Stephen Chambers (smx227)

CS858

Assignment 6 Writeup

1. **Is there anything special that we should know when evaluating your implementation work?**

Everything is working correctly. The plots are attached.

1. **Exercise 15.3–2 in CLRS.**

Consider of the merge sort of the array {8 2 7 9 2 1 9 7 6 3 4 5 9 4 2 1] The recursive calls would look like the following:

[1 1 2 2 2 3 4 4 5 6 7 7 8 9 9 9]

[1 2 2 7 7 8 9 9] [1 2 3 4 4 5 6 9]

[2 7 8 9] [1 2 7 9] [3 4 5 6] [1 2 4 9]

[2 8] [7 9] [1 2] [7 9] [3 6] [4 5] [4 9] [1 2]

[8] [2] [7] [9] [2] [1] [9] [7] [6] [3] [4] [5] [9] [4] [2] [1]

Dynamic programming does not provide a speedup over merge-sort because the tree does not hold the "overlapping subproblems” property of dynamic programming. You cannot reuse nodes in the tree.

1. **(Those in 858 only) Exercise 15.4–4 in CLRS.**

*min(m,n) \* 2 space:*

To calculate the length of the LCS of an element at [*i*,*j*], the only information that is required are the nodes in the previous row ([i-1][j], [i][j-1], and [i-1,j-1]). Therefore, the algorithm would look like the following:

1. Fill out row1 (i = 0)
2. Fill out row2 (i = 1)
3. Fill out row3 starting at position [0,0] using the information in the second row(i = 1)
4. Fill out row4 starting at position [1,0], using the information in the first row(i = 0).

The row that we are filling in is constantly swapped between i = 0 and i = 1.

*min(m,n) space:*

The same strategy would apply, but we would store the information required by the parent (nodes [i-1][j], [i][j-1], and [i-1,j-1]) in the node itself.

1. **(Those in 858 only) Part a of problem 15–10 in CLRS.**

This assumes, as the text does, that we are making a massive assumption that minimizing risk is not a priority.

This problem holds the overlapping subproblems property. This fact allows us to prove the following:

1. The best investment to make at $1 is the one with the highest return. Let the highest return be *R*.
2. Due to the overlapping subproblems property, the best investment to make at $2 is the best investment to make at $1, twice.
3. Therefore, the best investment to make at $10,000 for a given year is 10,000*R*, where R is the investment with the highest return.

This first part of the proof only paints part of the picture, however. The problem states that a fee *f1* is incurred if the same investment is kept from year to year, and a fee *f2* is incurred if an investment is switched, and *f2 > f1*. In any particular year, there are two options.

1. An investment is switched if *R’*, the investment with the new highest return, minus *f2* is greater than *R*, the investment in the previous year minus *f1*.
2. The same investment is kept if *R – f1* is greater than *R’* – *f2*.

If there is not a benefit of switching, then the optimal investment strategy is to keep the current investment. If there is a benefit to switching, the optimal investment strategy is to move *all* the money to the new investment.

1. **Parts b and c of problem 15–10 in CLRS. (You may assume part a.)**

*Part b:*

1. The best investment to make at $1 is the one with the highest return. Let the highest return be *R*.
2. Due to the overlapping subproblems property, the best investment to make at $2 is the best investment to make at $1, twice.
3. Therefore, the best investment to make at $10,000 for a given year is 10,000*R*, where R is the investment with the highest return.

*Part c*:

*n =* number of investments

*y* = number of years investing

*d* = number of dollars available to invest

*investments* = stack[]

table[n][y] = []

returns[*n*][*y*] *The precomputed return values given for each investment per year*

for i = 0 to *n*

table[i][0] = 0

for j = 0 to *y*

table[0][j] = 0

/\* compute the table \*/

for i = 1 to *n*

for j = 1 to *y*

if(d\*r[i][j] – *f2* > d\*r[i][j-1] – *f1*) *Switch investments*

table[i][j].prev = table[i-1][j-1]

else

table[i][j].prev = table[i][j-1] *Keep the same investment*

table[i][j].investment = i

table[i][j].year = j

/\* return the investment plan \*/

cur = table[n][j]

while(cur.year != 0)

investments.push(cur.investment)

cur = cur.prev

return investments

1. **What suggestions do you have for improving this assignment in the future?**

No suggestions.