1 Q1 (25 pts) S.T. : DP Leftovers

Say you are given a set of $n$ integers, $\{a_1, \ldots, a_n\}$, such that each $a_i$ is between 0 and $k$. You now want to partition this set into two disjoints subsets $S_1$ and $S_2$ as fairly as possible – that is, so that the sums of both sets are as close as possible. Formally, we wish to minimize the following quantity:

$$|\sum_{s \in S_1} s - \sum_{s \in S_2} s|$$  \hspace{1cm} (1)

Give a dynamic programming algorithm to find $\{S_1, S_2\}$ and analyze its running time.

**Hint.** If you are stuck, try to find the minimum value of the objective function (1), and then use backtracking to find the actual sets.

2 Q2 (25 pts) O.S. : MSTs

Suppose you are given a weighted graph $G$ as part of your 451 homework. You are asked to compute a minimum spanning tree $T$ of $G$, and immediately come up with one – just like you saw in class. However, as soon as you’re done, your TA finds a typo in the homework and emails everybody a correction. Does it mean you have to start again?

1. Describe an algorithm to update $T$ when the weight of a single edge $e$ is decreased.

2. Describe an algorithm to update $T$ when the weight of a single edge $e$ is increased.

3. Describe an algorithm to update $T$ when a single edge $e$ is removed from the graph.

4. Describe an algorithm to update $T$ when a single edge $e$ is added to the graph.

The input is $G, T$ the edge $e$ and the way it changed; your algorithms should modify $T$ so that it is still a minimum spanning tree. Make them as efficient as you can, and analyze their complexity. Prove their correctness.
3 Q3 (30 pts) T.S. : Connectivity and Cut-vertices

1. (5 pts) In any connected undirected graph \( G \) there is a vertex \( v \) whose removal leaves \( G \) connected. Prove or give a counter example.

2. (15 pts) In a connected undirected graph \( G \), if the removal of some vertex \( v \) results in \( G \) being disconnected, then \( v \) is called a cut-vertex. Give an \( O(V + E) \) time algorithm to find a cut-vertex in a connected undirected graph \( G \).

3. (10 pts) Let \( v \) be a cut vertex of a connected, undirected graph \( G = (V,E) \). Let \( G' = (V,E') \) be the complement of \( G \) (the graph on the same set of vertices such that two vertices \( u,v \) are adjacent in \( G' \) if and only if they are not adjacent in \( G \)). Prove that \( (G' - v) \) is connected.

4 Q4 (20 pts) A.V.G. : Colorings!

Let \( G = (V,E) \) be an undirected graph. A \( k \)-coloring of the vertices of \( G \) is a map \( \phi : V \to [k] \) (so you should think of colors as numbers). A \( k \)-coloring \( \phi \) of \( G \) is proper iff no two vertices of the same color are connected by an edge:

\[ \forall \{u,v\} \in E, \phi(u) \neq \phi(v) \]

A graph is \( k \)-colorable iff there exists a proper \( k \)-coloring of \( G \).

We now describe the greedy algorithm for coloring the vertices of a graph. If \( v_1, v_2, \ldots, v_n \) is an ordering of \( V \), we color the vertices using the following: take the smallest vertex not yet colored and color it with the smallest color that none of its colored neighbors use. Stop when all vertices are colored.

For example: Take \( G \) the 5-cycle over \( V = \{1, 2, 3, 4, 5\} \): That is, \( E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 1\}\} \).

Then we color this graph under the ordering \( 1 < 2 < 3 < 4 < 5 \).

1 has no colored neighbors, so it is assigned color 0.

Then, 2 cannot be colored with 0 since 1 is a neighbor, so it is assigned color 1.

Then, 3 is assigned color 0, since this is the smallest color unused by its colored neighbors (2 had color 1)

Then, 4 is colored with 1.

Finally, 5 has neighbors 4 and 1 of colors 1 and 0 respectively, so 5 is assigned color 2.

In summary, we have \( \phi(1) = 0, \phi(2) = 1, \phi(3) = 0, \phi(4) = 1, \phi(5) = 2 \).

1. Take \( \Delta(G) \) to be the maximum degree of a vertex in \( G \). Show that no matter what ordering on the vertices is used, the above algorithm uses at most \( \Delta(G) + 1 \) colors.

2. Show that the above algorithm can be very bad; for every natural \( n \), construct a 2-colorable graph on \( 2n \) vertices and an ordering of the vertices such that the above algorithm uses \( n \) colors.

3. Show that the above algorithm can be very good; show that, for any \( k \)-colorable graph \( G \), there is an ordering of the vertices such that the above algorithm uses at most \( k \) colors.