CS 3340 Assignment 3

1. P = babbabbabbababbab b

Next[1] = 0

q=0

i= 2

next[2] = 0

i = 3

q=1

next[3]=1

i=4

q=next[1]=0

q=1

next[4]=1

i=5

q=2

next[5]=2

i=6

q=3

next[6] = 3

i=7

q=4

next[7] = 4

i=8

q=5

next[8] = 5

.

.

.

i=12

q= 9

next[12] = 9

i=13

q=next[9] = 6

q=next[6]= 3

q=next[3]= 1

q=2

next[13]=2

i=14

q=3

next[14] = 3

i=15

q=next[1]=0

q=4

next[15]=4

i=16

q=5

next[16]=5

i=17

q=6

next[17]=6

i=18

q=7

next[18]=7

Therefore

next =

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 3 | 4 | 5 | 6 | 7 |

1. Algorithm: String Matching(T,n, P, m)

Input: text T[1 . . . n] and pattern P[1 . . . m].

Output: matching positions.

begin

i := 1 ;

q := 0 ;

maxMatch =0; // max prefix matched

maxPos =0;// max prefix position

while i ≤ n do

if T[i] == P[q + 1] then

i := i + 1 ;

q := q + 1 ;

else

if q == 0 then

i := i + 1 ;

else

q := next[q];

if q == m then

maxMatch=m; // all the chars in the pattern match

maxPos=i-m; //mx prefix is the size of the entire pattern

q = next[q];

if q> maxMatch then

maxMatch=q; // max num chars matched

maxPos = i-q // location of max pattern

print “Largest pattern found at position” maxPos

end

The algorithm is correct because it follows the structure of the KMP algorithm so once i is the same value as n the while loop will exit. The algorithm also produces the correct answer because if the entire pattern is in the text, the algorithm works like KMP, and if a shorter part of the pattern matches the text, the variable storing the max size of the pattern matched will change and be outputted. The time complexity is the same as the original algorithm since the only thing added was the if statement which is of constand time. Therefor the time complexity is O(n).

Algorithm ModifiedLCS

Input: c,X,Y

Output: Longest Common Sequence between X and Y

begin

n = c[X.length, Y.length]

Array s with size n

i = X.length

j = Y.length

while i > 0 and j > 0 do

if xi == yj then

s[n] = xi

n = n – 1

i = i − 1

j = j – 1

else if c[i − 1, j] ≥ c[i, j − 1] then

i = i – 1

else

j = j − 1

for k = 1 to s.length do

Print s[k]

end

The algorithm is correct because during each iteration of the while loop, i or j decrement, so there will be a point when one of the variables is 0 and the while loop will exit. The answer produced is correct since the values of b are calculated as the program goes on, so there is no need to store the values. All other instructions work like the original function. The time complexity of the algorithm is O(m + n) because the while loop executes m times and the for loop executes min(m,n) times.

1. The first stop will be the furthest station that is less than or equal to m miles. We continue with the idea throughout the journey (i.e skate n miles where n<=m). If S is the solution which contains stops 1 – k. Let n be the furthest stopping position before the professor runs out of water, now replace the first stop as n ( the new starting position). If stop 1 – stop 2 is m stop 2 -n will be less than or equal to m making it possible to get to the stop without running out of water.
2. Form a paragraph of width M by putting line breaks within the list so the lines can be printed neatly. The empty spaces are calculated by squaring the number of empty spaces in a line. For every range of words between Wi and Wj, we need to figure out the cost of it being on one line. If Wi to Wj can’t fit in a line the cost will be C{I,j)= ∞. The minimum cost will therefore be the cost of the line with the words plus the cost of the spaces. Since the algorithm will look through I and Wj until best cost is calculated, the time complexity of the algorithm will be O(n2).

1. MST Prim(G, w, r)

for each u ∈ V [G] do

key[u] := ∞;

π[u] := NIL;

key[r] := 0;

Q := V [G];

while Q 6= ∅ do

u := Extract Max(Q);

for each v ∈ Adj[u] do

if v ∈ Q and w(u, v) > key[v] then

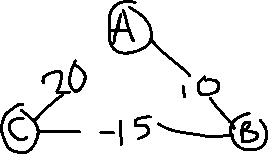
π[v] := u;

key[v] := w(u, v);

update key[v] in Q

This algorithm is exactly like Prims algorithm except instead of using a min heap you are using a max heap. The algorithm will produce the correct answer and terminate since none of the loops were changed. The time complexity is : O((|V | + |E|) log(|V |))

1. Dijkstra relies on the fact that adding an edge does not make the path shorter. Therefore when we have a graph like the following:



The algorithm recognizes that A-B < A-C therefore it will never check the value of ACB, meaning the algorithm will choose B no matter what the value of CB is.

1. All-pair -shortest-path algorithm works will produce the correct output. The algorithm computes every possible paths between pairs of vertices, then then selects the lowest weighted path. Since the algorithm checks all the paths the negative weighted paths will also be included in the calculation.