CS3510-A: Design and Analysis of Algorithms, Spring 2021

Homework-2

January 26, 2021

DUE DATE: Tuesday, February 2, 1l:59pm

Note-1: Your homework solutions should be electronically formatted as a single PDF document that you will upload on Gradescope. If you have to include some handwritten parts, please make sure that they are very clearly written and that you include them as high resolution images.
Note-2: Please think twice before you copy a solution from another student or resource (book, web site, etc). It is not worth the risk and embarrassment.
Note-3: You need to **explain/justify** your answers. Do not expect full credit if you just state the correct answer.
Note-4: You will get 2 extra points if you submit electronically typed solutions instead of hand-written.

Problem-1 (30 points)

Undirected Graph → (Symmetrical matrix)

You are planning the seating arrangement for a wedding. You are given the list of guests, V, and a lookup-table T where $T[v]$ for $v \in V$ is a list of guests that v knows. If u knows v, then v also knows u. You are required to arrange the seating such that any guest at a dining table knows every other guest sitting at the same dining table either directly or through some other guests sitting at the same dining table. For example, if x knows y and y knows z and z knows w, then x, y, z, w can sit at the same dining table. Describe an efficient algorithm that takes as input the set V and lookup-table T and returns the minimum number of dining tables needed to achieve a valid seating. You can assume a check in the lookup-table is an $O(1)$ operation.

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode. Also, explain the time complexity and correctness of the prescribed algorithm.

Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

Qs: Do a search & prune last x to a table?

Not guaranteed to be a strongly connected graph?

Psuedo:

1) Start w/ empty explored set
2) Pick a random unexplored node & do BFS until reach stopping point (adding to explored set)
3) Use output from #2 as a "table"
4) Repeat #2 - #3 until all nodes explored
5) Return total number of "tables"

Eff: Should be same as BFS/DFS ?

Check notes to verify

Problem-2 (35 points)

A number of art museums around the country have been featuring work by an artist named Mark Lombardi (1951–2000), consisting of a set of intricately rendered graphs. Building on a great deal of research, these graphs encode the relationships among people involved in major political scandals over the past several decades: the nodes correspond to participants, and each edge indicates some type of relationship between a pair of participants. And so, if you peer closely enough at the drawings, you can trace out ominous looking paths from a high-ranking U.S. government official, to a former business partner, to a bank in Switzerland, to a shadowy arms dealer. Such pictures form striking examples of social networks, which have nodes representing people and organizations, and edges representing relationships of various kinds. And the short paths that abound in these networks have attracted considerable attention recently, as people ponder what they mean. In the case of Mark Lombardi's graphs, they hint at the short set of steps that can carry you from the reputable to the disreputable.

Of course, a single, spurious short path between nodes v and w in such a network may be more coincidental than anything else; a large number of short paths between v and w can be much more convincing. So in addition to the problem of computing a single shortest $v - w$ path in a graph G, social networks researchers have looked at the problem of determining the number of shortest $v - w$ paths.

You are given an undirected graph, $G = (V, E)$ and two nodes in G, v, w. Design an algorithm that computes the number of shortest paths between these two nodes in G. You can assume that all edge weights are positive. You do not need to identify these paths, but simply find the number of paths. Justify your correctness and running time.

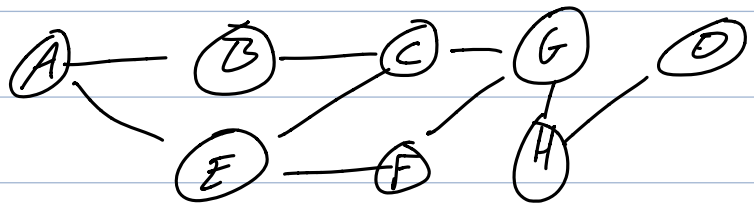
Note 1: Please provide a detailed description of your solution in plain English or in pseudocode.

Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

1) Do BFS from v to w to find shortest path len
2) Keep doing BFS for all paths <= shortest ?
   try Depth-Limited Search for all possible paths <= Shortest len ?

Eff:
$T(n) = BFS(n, m) + BFS(n, m)$?

1) Do BFS traversal over whole graph from (v)
   keep track of all node distances from (v).
2) From (w) checks all adjacent nodes



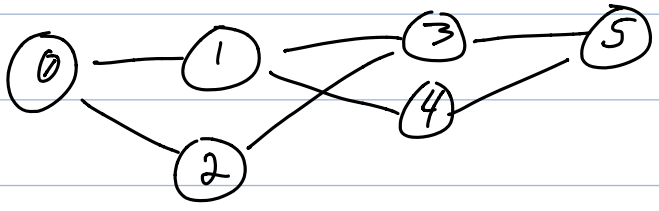
ABCD

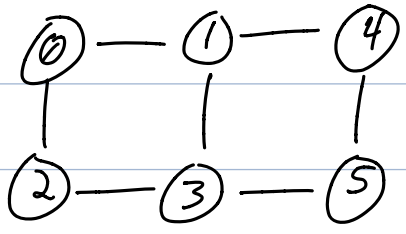
AECD

AEFD

$n = 8$

$n/2 = 4$





Problem-3 (35 points)

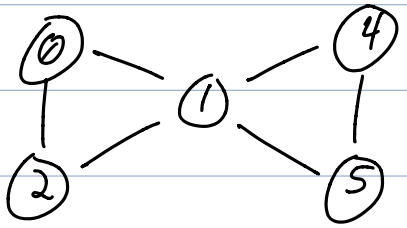
There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops – have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

Suppose there exists a communication network $N = (V, E)$ with n nodes (or "hubs"). N contains two hubs s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some hub v ($v \neq s, t$) such that deleting v from the network destroys all $s-t$ circuits (paths). In other words, the communication network obtained by deleting v contains no path from s to t. Give a linear time algorithm to find such a hub v. Justify the correctness and running time of your algorithm.

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode.

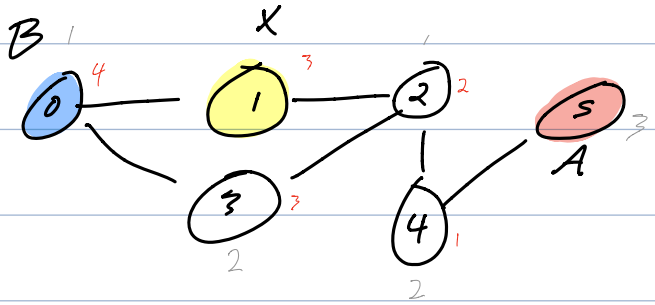
Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

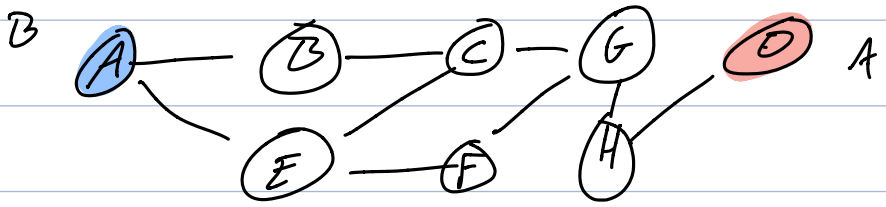
like finding root node of tree?



1) Pick random node X
2) Set A node furthest from X
3) Set B node furthest from A

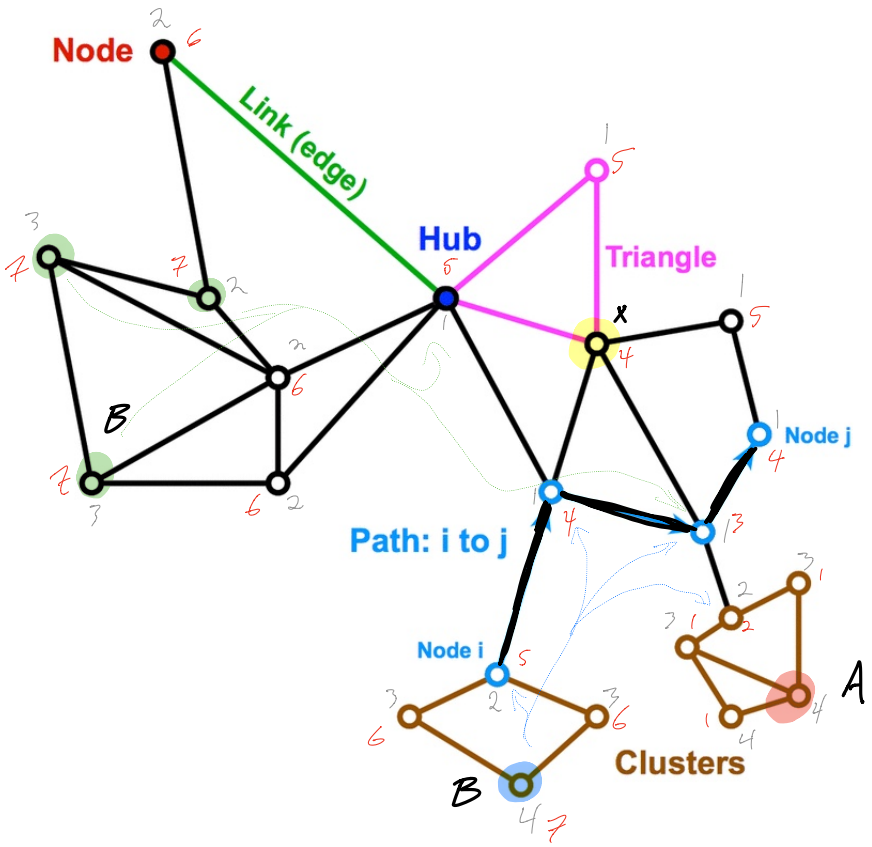
Something with path A-B ?

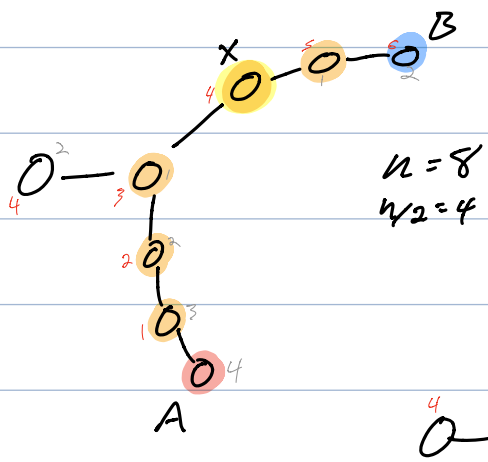




A B C G H D
A E C G H D
A E F G H D

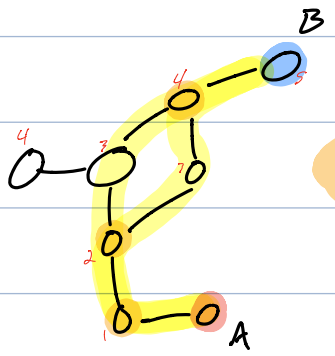
G, H appear in all paths ∴ hubs



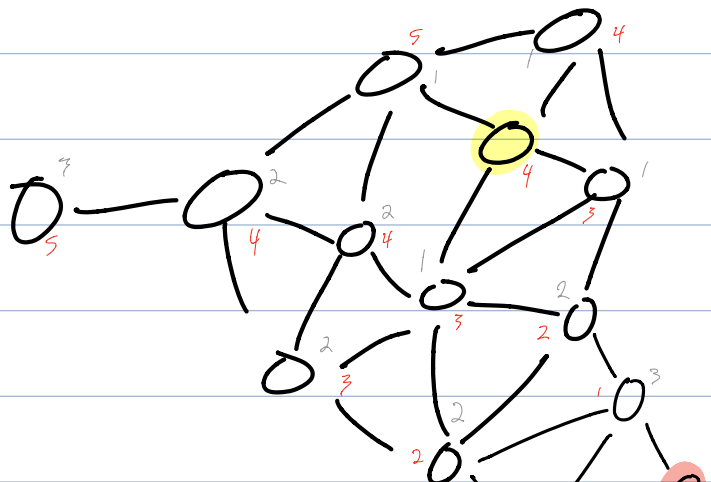


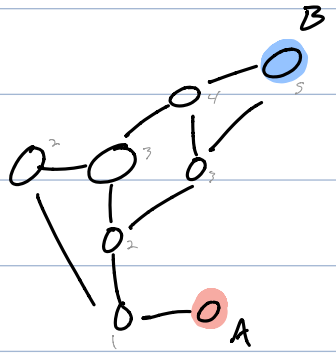
$n = 8$

$n_2 = 4$



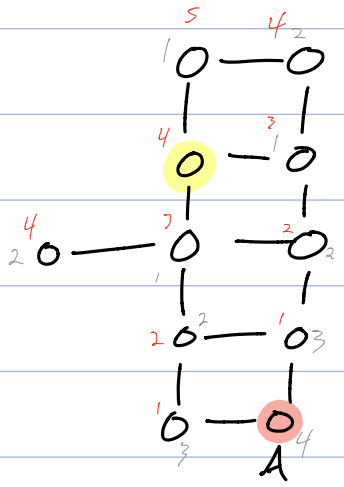
ALC Hubs





$n = 8$

$n/2 = 4$



$n = 11$

$n/2 = 5$