Final Examination 2021

Algorithms & Data Structures

IDATA 2302

Dec. 2021

This is the final examination of the algorithm an data structure course. It contains four parts, each having 5 questions worth 2.5 points. You have four hours.

1 Basic Knowledge

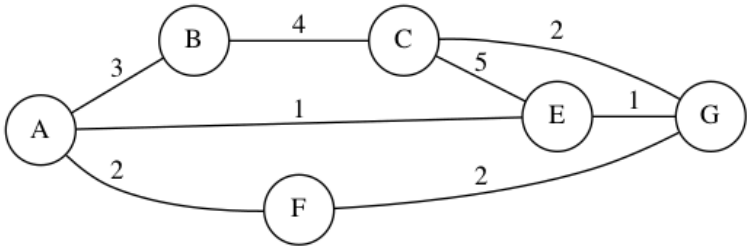
1. Explain why does a binary search runs in logarithmic time (i.e, in $O(\log n)$ where n represent the length of the array with search in).

2. An algorithm takes 1 ms to solve a problem of size 100. What is the biggest problem size that can be solved in 1 s if its runtime efficiency is $\Theta(n^2)$?

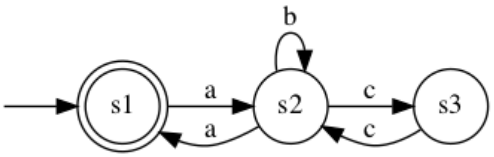
3. What binary search tree results from the following. Which element will end at the root of the tree? Why?

```
var tree = new BinarySearchTree();
tree.insert(25);
tree.insert(45);
tree.insert(46);
tree.insert(32);
tree.delete(25);
tree.insert(12);
tree.delete(45);
tree.insert(21);
```

4. Looking at the following graph, find one minimum spanning tree. Describe the tree as a list the edges which you would include, e.g, $(A, B), (B, C)$ and (C, D).



5. What is the transition table underpinning the following finite state automaton?



2 Algorithm Analysis

The LOGO language aims at providing children with a first "programming" experience. LOGO programs run on a turtle-like robot, which obeys basic instructions such as `FORWARD`, `BACKWARD`, `LEFT`, `RIGHT`, etc. Children can thus write short programs and see visually whether their turtle moves as they expect. The robot turtle also embeds a pen which can touch the ground and therefore draw a line as the turtle moves forward.

We look here at a program written in Java against a possible LOGO API. This API defines a `Robot` class equipped with the following methods:

- `forward(int distance)` The robot moves forward by the given distance (in cm). It takes a time proportional to the given distance, denoted by $time_F(d) = d$;
- `backward(int distance)` The robot moves backard by the given distance (in cm). It takes a time proportional to the given distance, denoted by $time_B(d) = d$.
- `turnLeft(int angle)` The robot spins on the left by given angle (from 0 to 360). This takes a time proportional to the given angle, denoted by $time_L(a) = a$.

- `turnRight(int angle)` The robot spins on the right by the given angle. This takes a time proportional to the given angle denoted by $time_R(a) = a$;
- `penUp()` The robot lifts the pen up so that it does not touch the ground anymore. This has no effect if the pen is already up and takes a constant amount of time denoted by $time_U$.
- `penDown()` The robot puts down its pen so that it marks the ground. This has no effect if the pen is already down and takes a constant time $time_D$.

A friend suggests the following program to draw regular polygons (i.e., triangle, square, pentagon, etc.) using the previous API. This exercise looks at the efficiency of this algorithm—regardless of its imperfection. Because the time spent performing arithmetic operations is negligible compared to the time spent moving the robot, we assume that arithmetic operations, logical operations and memory accesses all take 0 time.

```
void drawPolygon(Robot turtle, int sideCount, int sideLength) {
    var angle = 360 / sideCount;
    turtle.penDown();
    for (int side=0 ; side <= sideCount ; side++) {
       turtle.forward(sideLength);
       turtle.turnLeft(angle);
    }
    turtle.penUp();
}
```

1. How much time does it takes for this program to draw a square with 10 cm sides (i.e., invocation `drawPolygon(turtle, 4, 10)`).
2. Which parameters govern the time the robot takes to draw any given polygon, and why?
3. Estimate the time the robot needs to draw a given polygon whose number of side is s, and the side length is ℓ. Could you suggest an improvement?
4. Prove that your time estimate admits an upper bound such that $time(s, \ell) \in O(s \times \ell)$.
5. Given the API of this robot, do you think there exists an alternative algorithm that would admit a tighter upper bound? Why?

3 Algorithm Design

We now turn to the "rod cutting problem" where we are given a long piece of wood (the rod) which we must chop into segments. The particularity is that segments of different length are sold at different prices, and we would like to maximize our revenue. The problem is thus the following: Given the rod of a specific length, and given the selling prices of various length, how should we cut this rod to make as much money as possible?

Let us consider for example that we are given a rod of 4 meters, and that we can sell segments for the prices shown in the table below. Given these length and prices, we could cut the rod as follows:

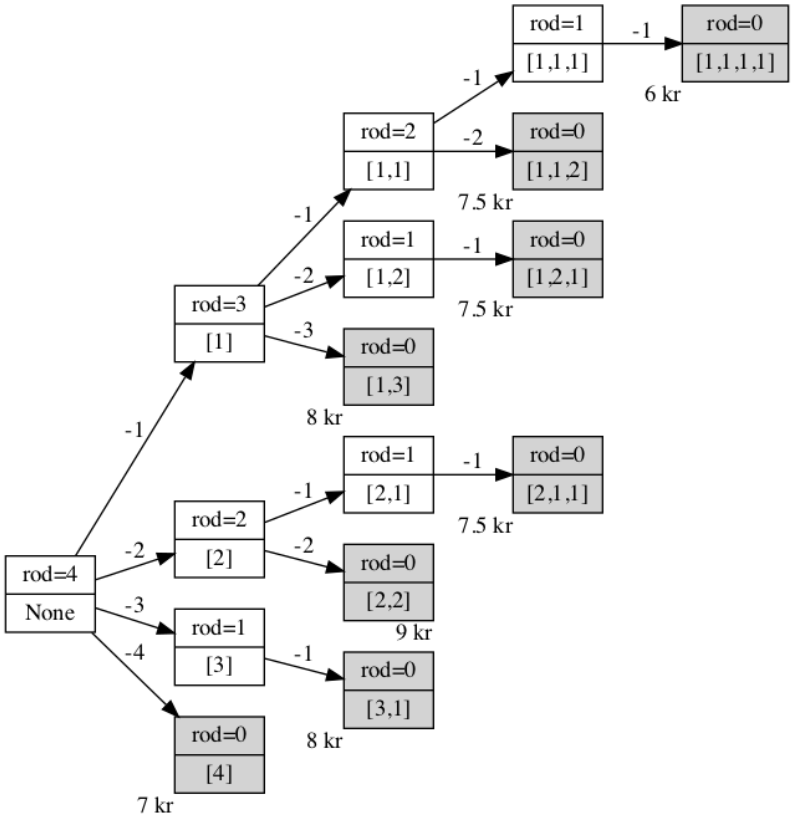
- 4 segments of 1 meter, which would yield a profit of $4 \times 1.5 = 6$ kr.
- 2 segments of 1 meter, and 1 segment of 2 meter, which would yield $2 \times 1.5 + 4.5 = 7.5$ kr.
- 2 segments of 2 meters, which would yield a revenue of $2 \times 4.5 = 9$ kr.
- 1 segment of 3 meters and 1 segment of 1 meter, which would yield a revenue $6.5 + 1.5 = 8$ kr.
- 1 segment of 4 meters for 7 kr.

Length (m)	1	2	3	4
Selling price (kr)	1.5	4.5	6.5	7

1. The following listing uses a "greedy" approach: It always maximizes the price per meter. In the previous example, we would always choose the segment of length 2, and then one segment of length 1 if the given length permits it. Show that this approach does not always yield the highest possible revenue, by giving a scenario (prices and total length) where it actually fails.

```
1  List<Integer> greedySolution(int rodLength, int[] prices) {
2      List<Integer> cuts = new ArrayList<Integer>();
3      var remaining = rodLength;
4      while (remaining > 0) {
5         var newCut = findHighestPricePerMeter(prices, remaining);
6          cuts.add(newCut);
7          remaining -= newCut;
8      }
9      returns cuts;
10 }
```

2. We propose to use a *combinatorial search* to enumerate all possible ways to cut the given rod, and find the most profitable one. We start with the original rod, and we explore all the possible segments we can cut. The following illustration portrays the associated search tree where each box denotes a configuration with its remaining rod length on top and the list of segments cut so far below.



What strategy would you use to prune this search tree, and in turn, find the most profitable strategy faster?

3. Can you see any *self-similar sub problems* in this tree? If yes, where and why?

4. Give a recursive algorithm to solve this *rod-cutting problem*. The recurrence formula is enough.

5. How would you improve the runtime efficiency of your recursive solution? Sketch a solution and motivate your decision.

4 Data Structure Design

Finally, let us look at the *two-sum* problem. Given an array of integer values, find the pair of values that sums up to a given value. For example, provided with the array $[1, 3, 6, 4, 2]$ and 10 as a target, the algorithm outputs $(6, 4)$. By contrast, provided with $[4, 2, 3, 9, 8]$ and 19, the algorithm raises an error since there is no pair that sums up to 19.

1. Propose an algorithm that uses a "brute force" approach.
2. What is its worst-case runtime efficiency? Explain your reasoning.
3. What data structure can help outperform this brute-force approach, and why?
4. Describe an algorithm that leverages this data structure.
5. What runtime efficiency do you obtain? Explain your reasoning.

End of the examination