CS 161 (Stanford, Winter 2023) Section 7

1 Encoding

Suppose we encode lowercase letters into a numeric string as follows: we encode a as 1, b as 2, etc., and z as 26. Given a numeric string S of length n, develop an $O(n)$ algorithm to find how many letter strings this can correspond to. For example, for the numeric string 123, the algorithm should output 3 because the letter strings that map to this numeric string are abc, lc, and aw.

2 Knight Moves

Given an 8×8 chessboard and a knight that starts at position $a1$ (the letter denotes the column and the number the row), devise an algorithm that returns how many ways the knight can end up at position xy after k moves. Knights move in an "L" shape: ± 1 square in one direction and ± 2 squares in the other direction.

3 Greedy or Not?

Sometimes it can be tricky to tell when a greedy algorithm applies. For each problem, say whether or not the greedy solution would work for the problem. If it wouldn't work, give a counter-example.

1. You have unlimited objects of different sizes, and you want to completely fill a box with as few objects as possible. (Greedy: Keep putting the largest object possible in for the space you have left)
2. You have unlimited objects, all of which are size 3^k for some integer k, and you want to completely fill a box with as few objects as possible. (Greedy: same approach as the previous problem)
3. You have lines that can fit a fixed number of characters. You want to print out a series of words while using as few lines as possible. (Greedy: Fit as many words as you can on a given line)
4. You want to get from hotel 1 to hotel n, and you can travel at most k distance between hotels before collapsing from exhaustion. Find the minimum cost of hotels. (Greedy: Go as far as you can before stopping at a hotel)

4 Mice to Holes

There are n mice and n holes along a line. Each hole can accommodate only 1 mouse. A mouse can stay at his position, move one step right from x to $x+1$, or move one step left from x to $x-1$. Any of these moves consumes 1 minute. Mice can move simultaneously. Assign mice to holes such that the time it takes for the last mouse to get to a hole is minimized, and return the amount of time it takes for that last mouse to get to its hole.

Example:

Mice positions: $[4, -4, 2]$

Hole positions: $[4, 0, 5]$

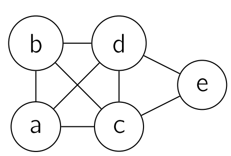
Best case: The last mouse gets to its hole in 4 minutes. $\{4 \to 4, -4 \to 0, 2 \to 5\}$ and $\{4 \to 5, -4 \to 0, 2 \to 4\}$ are both possible solutions.

5 Well Connected Graphs

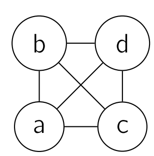
Let $G = (V, E)$ be an undirected, unweighted graph. For a subset $S \subseteq V$, define the **subgraph induced by** S to be the graph $G' = (S, E')$, where $E' \subseteq E$, and an edge $\{u, v\} \in E$ is included in E' if and only if $u \in S$ and $v \in S$.

For any $k < n$, say that a graph G is k-well-connected if every vertex has degree at least k. (That is, if there are least k edges coming out of each vertex).

For example, in the graph G below, the subgraph G' induced by $S = \{a, b, c, d\}$ is shown on the right. G' is 3-well-connected, since every vertex in G' has degree at least 3. However, G is not 3-well-connected since vertex E has degree 2.



$G = (V, E)$



$G' = (S, E')$, for $S = \{a, b, c, d\}$

Observation: If G' is a k-well-connected subgraph induced by S, and $v \in V$ has degree $< k$, then $v \notin S$. This is because v would have degree $< k$ in the induced subgraph G' as well, and so G' couldn't be k-well-connected if it included v.

Guided by the **observation** above, design a greedy algorithm to find a maximal set $S \subseteq V$ so that the subgraph $G' = (S, E')$ induced by S is k-well-connected. You do not need to formally prove why your algorithm is correct, but give an informal but convincing justification.

In the example above, if $k = 3$, your algorithm should return $\{a, b, c, d\}$, and if $k = 4$ your algorithm should return the empty set.

You may assume that your representation of a graph supports the following operations:

- `degree(v)`: return the degree of a vertex in time $O(1)$
- `remove(v)`: remove a vertex and all edges connected to that vertex from the graph, in time $O(\text{degree}(v))$.

Your algorithm should run in time $O(n^2)$.