Austin Emery

Algorithms
HW 5
11/9/2017

Austin Emery
Algorithms hw5
11/9/17

1.
a) Determine and prove the optimal substructure of the problem and write a recursive formula of an optimal solution (i.e. define the variable that you wish to optimize and explain how a solution to computing it can be obtained from solutions to subproblems)

The goal of the algorithm is to determine the best route across the week to earn the most money. The best way to do this is by starting at the end of both arrays. Going by the rules we can create a subproblem that is comparing the earnings of the current low-stress and high-stress weeks.

```
max( l[n], h[n] );
```

Where max just returns the max of the two, l is low-stress, h is high-stress and n is the current week.

However, this is only the first subproblem. The next would be to determine which of the previous weeks are more lucrative. We can do this with recursive calls. But because high-stress jobs require the previous week to be no job, we need to “skip” over that week.

```
max( optSol(n - 1), optSol(n - 2) );
```

Where optSol is our function to determine best solution and n is the current week.

With this second algorithm we can see which prior weeks are better. Next we need to merge the two equations into:

```
max( l[n] + optSol(n - 1), h[n] + optSol( n- 2) );
```

This is the main algorithm that breaks the problem down into subproblems.

All we need now are the base cases:

```
if (n == 0) return max( h[n], l[n] );
```

This would occur if one of the subproblems reaches week 0, the first week. When this happens we just need to return the max of the two jobs for that week, because we can take the high-stress job without a side effect on the first week.

```
if (n == 1) return max( l[n] + optSol(n-1), h[n] );
```

This occurs when one of the subproblems reaches the second week. We would need to determine the current low-stress and the previous week's best price and compare that to the current high-stress to see which one is greater.

Putting all of these pieces together should yield the most optimal solution.

b) Write an algorithm that computes an optimal solution to this problem, based on the recurrence above. Implement your algorithm in C/C++ and run it on the following values:

	Week 1	Week 2	Week 3	Week 4
l	10	1	10	10
h	5	50	5	1

```cpp
//Algorithms Homework 5

#include <iostream>
#include <string>
#include <algorithm>

#define N 4

using namespace std;

int l[N] = {10, 1, 10, 10};
int h[N] = {5, 50, 5, 1};

int optSol(int);

int main ()
{
    cout << "Best Monies: " << optSol(N - 1) << endl;
}

//1. a)
//n is length of arrays - 1, l is lowstress array, h is highstress array
int optSol(int n)
{
    if (n == 0)
    {
        return max(h[n], l[n]);
    }
    if (n == 1)
    {
        return max(l[n] + optSol(n-1), h[n]);
    }

    return max(l[n] + optSol(n-1), h[n] + optSol(n-2));
}
```

```
austin@austin-IdeaPad-Y510P:~/algorithms/hw5$ ./hw5
Best Monies: 70
```

c)

```
10
11  int l[N] = {10, 1, 10, 10};
12  int h[N] = {5, 50, 5, 1};
13  int moneyForWeek[N] = {0, 0, 0, 0};
14  //int choiceForWeek[N] = {0, 0, 0, 0};
15  int total = 0;
16
17  int optSol(int);
18
19  int main ()
20  {
21      cout << "Best Monies: " << optSol(N - 1) << endl << "Best Path: ";
22      for (int i = 0; i < N; i++)
23      {
24          cout << moneyForWeek[i] << ' ';
25      }
26      cout << endl;
27  }
28  //n is length of arrays - 1, l is lowstress array, h is highstress array
29  int optSol(int n)
30  {
31      int tempHigh;
32      int tempLow;
33
34      if (n == 0)
35      {
36          moneyForWeek[n] = max(l[n], h[n]);
37          return moneyForWeek[n];
38      }
39      if (n == 1)
40      {
41          tempLow = optSol(n-1);
42          if (l[n] + tempLow > h[n])
43          {
44              moneyForWeek[n] = l[n];
45              //choiceForWeek[n] = -1;
46              return moneyForWeek[n] + tempLow;
47          }
48          else
49          {
50              moneyForWeek[n] = h[n];
51              moneyForWeek[n - 1] = 0;
52              //choiceForWeek[n] = 1;
53              //choiceForWeek[n - 1] = 0;
54              return moneyForWeek[n];
55          }
56      }
57      tempLow = optSol(n-1);
58      tempHigh = optSol(n-2);
59      total = max(l[n] + tempLow, h[n] + tempHigh);
60      if (l[n] + tempLow > h[n] + tempHigh)
61      {
62          moneyForWeek[n] = l[n];
63          //choiceForWeek[n] = -1;
64      }
65      else
66      {
67          moneyForWeek[n] = h[n];
68          moneyForWeek[n - 1] = 0;
69      }
70      return total;
71  }
```

```
austin@austin-IdeaPad-Y510P:~/algorithms/hw5$ ./hw5
Best Monies: 70
Best Path: 0 50 10 10
```

d)

```cpp
//Algorithms Homework 5

#include <iostream>
#include <string>
#include <algorithm>

#define N 4

using namespace std;

int l[N] = {10, 1, 10, 10};
int h[N] = {5, 50, 5, 1};
int moneyForWeek[N] = {0, 0, 0, 0};
int choiceForWeek[N] = {0, 0, 0, 0};
int total = 0;

int optSol(int);

int main ()
{
    cout << "Best Monies: " << optSol(N - 1) << endl << "Best Path: ";
    for (int i = 0; i < N; i++)
    {
        cout << moneyForWeek[i] << ' ';
    }
    cout << endl << "Best Choices: ";

    for (int i = 0; i < N; i++)
    {
        if (choiceForWeek[i] == -1)
        {
            cout << "Low ";
        }
        else if (choiceForWeek[i] == 0)
        {
            cout << "None ";
        }
        else
        {
            cout << "High ";
        }
    }
    cout << endl;
}
```

```
45  //n is length of arrays - 1, l is lowstress array, h is highstress array
46  int optSol(int n)
47  {
48      int tempHigh;
49      int tempLow;
50
51      if (n == 0)
52      {
53          if (l[n] > h[n])
54          {
55              moneyForWeek[n] = l[n];
56              choiceForWeek[n] = -1;
57          }
58          else
59          {
60              moneyForWeek[n] = h[n];
61              choiceForWeek[n] = 1;
62          }
63          return moneyForWeek[n];
64      }
65      if (n == 1)
66      {
67          tempLow = optSol(n-1);
68          if (l[n] + tempLow > h[n])
69          {
70              moneyForWeek[n] = l[n];
71              choiceForWeek[n] = -1;
72              return moneyForWeek[n] + tempLow;
73          }
74          else
75          {
76              moneyForWeek[n] = h[n];
77              moneyForWeek[n - 1] = 0;
78              choiceForWeek[n] = 1;
79              choiceForWeek[n - 1] = 0;
80              return moneyForWeek[n];
81          }
82      }
83      tempLow = optSol(n-1);
84      tempHigh = optSol(n-2);
85      total = max(l[n] + tempLow, h[n] + tempHigh);
86      if (l[n] + tempLow > h[n] + tempHigh)
87      {
88          moneyForWeek[n] = l[n];
89          choiceForWeek[n] = -1;
90      }
91      else
92      {
93          moneyForWeek[n] = h[n];
94          moneyForWeek[n - 1] = 0;
95          choiceForWeek[n] = 1;
96          choiceForWeek[n - 1] = 0;
97      }
98      return total;
99  }
```

```
austin@austin-IdeaPad-Y510P:~/algorithms/hw5$ ./hw5
Best Monies: 70
Best Path: 0 50 10 10
Best Choices: None High Low Low
```

Extra Credit:

a) If X and Y are sequences that both begin with the character A, every longest common subsequence of X and Y begins with A.

True

If both X and Y began with A then every longest common subsequence will also start with A. There would be nothing else to do but make the A the first character.

Examples:

X = A B C D
Y = A C E D
LCS = A C D

If we remove both or one of the A's then the LCS decreases by 1

X = B C D
Y = C E D
LCS = C D

So having similar characters at the start is somewhat of a freebie and choosing it for the start of the LCS does not have an effect on the rest of the LCS.

b) If X and Y are sequences that both end with the character A, some longest common subsequence of X and Y ends with A.

True

For pretty much the same reasons as above which are that choosing A for the last spot in the LCS does not affect the choices for the rest of the LCS, so it would be best to choose A to get the LCS.

Examples:

X = B C D A
Y = C E D A
LCS = C D A

If we remove both or one of the A's then the LCS decreases by 1

X = B C D
Y = C E D
LCS = C D