COSC 3P03

Assignment 3

March 21st

1. To find the longest ascending sequence, you will need to duplicate the list S into S'. Order S' and perform the LCS algorithm on those 2 lists.

**Function**: Ci is the longest increasing subsequence to index i

{ 1 i = j

Ci { 1 + max{Cj|1<A(j)<A(i) and j<i } j < i

* the max of the longest path to j where the value of J < the value of i and the index of j is greater than i

**Principle of Optimality**: if the solution is the longest potential subsequence in the list, a subset of the list cannot have a longer subsequence, or a longer path to that space. If that were the case, then the initial determination that the sequence is the longest subsequence is false.

|  |
| --- |
|  |
|  | public int getLargestIncreasingPath() |
|  | { |
|  | for(int i = 0; i < list.length; i++){ |
|  | int local\_max = Integer.MIN\_VALUE; |
|  | for(int j = 0; j <= i; j++){ |
|  | int val = Integer.MIN\_VALUE; |
|  | if(i == j) val = 1; |
|  | else if(list[j] < list[i]) |
|  | { |
|  | val = 1 + count[j]; |
|  | } |
|  | if(val > local\_max) |
|  | local\_max = val; |
|  | } |
|  | count[i] = local\_max; |
|  | System.out.println("Longest increasing path to the value "+list[i]+" is: "+count[i]); |
|  | if(local\_max > overall\_max) overall\_max = local\_max; |
|  | } |
|  | return overall\_max; |
|  | } |

therefore the runtime is:

O(n2) since there 2 for loops whose worst case if O(n)

Question 2.

Coded

**Question 3.**

**Function:**

Ai = the cheapest path to get i miles by taking any number of busses along the way

**Recurrence:**

Ai {0 i<=0

{ min{Ak, Aj-(k+1) | S[e]} e<m

**Principle of Optimality**: if the path to i is optimal, that implies that it is the cheapest way to travel i miles. If you can split the path in 2, this implies that the paths are also optimal. If you can find a cheaper way to travel k or i-(k+1) miles then using those would lead to a more optimal solution to the entire path, making the initial claim incorrect.

**Justification:**

There are 2 nested for loops that go in their worst case from 0 to n, therefore O(n2)

**Question 4.**

Adding up n integers. Given a sequence of integers a1, a2, ..., an, and we want to compute a1 + a2 + ... + an.

a) Show how to compute all n 2 entries of S in time O(n2) by dynamic programming

S[i,j] is the sum of the values from index i to index j

**Recurrence:**

S[i,j] { Si i=j

{ Si,j-1 + Sjj i<j

To compute all n2 entries, you must build the table in the diagonal similar to how it is done in the matrix chain multiplication. To get the value for S[0][4] you would need to sum S[0][3] plus the value at S[4][4]. The values at s[0][0], S[1][1], etc, are all the actual value. You then use the computed values on the diagonal to compute the next value. Since we do not need to recurse through all potential solutions (the sum will always be the same since addition is associative) we only need 2 nested for loops to achieve this task.

**Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 7 | 2 | 4 | 5 | 3 |
| 7 | 7 | 9 | 13 | 18 | 21 |
| 2 | - | 2 | 6 | 11 | 14 |
| 4 | - | - | 4 | 9 | 12 |
| 5 | - | - | - | 5 | 8 |
| 3 | - | - | - | - | 3 |

b) Give a dynamic programming algorithm for computing the cheapest cost of adding n positive integers

**Cost Function**: A[i,j] is the cheapest cost to summate the digits i through to j. The cost of summating a number is the total of the summation.

**Principle of Optimality**: if the path from i to j is optimal, that implies that it is the cheapest way to add those numbers. If you can split the path in 2, this implies that the those additions are also optimal. If you can find a cheaper way to add i,k or k+1,j then swapping those would lead to a more optimal solution to the entire equation, making the initial claim incorrect.

**Recurrence**:

A[i,j] { 0 i >= j

{ Sij + min{Aik + AK+1,j} i < j

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 7 | 2 | 4 | 5 | 3 |
| 7 | 0 | 9 | 19 | 35 | 46 |
| 2 | - | 0 | 6 | 17 | 27 |
| 4 | - | - | 0 | 9 | 20 |
| 5 | - | - | - | 0 | 8 |
| 3 | - | - | - | - | 0 |

CheapestSumCost(P) P=<p0,p1,...pn>{

n = length[p] -1

for i = 1 to n

do m[i,i] = 0 //set this adding nothing to be 0

for L = 2 to n

for i = L to n-L+1 //iterate diagonally across the tree

j = i + L - 1 //set the end value

m[i,j] = inf //set the min value

for k = i to j-1 //iterate though the possible values

q = S[i,j] + m[i,k] + m[k+1,j] //the value of adding is min path value plus

// the cost for this addition

if q < m[i,j] //if it is a min value

m[i,j] = q //set it as the new value

}

O(n3) since there are 3 nested for loops that in their worst case goes from 0 to n