Ositadinma Arimah
oarimah@uwo.ca
CS 3340B Assignment 3
250 981 235

1. Π=

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Next	0	0	1	1	2	3	4	5	6	7	8	9	2	3	4	5	6	7

2.

```
MODIFIED-KMP-PREFIX(P,T)
    m = T.length
    let π[1..m] be a new array
    π[1]=0
    k=0
    for q=2 to m
        while k>0 and P[k+1]!=T[q]
            k = π[k]
        if P[k+1]==T[q]
            k = k + 1
        π[q] = k
    return π
```

Knuth-Morris-Pratt algorithm can be modified to help find the longest prefix string between two strings as it runs in O(m+n) time.

When finding each matching prefix, we keep track of the length and store the length and location of the longest prefix at each turn of the next table used in the Knuth-Morris-Pratt algorithm or update the length and location if there is a new longest prefix with a longer length than the one stored. However, at the end, if we find that both strings are the same or that the first string contains the second string, we just return the found string as this is the longest prefix.

The algorithm is correct because it functions the same way as traditional KMP next table generation and provides valid prefix matching values in the same way.

The time complexity is O(m), where m is the length of the text, because the original Compute-Prefix-Function is O(n), where n is the length of the pattern, and no changes were made that would alter the time complexity. As such, the modified algorithm takes O(m+m) = O(m) time.

3.

Explanation

In order to reconstruct the LCS from the completed c table without using the b table, I will design an algorithm- reconstructLCS(c,X,Y,i,j), which contains the parameters that include: the completed c table and the original X and Y sequences and the i and j index.

The algorithm reconstructLCS(c,X,Y,i,j) will print the LCS of X and Y from the completed table by only computing the required entries of B during compilation time. Consequently, this means that the b table will not be used, as stated in the problem.
This algorithm will run in O(m + n) time. This is because each iteration of the while loop decrements either i(i--) or j(j--) or both by 1, and the program is finished with a return statement when either reaches 0.
Therefore, O(m+n) is an accurate time complexity, as program will never run faster than this said time.

Pseudocode

```
reconstructLCS(c, X, Y, i, j)
do c[i, j] == 0
    return
while (c[i, j] != 0)
  if X[i] == Y[j]
    reconstructLCS(c, X, Y, i - 1, j - 1)
    print X[i]
  else if c[i - 1, j] > c[i, j - 1]
    reconstructLCS(c, X, Y, i - 1, j)
  else
    reconstructLCS(c, X, Y, i, j - 1)
```

4. The greedy solution solves this problem optimally. We must maximize the distance we can cover from a specific point, such that, there is still a place to get water before we run out.
At each refilling location, Professor Gekko should check whether he can make it to the next refilling location without stopping at this one. If he can, he should skip this one. If he cannot, then he should fill up.

This problem has optimal substructure. Proof:

- Let O be any optimal solution which has the professor stop at positions o1,o2,..,on.
- Let g1 be the furthest stopping point we can reach from the starting point.
- Create a modified solution G: o2-o2<o2-g1.
- This modified solution means we can actually make it to the positions in *G* without running out of water.

Since G has the same number of stops, we conclude that g1 is contained in some optimal solution.

If there are n refilling locations on the map, Professor Gekko needs to inspect each one just once. Therefore, the time complexity of this strategy is O(n).

Ositadinma Arimah
oarimah@uwo.ca
CS 3340B Assignment 3
250 981 235

5. The time complexity of this algorithm is O(n^2) since we are checking every word and then checking each character within those words.

   The algorithm

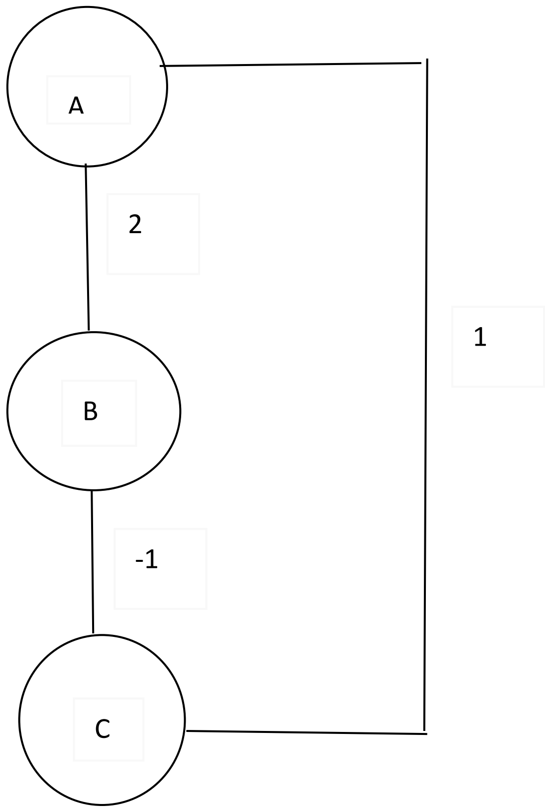
   1. create array minCost[i] and store the minimal cost of the line in
   2. create array wordsArray[i] which represent the first word
   3. create array extraSpace[i] which represents represents the index of the last word in the line where wordsArray[i] is the first word.
   4. The last word in the line is stored in wordsArr[j]. wordsarray[] size can range from i to wordsArr.length.
   5. We iterate over all the values of j and check the number of characters added so far in the current line.
   6. If the number of characters is not within the character limit M, then we set the currentLength to 0 and iterate over the next word to see if we can fit it into the next line.
   7. If it is within the limit, then we try to find the cost of the current line with the current number of characters and then compare this cost with the minimum found so far in minCose[i].
   8. Then we update minCost[i] and endIndex[i].
   9. This is repeated for all the words in the wordsArray.

      This solution was helped solves with https://www.geeksforgeeks.org/word-wrap-problem-dp-19/

6. Kruskal's algorithm for finding a minimum spanning tree can be modified to look for a maximum spanning tree while also using a Union-find algorithm to check for cycles in the graph. Commonly, one would use a minimum heap to sort the edges; however, in this case for finding the maximum spanning tree, one would use a maximum heap to sort the edges into the correct order. In this case, the maximum edge (the edge with the greatest weight) is taken out every time of the graph goes into the max heap. The correctness and time complexity should be unchanged.

Ositadinma Arimah
oarimah@uwo.ca
CS 3340B Assignment 3
250 981 235

7. Dijkstra's algorithm doesn't work with negative edge weights. This is because it is a greedy algorithm.
   Theorem: Dijkstra relies on the fact: if all weights are non-negative, adding an edge can never make a path shorter
   Proof:



Consider the above graph above:

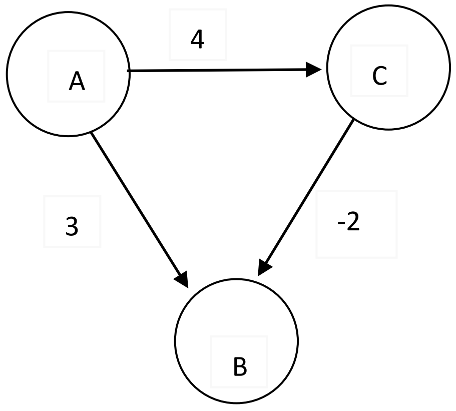
The graph consists of vertices A B and C.
The graph consists of edges AB, BC, and AC with weights 2, -1, and 1 respectively.
Dijkstra is a greedy algorithm it would take path AC as 1<2.
However, shortest path is AB to BC with the sum of the weights along that path being 1.

In conclusion, we have not found the shortest path in a Dijkstra's algorithm implementation involving negative edge weights.

Therefore, the theorem is then proven.

Ositadinma Arimah
oarimah@uwo.ca
CS 3340B Assignment 3
250 981 235

8. The all-pair-shortest-path algorithm is still correct when there are negative paths. This is because unlike Dijkstra's algorithm, the all-pair-shortest-path algorithm is not greedy. It considers all possible paths between two given vertices, and it takes the globally optimal choice, rather than the locally optimal choice. Consequently, the algorithm avoids the same downside present in Dijkstra's algorithm.



For example, referring to the graph above, when considering the shortest path from A -> B, even though the weight of the direct path A ->C (4) of path A -> C -> B is greater in weight than that of another shorter direct path A->B, the algorithm does not eliminate path A->C->B from being considered as the shortest path from A to B.
Therefore, by considering all possible paths, the all-pair-shortest-path algorithm is still correct when there are negative paths.

9.

My algorithm is a version of Prim's greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. If there are |V| total verticies in the graph, the algorithm loops through all combinations of i and j where i and j are values from (1,2,3….|V|), excluding the combinations where i=j.

In each loop of the algorithm, for each combination of i and j, we record the value of E[i,j] and E[j,i], if these edges exists, and their respective weights. Once we've taken a look at all pairs of i and j, and we have confirmed there is an existed edge between these vertices, we will add their weight to a respective weightList. The weight list stores the edge as a key and the respective weight as the value. This ensures that the when printing the adjacent edges, the output will be correct.

When creating the algorithm for the Prims MST, this is correct as we use a minimum Heap to remove the smallest item and then perform the algorithm, which was followed from the slide's teachings. Therefore, when printing out the minimum spanning tree, this output will be correct. Because the algorithm examines all combinations of i and j, the algorithm must be correct.

The algorithm runs in of O((|V| + E) log V) time, this is because the Prims algorithm has time complexity of O((|V| + E) log V). My program requires finding and |E| based on a pair of verticies, adding 2|V| and their weight to the weightList.