check.py`, `graphs.py`, `pair.py`, `graphs.py`, and `equiv.py` as well as built-in modules such as `math` and `copy`.

P1. *[10 marks]* In this question, you will implement the greedy algorithm for finding a cluster in a graph as described in W2.

Write a function `cluster_greedy` that consumes a graph and produces a cluster. For full marks, your function must implement the algorithm described in W2. Do not try to "fix" the algorithm: if the algorithm produces a cluster that is not the largest cluster in the graph, your function should do the same.

Submit your work in a file with the name `clustergreedy.py`.

P2. *[15 marks]* In this question, you will use divide-and-conquer to solve the problem ALIGN, as defined below:

ALIGN

Input: A sequence of distinct integers in increasing order

Output: Yes or no, answering "Is there a position p such that position p contains the value p?"

Write a function `align` that consumes a (possibly empty) Python list containing distinct integers in increasing order, and produces `True` if there exists a position p such that position p contains the value p and `False` otherwise. For example, `align([])` will produce `False`, as there are no values at all, `align([-2, 0, 2])` will produce `True`, as value `2` is in position 2, and `align([-3, 4, 6])` will produce `False`.

For the divide step, your algorithm should divide the list roughly in half. You will lose most of the marks for a less even divide step, and almost all of the marks if you do not use divide-and-conquer.

Submit your work in a file with the name `align.py`.