Patrick Austin
CS 477
Homework # 5
11/1/2016

1.

a. We wish to select A or B to ship with for each week over a period of i weeks. We wish to do so optimally by determining the selection of companies for each week that will minimize the total cost of hiring A and B over the period in question.

So an optimal schedule is defined as a list of A or B for each of the i weeks where the total amount paid to hire A and B given their contract costs is as low as possible (less than or equal to that paid for any other schedule).

As the prompt specifies, A costs rs_i to hire for one week, where r is A's rate per pound and s_i is the amount that must be shipped that week.

B costs a flat rate c to hire for one week, but B can only be contracted for four consecutive weeks.

So an optimal choice for week i is one where we choose between these two options, for whichever costs less:

rs_i + the optimal choices for all weeks starting at i-1

or

4c + the optimal choices for all weeks starting at i-4

If these values are equal then we can pick either option and still have an optimal schedule.

Additionally, before week 0, costs are 0.

So the recursive formula is along the lines of:

If i < 0, schedule(i) = 0.
Otherwise, schedule(i) = min(rs_i + schedule(i - 1) , 4c + schedule(i - 4))

b. Here is the specified program, schedule.cpp:

```
//part B: creates and prints table of subproblems and an optimal value for the
//optimization problem

//note: uses hard-coded problem size of 9 weeks for clarity of code.
//modification for i weeks should be self-explanatory

#include <iostream>
#include <cstdlib>
using namespace std;

//prototypes

int schedule (int rValue, int cValue, int* weights);

int main()
{

//declare specified sample values for testing the algorithm

//r*s[i] is the cost of shipping with A for one week
int r = 1;

//c is the cost of shipping with B for one week (but must book 4 consecutive weeks)
int c = 10;

//supplyValues[i] is the number of pounds that must be shipped for week i (9 weeks hard-coded)
int supplyValues[] = {11, 9, 9, 12, 12, 12, 12, 9, 9, 11};

//int result to denote total cost of the schedule
int result;

//run the algorithm on the sample values. will print table containing solutions to subproblems
result = schedule(r, c, supplyValues);

//spit out results to the console

cout << "Part B: An optimal schedule will cost " << result << "." << endl << endl;

//end program

return 0;

}
```

```
//implementations

int schedule (int rValue, int cValue, int* weights)
{

int i = 0;

//order of business: create table containing solutions to the subproblems.
//Will have 2 rows, optimalA and optimalB
//optimalA will contain the total cost at week i for choosing A,
//assuming optimal choices were made in the previous i-1 weeks.
int optimalA[9];
//optimalB will contain the total cost at week i for choosing B,
//assuming optimal choices were made in the previous i-1 weeks.
int optimalB[9];

//zero out the table

for ( i = 0; i <= 9; i++ )
    {
    optimalA[i] = 0;
    optimalB[i] = 0;
    }

//calculate values for the first week - base case, no subproblems
optimalA[0] = weights[0] * rValue;    //cost of A for week 0 is simply s[0] * r
optimalB[0] = 4*cValue;                 //cost of B for week 0 is simply 4c

//for each subsequent week, add the current week's choice +
//the optimal solution for the previous weeks

for ( i = 1; i <= 9; i++ )
    {
    if ( (i-4) < 0 )    //case to avoid accessing negative indices- refer to optB[0] instead of
optB[neg value]
         {
         optimalA[i] = std::min(optimalA[i-1], optimalB[0])//cost of choosing A is optimal prior +
s[i]*r
                              + (weights[i] * rValue);
         optimalB[i] = std::min(optimalA[i-1], optimalB[0])//cost of choosing B is optimal prior + 4*c
                              + (4 * cValue);
         }

    else                //normal case
         {
         optimalA[i] = std::min(optimalA[i-1], optimalB[i-4])//cost of choosing A is optimal prior +
s[i]*r
                              + (weights[i] * rValue);
         optimalB[i] = std::min(optimalA[i-1], optimalB[i-4])//cost of choosing B is optimal prior +
4*c
                              + (4 * cValue);
         }
    }
```

```
//table built, spit to console

cout << "Part B: Table of Subproblems" << endl;
cout << "Week   ";
for ( i = 0; i <= 9; i++ )
    cout << i << '\t';
cout << endl;
cout << "-------------------------------------------------------------------------------" << endl;
cout << "Choose A: ";
for ( i = 0; i <= 9; i++ )
    cout << optimalA[i] << '\t';
cout << endl << "Choose B: ";
for ( i = 0; i <= 9; i++ )
    cout << optimalB[i] << '\t';
cout << endl << endl;

//now, calculate the result

return std::min(optimalA[9], optimalB[9]);    //as expected, optimal result is min between
                                        //r*s[i] + opt(i-1) and 4*c + opt(i-4)
}
```

And here is a printout of the output for the specified values:

Part B: Table of Subproblems

Week	0	1	2	3	4	5	6	7	8	9
Choose A:	11	20	29	41	52	63	72	78	87	98
Choose B:	40	51	60	69	80	91	100	109	118	127

Part B: An optimal schedule will cost 98.

c. Here is the specified program, schedule_1.cpp:

```
//new in part C: schedule method generates and prints an auxillary
//array that will be used in part D to display an optimal solution.

//note: uses hard-coded problem size of 9 weeks for clarity of code.
//modification for i weeks should be self-explanatory

#include <iostream>
#include <cstdlib>
using namespace std;

//prototypes

int schedule (int rValue, int cValue, int* weights, char* aux);

int main()
{

//declare specified sample values for testing the algorithm

//r*s[i] is the cost of shipping with A for one week
int r = 1;

//c is the cost of shipping with B for one week (but must book 4 consecutive weeks)
int c = 10;

//supplyValues[i] is the number of pounds that must be shipped for week i (9 weeks hard-coded)
int supplyValues[] = {11, 9, 9, 12, 12, 12, 12, 9, 9, 11};

//after the algorithm is finished, choices[i] will contain A to denote selecting A is optimal for week
i,
//or B to denote selecting B is optimal- will be used in part D
char solution[9];

//int result to denote total cost of the schedule
int result;

//run the algorithm on the sample values. will print table containing solutions to subproblems

result = schedule(r, c, supplyValues, solution);

//spit out final result to the console

cout << "Part B: An optimal schedule will cost " << result << "." << endl << endl;

//end program

return 0;

}
```

```
//implementations

int schedule (int rValue, int cValue, int* weights, char* aux)
{

int i = 0;

//order of business: create table containing solutions to the subproblems.
//Will have 2 rows, optimalA and optimalB
//optimalA will contain the total cost at week i for choosing A,
//assuming optimal choices were made in the previous i-1 weeks.
int optimalA[9];
//optimalB will contain the total cost at week i for choosing B,
//assuming optimal choices were made in the previous i-1 weeks.
int optimalB[9];

//zero out the table

for ( i = 0; i <= 9; i++ )
    {
    optimalA[i] = 0;
    optimalB[i] = 0;
    }

//calculate values for the first week - base case, no subproblems
optimalA[0] = weights[0] * rValue;    //cost of A for week 0 is simply s[0] * r
optimalB[0] = 4 * cValue;        //cost of B for week 0 is simply 4c

//for each subsequent week, add the current week's choice +
//the optimal solution for the previous weeks

for ( i = 1; i <= 9; i++ )
    {
    if ( (i-4) < 0 )    //case to avoid accessing negative indices- refer to optB[0] instead of
optB[neg value]
        {
        optimalA[i] = std::min(optimalA[i-1], optimalB[0])     //cost of choosing A is optimal prior +
s[i]*r
                              + (weights[i] * rValue);
        optimalB[i] = std::min(optimalA[i-1], optimalB[0])     //cost of choosing B is optimal prior +
4*c
                              + (4 * cValue);

        if ( optimalA[i-1] < optimalB[0] )      //if optimalA[i-1] < optimalB[0], A was chosen for
week i-1
                aux[i-1] = 'A';                         //store A at i-1 in aux
        else
                aux[i-1] = 'B';                         //otherwise, store B
        }
```

```
    else                //normal case
         {
         optimalA[i] = std::min(optimalA[i-1], optimalB[i-4])   //cost of choosing A is optimal prior +
s[i]*r
                              + (weights[i] * rValue);
         optimalB[i] = std::min(optimalA[i-1], optimalB[i-4])     //cost of choosing B is optimal prior
+ 4*c
                              + (4 * cValue);

         if ( optimalA[i-1] < optimalB[i-4] )           //if optimalA[i-1] < optimalB[i-4], A was
chosen for week i-1
                 aux[i-1] = 'A';                        //store A at i-1 in aux
         else
                 aux[i-1] = 'B';                        //otherwise, store B
         }

    }

//now, calculate the last solution value

if ( optimalA[9] < optimalB[9] )        //calculate whether A or B is chosen for the final week
    aux[9] = 'A';
else
    aux[9] = 'B';

//tables built, spit to console

cout << "Part B: Table of Subproblems" << endl;
cout << "Week   ";
for ( i = 0; i <= 9; i++ )
    cout << i << '\t';
cout << endl;
cout << "-------------------------------------------------------------------------------------" << endl;
cout << "Choose A: ";
for ( i = 0; i <= 9; i++ )
    cout << optimalA[i] << '\t';
cout << endl << "Choose B: ";
for ( i = 0; i <= 9; i++ )
    cout << optimalB[i] << '\t';
cout << endl << endl;

cout << "Part C: Auxilliary table: Value chosen for the optimal solution" << endl;
cout << "Week   ";
for ( i = 0; i <= 9; i++ )
    cout << i << '\t';
cout << endl;
cout << "-------------------------------------------------------------------------------------" << endl;
cout << "Chosen:   ";
for ( i = 0; i <=9; i++ )
    cout << aux[i] << '\t';
cout << endl << endl;

return std::min(optimalA[9], optimalB[9]);    //as expected, optimal result is min between
                                        //r*s[i] + opt(i-1) and 4*c + opt(i-4)
```

```
}
```

And here is a printout of the result on the specified array:

Part B: Table of Subproblems

Week	0	1	2	3	4	5	6	7	8	9
Choose A:	11	20	29	41	52	63	72	78	87	98
Choose B:	40	51	60	69	80	91	100	109	118	127

Part C: Auxilliary table: Value chosen for the optimal solution

Week	0	1	2	3	4	5	6	7	8	9
Chosen:	A	A	A	B	B	B	B	A	A	A

Part B: An optimal schedule will cost 98.

d. Here is the specified program, schedule_2.cpp

```
//new in part D: recursive method printOpt that walks back through
//the given aux array to print an optimal solution to the console

//note: uses hard-coded problem size of 9 weeks for clarity of code.
//modification for i weeks should be self-explanatory

#include <iostream>
#include <cstdlib>
using namespace std;

//prototypes

int schedule (int rValue, int cValue, int* weights, char* aux);

void printOpt(char* solution, int size);

int main()
{

//declare specified sample values for testing the algorithm

//r*s[i] is the cost of shipping with A for one week
int r = 1;

//c is the cost of shipping with B for one week (but must book 4 consecutive weeks)
int c = 10;

//supplyValues[i] is the number of pounds that must be shipped for week i (9 weeks hard-coded)
int supplyValues[] = {11, 9, 9, 12, 12, 12, 12, 9, 9, 11};

//after the algorithm is finished, choices[i] will contain A to denote selecting A is optimal for week
i,
//or B to denote selecting B is optimal
char solution[9];

//int result to denote total cost of the schedule
int result;

//run the algorithm on the sample values. will print table containing solutions to subproblems

result = schedule(r, c, supplyValues, solution);

//using the solution array, recursively print out the optimal schedule
cout << "Part D: The optimal solution consists of these choices for these weeks:" << endl;
printOpt(solution, 9);
cout << endl << endl;

//spit out final result to the console

cout << "Part B: An optimal schedule will cost " << result << "." << endl << endl;

//end program
```

```
return 0; }

//implementations

int schedule (int rValue, int cValue, int* weights, char* aux)
{

int i = 0;

//order of business: create table containing solutions to the subproblems.
//Will have 2 rows, optimalA and optimalB
//optimalA will contain the total cost at week i for choosing A,
//assuming optimal choices were made in the previous i-1 weeks.
int optimalA[9];
//optimalB will contain the total cost at week i for choosing B,
//assuming optimal choices were made in the previous i-1 weeks.
int optimalB[9];

//zero out the table

for ( i = 0; i <= 9; i++ )
    {
    optimalA[i] = 0;
    optimalB[i] = 0;
    }

//calculate values for the first week - base case, no subproblems
optimalA[0] = weights[0] * rValue;    //cost of A for week 0 is simply s[0] * r
optimalB[0] = 4 * cValue;        //cost of B for week 0 is simply 4c

//for each subsequent week, add the current week's choice +
//the optimal solution for the previous weeks

for ( i = 1; i <= 9; i++ )
    {
    if ( (i-4) < 0 )    //case to avoid accessing negative indices- refer to optB[0] instead of
optB[neg value]
        {
        optimalA[i] = std::min(optimalA[i-1], optimalB[0])     //cost of choosing A is optimal prior +
s[i]*r
                              + (weights[i] * rValue);
        optimalB[i] = std::min(optimalA[i-1], optimalB[0])     //cost of choosing B is optimal prior +
4*c
                              + (4 * cValue);

        if ( optimalA[i-1] < optimalB[0] )      //if optimalA[i-1] < optimalB[0], A was chosen for
week i-1
                aux[i-1] = 'A';                         //store A at i-1 in aux
        else
                aux[i-1] = 'B';                         //otherwise, store B
        }
```

```
else             //normal case
        {
        optimalA[i] = std::min(optimalA[i-1], optimalB[i-4])   //cost of choosing A is optimal prior +
s[i]*r
                                + (weights[i] * rValue);
        optimalB[i] = std::min(optimalA[i-1], optimalB[i-4])     //cost of choosing B is optimal prior
+ 4*c
                                + (4 * cValue);

        if ( optimalA[i-1] < optimalB[i-4] )            //if optimalA[i-1] < optimalB[i-4], A was
chosen for week i-1
                aux[i-1] = 'A';                         //store A at i-1 in aux
        else
                aux[i-1] = 'B';                         //otherwise, store B
        }

    }

//now, calculate the last solution value

if ( optimalA[9] < optimalB[9] )        //calculate whether A or B is chosen for the final week
    aux[9] = 'A';
else
    aux[9] = 'B';

//tables built, spit to console

cout << "Part B: Table of Subproblems" << endl;
cout << "Week   ";
for ( i = 0; i <= 9; i++ )
    cout << i << '\t';
cout << endl;
cout << "-------------------------------------------------------------------------------------" << endl;
cout << "Choose A: ";
for ( i = 0; i <= 9; i++ )
    cout << optimalA[i] << '\t';
cout << endl << "Choose B: ";
for ( i = 0; i <= 9; i++ )
    cout << optimalB[i] << '\t';
cout << endl << endl;

cout << "Part C: Auxilliary table: Value chosen for the optimal solution" << endl;
cout << "Week   ";
for ( i = 0; i <= 9; i++ )
    cout << i << '\t';
cout << endl;
cout << "-------------------------------------------------------------------------------------" << endl;
cout << "Chosen:   ";
for ( i = 0; i <=9; i++ )
    cout << aux[i] << '\t';
cout << endl << endl;

return std::min(optimalA[9], optimalB[9]);    //as expected, optimal result is min between
                                        //r*s[i] + opt(i-1) and 4*c + opt(i-4)
}
```

```
void printOpt(char* solution, int size)          //recursively steps through the solution array,
printing the optimal schedule
{

if ( size == 0 )
    {
    cout << "Week 0: " << solution[0] << '\t';    //base case: don't make another call, just print the
value
    return;
    }

else if ( size - 4 < 0 )                         //case to avoid accessing negative indices
    {
    if (solution[size] == 'A')                   //if A, call for the previous week and print A
         {
         printOpt(solution, size - 1);
         cout << "Week " << size << ": " << 'A' << '\t';
         }
    if (solution[size] == 'B')                   //if B, print Bs for the remaining values (also a base
case)
         {
         for ( int i = 0; i <= size; i++ )
                 cout << "Week " << i << ": " << 'B' << '\t';
         }
    }

else
    {

    if (solution[size] == 'A')
         {
         printOpt(solution, size - 1);
         cout << "Week " << size << ": " << 'A' << '\t';
         }
    if (solution[size] == 'B')
         {
         printOpt(solution, size - 4);    //if B, call for 4 weeks previous and print 4 Bs
         cout << "Week " << size-3 << ": " << 'B' << '\t';
         cout << "Week " << size-2 << ": " << 'B' << '\t';
         cout << "Week " << size-1 << ": " << 'B' << '\t';
         cout << "Week " << size   << ": " << 'B' << '\t';
         }
    }
}
```

And here is a printout of the result on the specified array:

Part B: Table of Subproblems

Week	0	1	2	3	4	5	6	7	8	9
Choose A:	11	20	29	41	52	63	72	78	87	98
Choose B:	40	51	60	69	80	91	100	109	118	127

Part C: Auxilliary table: Value chosen for the optimal solution

Week	0	1	2	3	4	5	6	7	8	9
Chosen:	A	A	A	B	B	B	B	A	A	A

Part D: The optimal solution consists of these choices for these weeks:
Week 0: A Week 1: A Week 2: A Week 3: B Week 4: B Week 5: B Week 6: B
Week 7: A Week 8: A Week 9: A

Part B: An optimal schedule will cost 98.

3.

a. True. Consider strings X' and Y', which are X and Y, each without its leading 'A'. The LCS of X and Y will be 'A' followed by the largest common substring of X' and Y'. Therefore an LCS of X and Y will always contain A.

b. True. By similar logic to above, can consider strings X' and Y' which are X and Y, each without its ending 'A'. The LCS of X and Y will be the LCS of X' and Y' followed by A. Therefore an LCS of X and Y will always contain A.