CS 161 (Stanford, Fall 2025) Section 8 Extra

1 The Max-Cut Problem

In this question we will investigate several approaches to the Max-Cut problem. Recall that Max-Cut is the following: given an undirected, unweighted graph $G = (V, E)$, find a partition of the vertices into subsets S and $V \setminus S$ that maximizes the number of edges crossing the cut.

The Max-Cut problem is known to be NP-Hard, so we do not expect a polynomial-time exact algorithm. Parts (1)–(2) will ask you to show why certain natural ideas fail, and in part (3) you will design a greedy approximation algorithm.

1. **Modified Ford–Fulkerson.** Assume all edges have weight 1. Enumerate all pairs (s, t), and for each pair compute the *minimum s–t flow*, thinking of it as a way to obtain a Max-Cut (by analogy with the Min-Cut = Max-Flow theorem).

   Find a counterexample demonstrating that this approach does **not** *compute a Max-Cut, and explain why it fails.*

2. **Modified BFS.** Initialize sets S_1 and S_2 to be empty. Start BFS from an arbitrary node, placing it in S_1. Whenever BFS discovers a new node, place it in the *opposite* set from its parent. Return $\{S_1, S_2\}$ as the cut.

   Find a counterexample showing that this BFS-based partition does **not** *necessarily give a Max-Cut.*

3. **Greedy Approximation Algorithm.**

   Since Max-Cut is NP-Hard, we do not expect a polynomial-time algorithm that always finds the optimal cut. However, we can efficiently compute a cut whose size is at least a 1/2-approximation of the maximum.

   Design a greedy algorithm running in $O(m + n)$ time that always returns a cut of size at least $\frac{1}{2} \cdot \text{OPT}$, where OPT is the size of the maximum cut. Provide an English description, an informal correctness justification, and a runtime analysis.

2 Expense Settling

You've gone on a trip with k friends, where friend i paid c_i for the group's expenses. The expenses should be split equally amongst the friends. You would like to develop an algorithm to ensure that everyone gets paid back fairly, but each person should either pay or receive money (not both). In other words, you cannot have everyone that owes money pay one person, and that person distributes the money back to everyone that is owed money.

3 Fear of Negativity

Do our graph algorithms work when the weights are negative? Let's answer that in this problem. Assume that the graph is directed and that all edge weights are integers.

Negative Prim?

Since Prim's algorithm is very similar to Dijkstra, we want to now consider a similar algorithm *Negative-Prim* for computing minimum spanning trees in graphs with negative edge weights. Again, this algorithm adds some number to all of the edge weights to make them all non-negative, then runs Prim's algorithm on the resulting graph, and argues that the Minimum Spanning Tree in the new graph is the same as the MST in the old graph. You can assume that all the edge weights are unique integers.

Negative-Prim(G, s):

- $minWeight =$ minimum edge weight in G
- For each edge $e \in E$ (iterate through all edges in G), set

$$modifiedWeight(e) = w(e) - minWeight.$$

- Let $modifiedG$ be G with weights $modifiedWeight$.
- $T =$ Prim($modifiedG, s$) (run Prim's algorithm starting from s).
- Update T with edges that correspond to graph G.
- Return T.

Please give either an informal explanation of why Negative-Prim computes the correct MST, or a counter-example of an undirected graph with negative edge weights where Negative-Prim does not output the correct minimum spanning tree, as well as an explanation of why it is a valid counter-example.