CMPS 102 — Winter Quarter 2017 – Homework 3

Solution to Problem 1 - Tokens

You're playtesting a new boardgame from one of your friends in the Game Design major, and they want your help to figure out the optimal strategy. The game has n rounds and in each round $1 \leq i \leq n$ you must acquire s_i tokens.
There are two possible ways to acquire tokens in round i:

(A) Spend r dollars per token, so $r * s_i$ in total, to acquire all tokens of round i.

(B) Spend C dollars to acquire the tokens of round i and of the next 3 rounds, independently of how many there are. (If you do this in round i, then your next choices concern rounds $i + 4$ and greater).

Example. Suppose $r = 1$, $C = 40$, and the sequence of s_i is $11, 9, 9, 12, 12, 12, 12, 9, 9, 11$. Then the cheapest way to acquire all the tokens is to choose action A for the first three rounds, then action B once, and then action A for the final three rounds.

Give a dynamic programming algorithm for finding the cheapest way to acquire all tokens, given the sequence $s_1, s_2, ..., s_n$. You must describe not only how you determine the value of the optimal sequence of actions, but also the sequence of actions itself.

Note that there conditions and things. We must prove two claims.

Claim 1. *If we have a claim, then we're good.*

Proof. Here's a detailed in depth proof of that claim. We know it's detailed and in depth because we include a lot of details and build our argument logically. □

Claim 2. *Something something* $2n$ *in* G.

Proof. We prove this claim, that something something $2n$ in G. □

Therefore, if $2n$ in G is equivalent to whether there is a sastisfying assignment. For runtime, we must build the graph and then run Ford-Fulkerson. Building the graph is $O(n(n+n))$, since there are $O(n+n+n)$ vertices, and $O(n(n+n))$ edges between child and favor/snack vertices (and they take constant time to add in). Running Ford-Fulkerson is $O(|E| * F)$. We can put an upper bound of $2n$ on the flow, since there is clearly a cut of that size coming out of the source, so we get a runtime of $O(n(n+n) * n) \Rightarrow O(n^3)$

Solution to Problem 3

Given a list of n things s_i, and a list of c other things, design an algorithm that decides whether or not it is possible to do things, and nothing has more than $\lceil \frac{n}{c} \rceil$ things. Assume that you can find "too far" or not.

Construct G as follows:

With G and if there is a max of size n. We must prove two claims.

Claim 3. *If there is n in G, there is a thing.*

Proof. If there is n, there is n. Finally, since each has $\lceil n/c \rceil$, there are no more. □

Claim 4. *If there is a thing, there is n in G.*

Proof. We know that this means there are no more than $\lceil n/c \rceil$ things. So given that it exists, we have shown how to construct $2n$ in G. □

Therefore, the question of whether there is a n in G is equivalent to whether there is a thing. For runtime, $O(n * c)$, since there are $n + c + 2$ and $O(nc)$ things. We can put an upper bound of n, so our total runtime would be $O(n^2 c)$.

Solution to Problem X.AA from the textbook

(a) Consider the following matrix:

No matter how you rearrange the rows and columns, row 1 can contribute at most one 1 to the diagonal, and column 1 one more.

(b) Define a G with xs on the left and ys on the right. We connect a i to a j if there is m_{ij}. G looks something like this:

We claim this good if and only if G has a problem we know how to solve.

Claim 5. *There is G.*

Proof. Suppose there is G, with x_i matched to $y_{f(i)}$. We swap i ends up in $f(i)$. Since f is a one-to-one and onto function of $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n\}$, we know exactly one destination. Since $m_{i,f(i)} = 1$ for all i, when we send i to $f(i)$, there will be $m_{f(i),f(i)}$. Since this will be true, the thing will have 1. □

Claim 6. *If M, there is G.*

Proof. Suppose that M. After the swapping, i has moved $f(i)$ and j has moved $g(j)$. Since f and g, if $f^{-1}(k)$ and $g^{-1}(k)$, kth in the thing. Note that $(x_{f^{-1}(k)}, y_{g^{-1}(k)}$, since we know that the original. If we pick these k, it will give us G. □

The run-time for this would be $O(n^2)$.