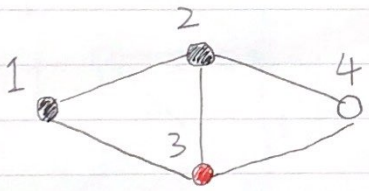
No

DATE

Solution

1. $2^{\sqrt{2\log n}} < \sqrt{n} < n < n\log n < n(\log n)^3 < 2^{2n} < 2^{n^2}$

2. We can define that node has 3 different color: white red, and black
Before node is visited, it's white. when we put the node into queue, it becomes red, and when it out of the queue, it becomes black
when root is out of queue, it becomes black. All of its adjacent node will get into the queue and becomes red, because BFS is layered from left to right.
If there is no cycle, the color of child nodes should be white which is not visited. If child node is red, it means this child node is in the queue, it means this child node has been visited, so there is a cycle.
In order to return the cycle, we can put every node which out of the queue into a list. Once we find a child node is red, get this child node out of queue and put it into the list, then return the list. (output a cycle)



Running Time: Each edge is visited at most twice. Because we go through every vertex of the graph, the overall complexity is $O(m+n)$ with m and n being the number of edges and vertices respectively

No.

DATE / /

3. Since node degree of binary tree is less or equal to 2,

sum of node = n_0 (node degree = 0) + n_1 (node degree = 1) + n_2 (node degree = 2)

$\Rightarrow n = n_0 + n_1 + n_2$ ..... ①

n_1 (node degree = 1) has 1 child, n_2 has 2 children

then the total number of children in a binary tree = $n_1 + 2n_2$

sum of node = root + number of children

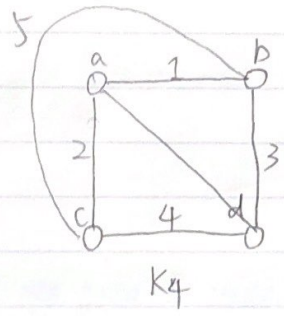
$n = 1 + n_1 + 2n_2$ ..... ②

Together ① and ② $\Rightarrow$ $n_0 = n_2 + 1$

Finally, in any nonempty binary tree the number of nodes with two children is exactly one less then the number of leaves

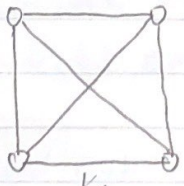
4. We can assume that K_5 is a planar graph.

According to the definition of planar graph: Planar graphs are the graphs which can be drawn without intersection

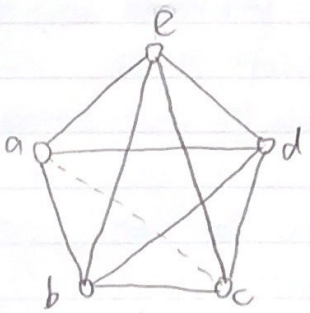


K_4

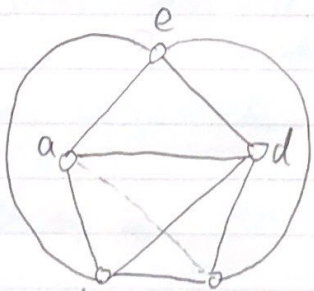
As we can see from K_4 graph, K_4 is a planar graph.



K_4



K_5



K_5

For graph K_5, there is no edge between a and c.

Hence, K_5 is not a planar graph

No.

DATE / /

5.

Operations	Cost
1	1
2	2
3	1
4	4
5	1
6	1
7	1
8	8

Aggregate Method:

# of operations $= 2^{(\log_2 n)} + 1$

# of cost:

$$\sum_{m=0}^{\log_2 n} 2^m = 1 + 2 + 4 + \cdots + 2^{\log_2 n}$$

$$= 2^{(\log_2 n)+1} - 1$$

Total work:

$$\left[2^{(\log_2 n)} + 1 \right] + \left[2^{(\log_2 n)+1} - 1 \right]$$

$$= 3 \times 2^{\log_2 n}$$

$$AC(\text{per operation}) = \frac{3 \times 2^{\log_2 n}}{2^{\log_2 n} + 1}$$

$$\lim_{n \to \infty} \frac{3 \times 2^{\log_2 n}}{2^{\log_2 n} + 1} = 3$$

Hence, $AC = O(3)$ is a constant.

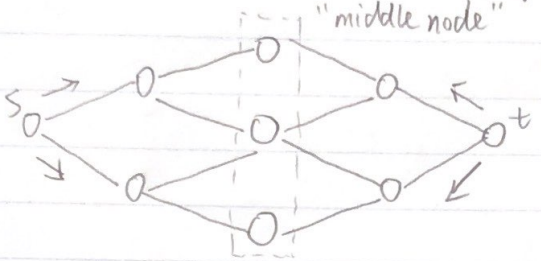
$O(1)$ amortized cost per operation.

6. If the size of table is filled with elements, the size of it double
The time complexity is $O(n)$, when size increased by 2, we have to double size of space., assume the size of new table is n.
Then we copy all elements to new table, so insert all elements to new table costs $O(n/2)$
Hence, the total complexity is $O(n \cdot n/2) = O(n^2/2) = O(n^2)$
Fred wants to increase size with every insert by just two over the pervious size, so we have to calculate amortized cost per insertion.
amortized cost per insertion $= \frac{O(n^2)}{n} = O(n)$

No.
DATE / /

7. Bidirectional BFS. + DFS

①We can use bidirectional BFS to find the minimum edge



Bidirectional BFS means that we can ~~search~~ start our searching from s and t at the same time using BFS. When two BFS get the same node, then we find the minimum path from s to t. (one thing I want to mention, we may find not only one minimum path, in the other word, there may be many paths equal to the minimum path)

② If we find only 1 path which equals to the minimum path, then calculate weight (total weight) from s to this "middle node" and "middle node" to t, then plus these two weights together to return the weight of the only one minimum weight.

If we find more the 1 path which equal to the minimum path. then we should use DFS to calculate weights of all paths, then compare them to find the minimum one. In detail, we should use DFS to go through all paths from "middle nodes" to s and from "middle nodes" to t seperately. Firstly, find minimum paths which starts from each node to s or t, Secondly, compare ~~the~~ minimum paths of each "middle node" to find only one path which go through only one "middle node", then retun the minimum path.

Time complexity:

① Time complexity of bidirectional BFS is $O(n+e)$

② Time complexity of DFS is also $O(n+e)$

The time required to find the adjacency points of all vertices is $O(e)$, and the time taken to access the adjacency points of vertices is $O(n)$.

In this case, the total time complexity is $O(n+e)$