Xi Liu, xl3504

“I understand the ground rules and agree to abide by them. I will not share ideas, solutions, or assist another student during this exam, nor will I seek assistance from another student or attempt to view their ideas or solutions.”

0

Graham’s algorithm

1

(a)

Jarvis’s algorithm has time O(nh), h is number of points on boundary of convex hull, input of n points

Graham’s scan has time O(n lg n) due to sorting

running time of Jarvis’s algorithm become best when h is a lot less than n

when h = 3, and n is a lot bigger than 3, then Jarvis’s algorithm has time O(3n) whereas Graham’s algorithm has time O(n lg n)

(b)

yes, T is still going to be the output. since prim’s algorithm involves an if test that is if(u is in Q & u.key > weight(u, v)){ u.key = weight(u, v); u.parent = v}, all of the same choices will be made when the weights of all the edges are decreased by 1 (for example: decrease every element of a sorted list will produce a list that is still sorted)

2

call max\_paper() function

int cmp\_dec(const void \* p1, const void \* p2)

{/\* comparison function to sort in decreasing order \*/

return \*(int \*)p2 - \*(int \*)p1;

}

int max\_paper(int \* P, int n, int \* E, int m)

{

qsort(P, n, sizeof(int), cmp\_dec); /\* sort P and E in decreasing order \*/

qsort(E, m, sizeof(int), cmp\_dec);

int i = 0, j = 0, count = 0;

while(i < n)

{

if(P[i] <= E[j]) /\* when length of paper less than length of envelope \*/

{

++count; ++i; ++j; /\* ++count means put a paper inside \*/

}

else

++i;

}

return count;

}

3

(a)

vector<vertex> sorted\_v = topological\_sort(G)

vector<float> d /\* weight \*/

vector<list<edge>> adj /\* adjacency list \*/

d[s] = 0

for vertex i in sorted\_v

for j in adj[i]

d[i] = max(d[j], d[i] + weight(i, j))

/\* now d has the longest paths from vertex s to all other vertices

in G \*/

for(int i = 0; i < d.size(); ++i)

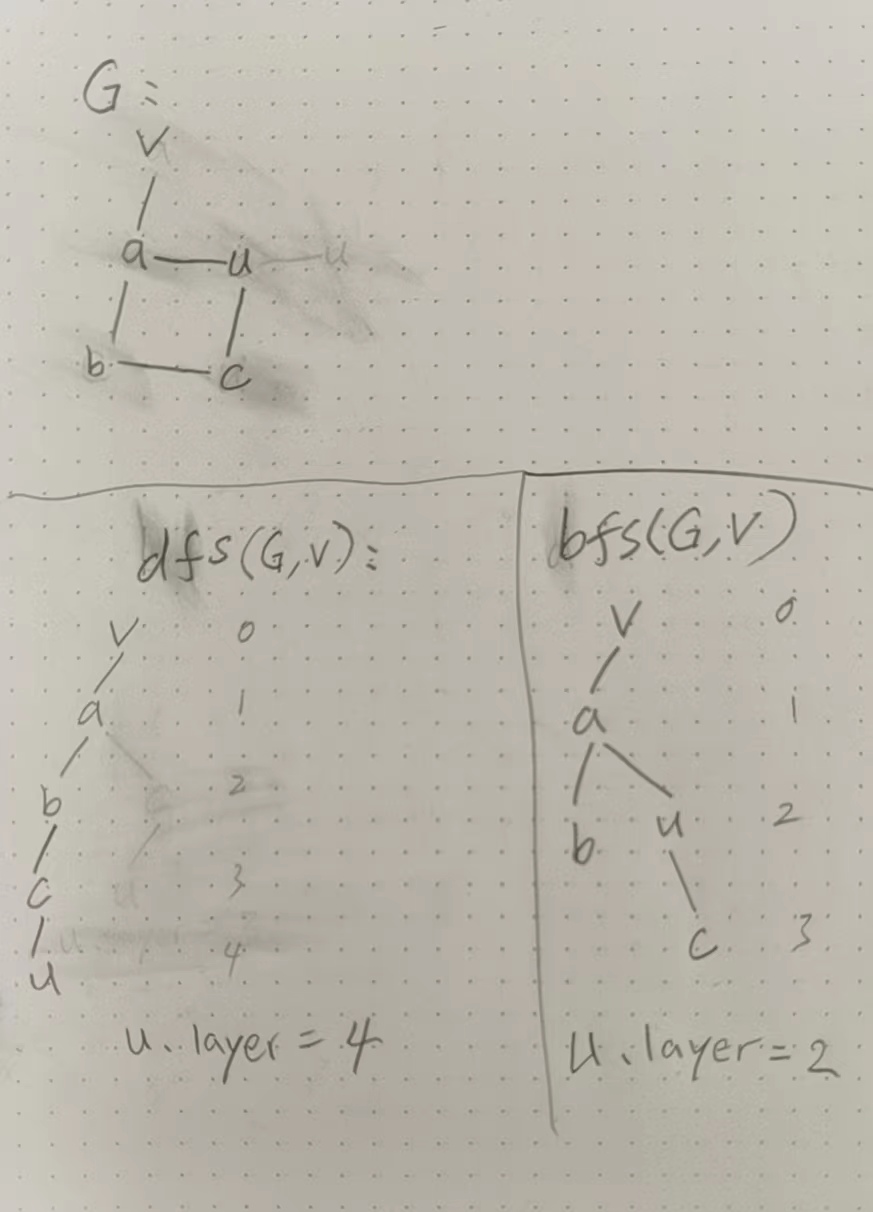
if(d[i] >= k) /\* at least k vertices \*/

return true

return false

(b)

bob’s answer is correct if there is a cycle in the graph, then the layer number of u in the DFS tree is greater than the layer number of u in the BFS tree since BFS finds the shortest path. see the example photo next page



4

for all good edges, label good edge weight be 0, since there is no need to minimize or maximize amount of good edge; for all bad edges, label bad edge weight be 1, take O(|E|) time

call Dijkstra’s algorithm implemented by fibonacci’s heap on the graph G, take O(|E| + |V| log |V|) time

iterate over the shortest path found from Dijkstra’s algorithm, count the length of the path that has the least number of bad edges O(|V|) time

total time = O(|E|) + O(|E| + |V| log |V|) + O(|V|) = O(|E| + |V| log |V|)

5

vertex has 3 different types of color: white, odd, even

white is vertex not visited

if the vertex has a color of odd means the vertex can be reached by odd number of steps

if the vertex has a color of even means the vertex can be reached by even steps

if the vertex has a color of both means the vertex can be reached by both even number of steps and odd number of steps

step 1: for each vertex u in G.V

u.color = white

call modified\_dfs\_visit(G, s)

modified\_dfs\_visit(G, s)

s.color = even

for vertex v in G.adj[s]

if v.color != both

if u.color == white

u.color = s.color + 1

/\* this means if s.color is even,

then u.color is odd; if s.color is odd, then u is even \*/

modified\_dfs\_visit(G, u)

if u.color == s.color

s.color = both

modified\_dfs\_visit(G, u)

step 2: if the color of vertex t is “even” or “both”, then

this means a walk from s to t using even number of good edges is found, so return true;

else, return false