Algorithm HW2 Chang Liu chang_liu@student.uml.edu

1. Solution:

Let's assume that Sij be the interval between activity ai and aj, so we could have the following equations:

According to the definition, there are two cases for c[i,j] value:

1) c[i, j] = 0 whenever Sij = 0
2) c[i,j] =max{c[i,k]+c[k,j]+1} if Sij ≠ 0

Now that, we build two arrays s and f, containing the start point and end point for each of the activity. Assume that we have n activity, then we could have n elements for each of the array s and f, s = {s0, s1, …. s(n-1)}, f = {f0, f1, … f(n-1)}.

Then by using dynamic algorithm, we could get the following algorithms:

```
DYNAMIC_ACTIVITY_TABLE(s, f)
n = s.length
// initialize the c[] vector, size is n * n
for i= 0 to n
    for j = i to 0
    c[i,i] = -1
    c[j,i] = 0

   // general case
    if (i - 1) > 1
        // compute c[i][j] and store activity number
        for k=j+1 to i -1
            if f[j] <= s[k] && f[k] <= s[i]
                if c[i][j] < (c[i][k] + c[k][j] + 1)
                    c[i][j] = c[i][k] + c[k][j] + 1
                    c[j][i] = k
return c
```

This algorithm returns the n * n array c, which records the information about the activity number in the diagonal below. For each c[i][j] we have to compute the ak in Sij(k is between i and j, so that we could know which activity to select from).

Then we could form the selection of activity by using the above table c

```
SUB_DYNAMIC_SELECTION(c, s, f, i, j)
    if i < j && j < c.length
        k = c[i][j]
        if i <k && k < j
            return SUB_DYNAMIC_SELECTION(c, s, f, i, k) U { ak} U
SUB_DYNAMIC_SELECTION(c, s, f, k, j)  /// add a[k] activity to the sets as solution

ALL_DYNAMIC_SELECTION(c, s, f, i, j)
    return {ai} U SUB_DYNAMIC_SELECTION(c,s,f,i,j) U {aj} // add a[i] and a[j] activity
```

So the running time is O(n2). As the textbook shows, the running time of GREEDY-ACTIVITY-SELECTOR is O(n), so dynamic algorithm is more time-consuming, and greedy algorithm is more efficient and it could also give us good result relatively.

2. Solution:

Here I would still use greedy algorithm to calculate the selection.

According to the description,
  1) $w1 < w2 < w3 \dots. < wn$,
  2) $v1 > v2 > v3 \dots. > vn$

As they are in the same order, so the above equation is correct. Then we should calculate the ratio v/w, and place them in an order.

Assume for any i and i+1 element, we should easily compare that $vi/wi > vi+1/wi+1$, because of the mathematical theory. So we could conclude that the ratio is still the same order.

In the textbook, when using the greedy algorithm to solve the partial knapsack problem, we also calculate the ratio and select the highest one from the beginning. So here as it is a 0-1 knapsack problem, and the ratio has been calculated, I need to select the ratio which is the highest and then the second highest …..

So in this case, select the items with the smallest weight(equals to select the item with the highest value/weight ratio), and it gives the best solution in greedy algorithm.

Prove by contradiction:
Suppose that we have select the items according to the algorithm I said above, the items are w1, w2, …. wk, these k items are select by their weight from the smallest to larger one. So in the next step, let assume that it is not optimal, there exist another item wj, which could replace wi in the previous sequence and gives an optimal solution.

So $j > k$, because our algorithm is from smallest, so not selected wj must be larger then selected one.

$wj > wi$, and $vj < vi$, so the total value is now:
  v1+v2+..+vi-1 + vj + vi+1+..+ vk (here replace i with j element)
and the total before is
  v1+v2+…+vi-1+vi+vi+1+….vk

To calculate their delta value and we find the new value is less than before one, so it results in a worse result, rather than a better result. And it contradicts with our assumption.

In this way, we could conclude that our proposed approach is the best solution for this problem.

3. Solution:

The solution is by greedy algorithm. The professor should go to the farthest location within m miles before running out of water, then fill the water up, and goes to the next station by the same strategy. By using this way, it could guarantee that the stops are the least and the times for filling water is also the smallest.

Looked at another way, at each gas station, Professor Midas should check whether he can make it to the next gas station without stopping at this one. If he can, skip this one. If he cannot, then fill up. Professor Midas doesnít need to know how much gas he has or how far the next station is to implement this approach, since at each fillup, he can determine which is the next station at which heíll need to stop.

We could know that at each water station he passed, the professor should check whether he can make it to the next water station with stopping. If he can, then go directly to the next location, otherwise he need to stop at this location and fill the water up.

This problem has optimal substructure. Suppose there are m possible water stops. Consider an optimal solution with s locations and whose first stop is at the kth station. Then the rest of the optimal solution must be an optimal solution to the subproblem of the remaining (m – k) locations. Otherwise, if there were a better solution to the subproblem, i.e., one with fewer than s – 1 stops, we could use it to come up with a solution with fewer than s stops for the full problem, contradicting our supposition of optimality.

And we could also know that it meets the greedy-choice property:"We can make whatever choice seems best at the moment and then solve the subproblems that arise later." Each subproblem is also generated by its optimal selection. Suppose for the first stop kth, if there are other stops which could also make it before running out of water, and the professor select that one, it also gives another solution. However, as we said in last paragraph, the kth stop is the farthest one under the condition of meeting water constrains. So if selecting another stop, it will give less traveling distance and increase the total number of water stops, in this case, we still prefer to select the kth stop.

If there are n locations in the state map, then the running time is O(n), because the professor just need to scan all the stations and find the next target of stops when moving from one place to another.

4. Solution:

First, let set up the description in the mathematical format, we could say that there 256 characters appearing in this data file, and each character has its own frequency f1, f2, … f256, let's sort the frequency by increasing, then f1 < f2 < f3 < …< f256

In the fixed-length scheme, each character is represented by a fixed-length of 8 to represent the 256 characters, so the total cost overall is:

total1 = 8(f1+f2+f3+…+f256)

In the Huffman coding scheme, each character is represented by a not-fixed-length with maximum length is 8, let say there are 8 possibility of length, from 1 to 8, here we use x1, x2, x3, … x8 to represent the length 1, 2, 3, … 8, according to the Huffman coding's theory, the lesser the frequency, the longer length is should have, so the total cost is:

$$total2 = x8*f1 + x8*f2 + \ldots + x1 * f256$$

Note that there're 256 characters in total, so in the above equation some characters share the same length, like f100, f101 both share the length of x5(=5).

Then according to the constrains, $f256 < 2 * f1$, so we need to calculate the delta of total1 and total2.

Now that we need to consider the construction of Huffman code tree, we first select two least frequency character, here as $f1 < f2 < \ldots < f256$, so we select character1 and character2 to form a new node, calculate the sum of f1+f2.

According to first constrains, $2 * f1 > f256$, so

$$f1 + f2 > f1 + f1 > f256 > f255 > f254 > \ldots > f3$$

In this situation, the new node(1,2)'s frequency is bigger than any other nodes, so still form the new node by using node3 and node4, node5 and node6, … node255 and node256, combine each of the neighborhood nodes.

In the next step, still connect the two neighborhood nodes in the upper layers, so the tree should look like this:

```
                              …          ….
                    node(1,2,3,4)
                         |                  …..
                      |      |                       ….
                |                |                         ….
            node(1,2)          node(3,4)
              |                  |                             …..
          |      |             |    |
       |            |        |        |                              ….
  node1       node2       node3   node4 ….                   node255   node256
```

So we combine the neighborhood nodes from bottom to top, and as 256 = power(2, 8), it is an even number, all layers have the same coding length, because in the last length is log(2, 256) = 8, each node's coding length is also 8 in this case, it is a special binary tree, and it is a balanced binary tree.

At last, according the constrains of the frequency, we get the whole coding scheme and know that even by using Huffman code, for this condition, each node's coding length is still the same, aka they all have the coding length of 8, so the new cost is also:

$$total2 = 8(f1+f2+\ldots+f256) \text{ ====> } total2 = total1$$

In this sense, we have proven that they share the same efficiency and Huffman coding in this case is no more efficient than using an ordinary 8-bit fixed-length code.