1.

Each house is different size in front, we need to place fire hydrant within 300 feet for any houses.

For instance consider in a sub-street, the easiest way is to place fire hydrant 300 feet distance from the first house, Then the last house is away 300 feet from fire hydrant. The sub-street is from the house that next to the last house in previous sub-street. Loop until the street is fulfilled enough fire hydrant.

2.

Assume n topics t1,… tn take time a1… an to present(ai < 50) , m is the total number of lectures in the semester.

So, we have: a1 + a2 + … + an < 50m

with each lecture, teacher will present 1 or more topics as possible as you want. That means:

For instance i-j topics for lecture z (0 < z < m): ai + … + aj < 50 and ai + … + aj+1 > 50. Remain topic j+1 → n need to be presented and m-z lectures.

Now, teacher will present topics ak + … + aj (i <= k <= j) in later lectures.

So, remain topic k→n need to be presented and m-z lectures.

Absolutely, teacher has more topics need to be presented and the remain lecture number is the same. So teacher cannot use this strategy to finish class soon.

3.

Assume within any i, cost ci >= 1. So with any i, ci <= c2i , that means the order of the edge cost(array after sorted across edged cost) remains unchanged.

Since P is a minimum-cost from s-t, so cost P = cs + … + ct

Now, we replace new instance but the graph is not changed except the cost of edge.

So the path is not changed => new cost P = cs2 + ... + ct2

Because cs + … + ct is the minimum number among any sum between edges for path s→t and the order of the edge cost unchanged , the graph is also not changed, so it is obvious that cs2 + ... + ct2 is the minimum number among any square sum => P is minimum-cost s-t path.

Otherwise within any i, cost 0 <= ci < 1. So with any i, ci => c2i that means the order of the edge cost(array after sorted across edged cost) is changed in reverse order. So, P is not minimum-cost s-t path.

4.

To prove this problem, I will prove within any x∈S, the length from root to x is exactly k.

Assume fx1, fx2 are 2 smallest fx numbers

fx1 + fx2 > max(x∈S, fx) Because of max(x∈S, fx) < 2min(x∈S, fx)

So the next sum of 2 smallest fx number is obvious not “fx1+fx2”, this repeats until all number in ‘old’ fx are summed (until the max number is summed), then the next round is the same.

After all (until remain only 1 number), we have an balanced tree.

Since |S| = 2k , that means the leaf number is 2k, so the height from root to any leaves in S of balanced tree is k, so the length of a codeword in S is exactly k.

5.

Consider in case the black to win, there needs to be a black stones path from upper left to lower right.

To make it easier, we can assume lower right is ground(head), and upper left is the highest from ground(tail). So the height is from the ground but the rank is only the height if and only if that black stone is connect to ground. The black wins only if there exist a black stone that is at highest rank.

The rank number of black stone in ground is 1. If a black stone has no rank, its rank number is 0.

I will define a cluster is to include all black stones connected, all black stones in a cluster have the same highest height and rank, it is also the height and rank of cluster.

We can use Union-Find like this:

Every time black stone player moves, traverse all neighbor black stones (or clusters of black stones),

Now we combine(union) all black stones (or clusters) related to the traversing black stone into a cluster with the highest height and rank, if the cluster has a rank then the rank is the height.

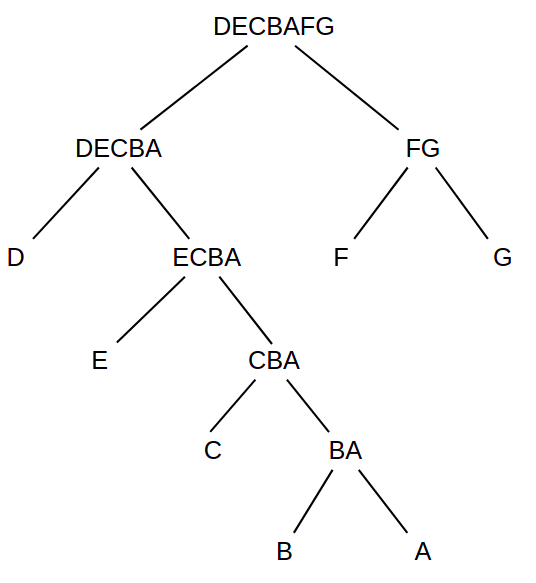
The black stone player will win when there is a cluster reached the highest rank (tail).

Similar to the white stone player.

The running time is O(1) for each move, O(n) to change height&rank for all stones in cluster with n is the black/white stones number in cluster.

6.

a.



b. Consider the Huffman tree, the frequencies has to be satisfied:

CBA <= D <= ECBA and G <= DECBA <= FG

===> 6 <= D <= 11 and 15 <= D + 11 <= 30

number of D occurrences can be: 6, 7, 8, 9, 10, 11

7.

Suppose graph H = HBFS = HDFS∈ G and there is an edge e∈ E(G) but ∉ E(H)

So this edge e is not visited by both DFS or BFS algorithm. The reason is with DFS that edge connect from parent to the child but in another side from the parent node, with BFS that edge is not connected to the parent node.

That cannot happen because if that edge from parent to the child node, there is no way BFS cannot visit it before.