Quiz 2

Adam Karl

October 16, 2020

1 Question 1 (20 points)

1.1 a. Recurrence

First sort I by a values (start values of the intervals)

This algorithm returns a tuple with (sum of gaps squared up to current point, number of intervals)

- Foo(i, R): //start location, remaining Intervals in consideration
  - //i = start location we're considering
  - if R is empty (no intervals are left in consideration), return $((L-1)^2, 0)$
  - //gap at the end = L-1, no intervals in solution
  - else
    - look at the first remaining interval. If we take it the overall minimum average squared length of the gaps is (Foo(b, R)[1] + a^2)/(Foo(b, R)[2] + 1)
    - for each R in recursive call, remove the chosen interval and any intervals that overlap with it
    - //in other words, take the solution from Foo(b) but add the gap before this interval, 1 interval to the count, then calculate the minimum average squared length of the gaps.
    - compare this solution to the solution from (Foo(b, R)[1])/(Foo(b, R)[2]), but where R only has the first interval removed (we're deciding not to put it in the solution)
    - choose to go with whichever solution minimizes the minimum average squared length of the gaps, and return the tuple of (minimum squared length of the gaps, num intervals) to match

Foo(0, I) returns a (minimum squared length of the gaps, num intervals) tuple, divide the first value by the second to get the optimal solution.

This algorithm compares solutions if we take a potential starting interval or not, and chooses the one that makes an optimal solution using recursion.

1.2 b. Dynamic Programming

A[i] will store the optimal solution from i to R. This solution is a tuple with (minimum squared length of the gaps, number of intervals)

First sort I by a values (start values of the intervals)

Create A as a 1D array with L+1 elements (from 0 to L)

- A[L] = (0,0) //no gap, no intervals
- for i from L to 0:
  - Looking at solutions for A[i], we must consider taking each possible interval (that starts at or after i), or not taking another interval.

- if there are no intervals that start after i, store $((L-i)^2, 0)$ in A[i]
- otherwise, if we take an interval (a,b), the minimum squared length of the gaps is A[b][0] + (a-i)^2 //solution starting at endpoint + gap from here to startpoint
- if we take an interval (a,b), the number of intervals is A[b][1] + 1 //solution starting at endpoint + 1
- calculate these values for ALL intervals that start at/after i (if any), and store the specifics for the one that produces the smallest average minimum squared length of the gaps (minimum squared length of the gaps/number of gaps) in A[i]

The solution C is found by looking at A[0] and dividing the (minimum squared length of the gaps) by (number of intervals in that solution) to get the minimum average squared length of the gaps

1.3 c. Explanation of Array

A[i] will store the optimal solution from i to R. This solution is a tuple with (minimum squared length of the gaps for solution from i to L, number of intervals in that solution)

Since A[i] is dependent on answers from A[i+1] to A[L], filling in the array from A[L] to A[0] will ensure all dependees are filled in before any elements that depend on them