Part A:

Input: G=(V,E) undirected graph (x,y)E

Output: Doubly linked list which contains Pu,x + (x,y) + Py,v = Pmax such that Pmax is a maximal path including (x,y)

Reset\_graph(G) {//is in the ctor

Foreach v  V(G)

Color[v] ← WHITE

π[v] ← NILL

End foreach

}

Print-Path(v, vertices\_in\_path){

While(v != NILL)

vertices\_in\_path ← v

v ← π[v]

π[v] ← NILL //avoid incorrect path after transpose of graph

}

Find\_max\_path(v, pointer\_last\_in\_path) {

Color[v] ← RED

Foreach u  N+(V)

If(color[u] = WHITE)

π[u] ← v

Find\_max\_path(u, pointer\_last\_in\_path)

end foreach

if(pointer\_last\_in\_path = NILL)

pointer\_last\_in\_path ← v

}

Find\_maximal\_path(G, (x,y)) {

List vertices\_in\_path\_a ← ∅

List vertices\_in\_path\_a\_b ← ∅

Reset\_graph(G)

pointer\_last\_in\_path\_a = y

Find\_max\_path(y, pointer\_last\_in\_path\_a)

Print-Path(pointer\_last\_in\_path\_a, vertices\_in\_path)

Reverse(vertices\_in\_path)

G\_TRANSPOSE = (V,E\_TRANSPOSE)

pointer\_last\_in\_path\_b = x

Find\_max\_path(x  G\_TRANSPOSE, List vertices\_in\_path\_a\_b)

Print-Path(pointer\_last\_in\_path\_b, vertices\_in\_path\_a\_b)

If(last[vertices\_in\_path\_a\_b] = last[vertices\_in\_path]) //possible cycle detected

Remove(last[vertices\_in\_path]))

APPEND(pointer\_last\_in\_path\_a, vertices\_in\_path\_a\_b)

Return vertices\_in\_path\_a\_b

}

Run time Complexity:

Sub-Routines:

------

Reset\_graph – O(V)

max\_path – O(V + E)

Print-Path – O(V)

Reverse – O(V)

Transpose Algorithm on adjacency list – O(V+E)

APPEND on doubly linked lists – O(1)

RemoveLast on doubly linked lists with pointer to last – O(1)

-----

Primary Routine:

Find\_maximal\_path:

2\* Reset\_graph + 2\*Print-Path + 2\* Find\_max\_path + G\_TRANSPOSE + Reverse + APPEND + RemoveLast ≡ 4O(V) + 3O(V+E) ≡O(V+E)

Space complexity:

O(V) : vertices\_in\_path

+

O(V+E) : G

= O(V+E)

Please note, that the algorithm has been implemented in object oriented programing for reusability and ease.

We know that the maximal path which includes (x,y)E on directed graph is:

Pmaximal = Pu,x + (x,y)+ Py,v where v,uV are leaves vertices of the spanning trees which can be derived by starting depth first search only for the strongly connected component of vertex y  G

And depth first search only for the strong connected component of vertex x  G\_TRANSPOSE

Here I modified DFS such that it will only scan the strongly connected component of the starting vertex, also, I made sure that a pointer to the first vertex of back-edge(or leaf) is returned properly, additionally I degraded DFS by removing begin and finish times, because I only care about the first leaf in the spanning tree encounter.

Now, because DFS will give us path from u → x then the result has to be reversed this is done on a doubly linked list with complexity of O(V).

Another modification was to avoid cycles which can appear if (x,y)  E is a bridge in G~ And x  Cn, y  Cn2

This is done by checking the last elements of Pu,x and Py,v, if they are equal a cycle exists and we need to remove the last element from Py,v in order to get correct path.