CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2025**

Homework 2

Due: Thursday, October 11, 2025

Problem 1. We reexamine the balls-and-bins experiment in this question, focusing on aspects other than the maximum load. Suppose we are throwing a number of balls into n bins one by one where each ball chooses one of the bins independently and uniformly at random.

(a) Prove that the expected number of balls we need to throw before every bin has at least one ball inside it is $\Theta(n \log n)$. **(10 points)**

(b) Find the sufficient and necessary asymptotic value for the number of balls we need to throw, so that with constant probability, at least one bin has two or more balls inside it. **(10 points)**

Problem 2. Recall that a cut in an undirected graph $G = (V, E)$ is any partition of vertices $(S, \bar{S})$ of V and $\delta(S)$ denotes the set of edges in the cut S, i.e., the edges between S and $\bar{S}$. In the **minimum cut** problem, we are interested in finding a cut $(S, \bar{S})$ with the minimum number of cut edges, i.e., minimize $|\delta(S)|$.

We examine a by-now classical algorithm for this problem due to Karger.

(a) Suppose S is a minimum cut of G and $e = (u, v)$ is not a cut edge of S. Prove that if we contract u and v into a single vertex and keep the parallel edges (but remove self-loops), then, the minimum cut size of G remains the same after the contraction. **(5 points)**

(b) Fix some minimum cut $(S, \bar{S})$ in G. Prove that if we sample an edge uniformly at random from G, then, the probability that e is a cut edge of S is at most $2/n$. **(10 points)**

Hint: What is the relationship between the number of edges, the minimum degree, and the minimum cut size in a graph?

(c) Consider the following algorithm:

Algorithm 1. Given an undirected multi-graph $G = (V, E)$,

(i) Sample an edge $e = (u, v)$ uniformly at random from G;

(ii) Contract the set $\{u, v\}$ in G to obtain $G_{\{u,v\}}$;

(iii) Recurse on $G_{\{u,v\}}$ and continue until only two vertices are remained; then, return the sets corresponding to these two (possibly) contracted vertices.

Prove that this algorithm outputs a minimum cut of a given graph G with probability at least $\binom{n}{2}^{-1}$. **(10 points)**

(d) Use the above algorithm to prove that in every graph G, there can be at most $\binom{n}{2}$ minimum cuts. **(5 points)**

Problem 3. In this question, we design another simple algorithm for MSTs with runtime better than the classical algorithms (although not as good as the advanced ones we studied). Recall the following two facts:

- Each round of Boruvka's algorithm takes $O(m+n)$ time and reduces the vertices by at least a half.
- Prim's algorithm can be implemented in $O(m+n\log n)$ time using Fibonacci heaps.

Combine these two algorithms to obtain an $O(m\log\log n)$ time algorithm for MSTs.

(20 points)

Problem 4. Suppose we have n pairs of non-negative integers $(a_1,b_1),\dots,(a_n,b_n)$. We are additionally given an integer $B\geqslant 1$. The goal is to find a set $S\subseteq[n]$ such that $\sum_{i\in S} b_i \leqslant B$ while maximizing $\sum_{i\in S} a_i$.

(a) Write this problem as an integer linear program and relax it to an LP relaxation. **(10 points)**

(b) Prove that this LP relaxation has an optimal solution $x^*\in[0,1]^n$ such that at most one variable has a value outside $\{0,1\}$. **(10 points)**

(c) Use the previous two parts to obtain an algorithm that outputs a set S that satisfies $\sum_{i\in S} b_i\leqslant B$ and the value of $\sum_{i\in S} a_i$ is at least half of the maximum possible value. **(10 points)**

Problem 5 (**Extra Credit**)**.** Design a *deterministic* $O(m)$ time algorithm for finding MST of a given *planar* graph. A planar graph is a graph that can be drawn on a surface so that no two edges cross each other.

(+10 points)

Hint: This is a pretty simple question. Think about (or search) how many edges are in a planar graph? And, what happens if you contract an edge in a planar graph?

Problem 6 (**Extra Credit**)**.** Consider the following problem introduced in Lecture 7.

Let $G=(V,E)$ be an n-vertex m-edge undirected graph with weights $w(e)$ for each edge $e\in E$. Suppose $T\subseteq E$ is some spanning tree of G (not necessarily an MST). The goal is to design an algorithm that outputs all ***T*-heavy edges** of G. Recall that an edge $e\in E$ is T-heavy if e is not in T and in the cycle created by adding e to T, e has the maximum weight.

Design a deterministic algorithm for this problem with $O(m\log\log n)$ runtime (or even faster).

Alternatively, you can instead design a randomized algorithm with expected $O(m)$ runtime.

(+10 points)