COMP 5711: Advanced Algorithms
Fall 2021 Midterm Exam

Name: _____ Student ID: _____

Instructions

1. This is an open-book, open-notes exam.
2. Time limit: 80 minutes.
3. You can write on the back of the paper if you run out of space. Please let us know if you need more scratch paper.

1. **Short Questions** (20 pts)

(a) (6 pts) An independent set is a set of vertices in the graph such that no two are adjacent. The vertex cover problem and the independent set problem are equivalent, since on any graph $G = (V, E)$, $C \subseteq V$ is a vertex cover iff $V - C$ is an independent set.

> *The 2-approximation algorithm for the minimum vertex cover problem is also a 2-approximation algorithm for the maximum independent set problem.*

Is this true or false? Give a one-sentence justification for your answer.

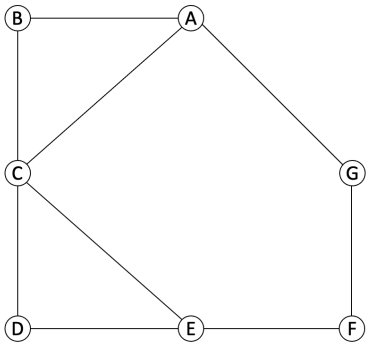
Answer: Suppose the optimal vertex cover is C^*. A 2-approximation algorithm for the vertex cover problem returns a C such that $|C| \leq 2|C^*|$, while a 2-approximation for independent set should return an I such that $|I| \geq \frac{1}{2}(|V| - |C^*|)$. Approximating $|C^*|$ does not mean you also approximate $|V| - |C^*|$.

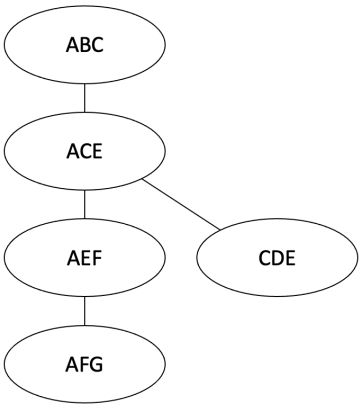
(b) (6 pts) For a problem with input size n and a parameter k, prove that it can be solved in $f(k) \cdot \text{poly}(n)$ time *if and only if* it can also be solved in $g(k) + \text{poly}(n)$ time, where $f(k)$ and $g(k)$ are arbitrary functions of k.

Answer: On one hand, we have, $f(k) \cdot \text{poly}(n) \leq (f(k))^2 + (\text{poly}(n))^2 = g(k) + \text{poly}(n)$. On the other hand, we have $g(k) + \text{poly}(n) \leq (g(k) + 1)(\text{poly}(n) + 1) = f(k) \cdot \text{poly}(n)$.

(c) (8 pts) Draw a tree decomposition with width 2 for the following graph.

Answer:





2. **k-Path** (30 pts) Given an undirected graph $G = (V, E)$ with n vertices and an integer k, the k-path problem is to determine if there is a simple path of length $k-1$ (i.e., it passes k vertices) in G. Recall that a path is simple if it does not pass the same vertex more than once. This problem is known to be NP-complete.

   (a) (15 pts) We first consider a variant of the problem, where each vertex in G is given one of k colors, and we want to determine if there is a *colorful* k-path, i.e., a path passing k vertices of distinct colors. Design an algorithm with running time $f(k) \cdot \text{poly}(n)$.

   (b) (15 pts) Design a randomized algorithm for the k-path problem so that when there exists a k-path, the algorithm returns YES with at least constant probability, and when no k-paths exist, the algorithm always returns NO. Analyze the running time of the algorithm, which should be in the form of $g(k) \cdot \text{poly}(n)$. Note: You can still try to solve this problem using the algorithm from (a), even if you can't solve (a).

Answer: For any subset S of the k colors and any vertex v, let $T(S, v)$ represent whether there exists a colorful $|S|$-path ending at v using the colors in S. Let $c(v)$ be the color of v. For the base case $|S| = 1$, we have $T(S, v) = \mathsf{True}$ if $S = \{c(v)\}$, and False otherwise. For $|S| \geq 2$, we have

$$T(S, v) = \bigvee_{(u,v) \in E} T(S - \{c(v)\}, u),$$

The output of the algorithm is

$$\bigvee_{|S|=k, v \in V} T(S, v).$$

For each vertex $v \in V$, there are 2^k $T(S, v)$'s to compute, and for each $T(S, v)$, it takes $O(n)$ time to compute the result. Therefore, the running time is $O(2^k \cdot n^2)$.

The randomized algorithm is to assign one of the k colors to each vertex uniformly and independently, and then run the algorithm above. Clearly, if there are no k-paths, there are no colorful k-paths and the algorithm will always return NO. Below we consider the case when a k-path exists. The probability that this k-path is colorful is $\frac{k!}{k^k}$. Then we repeat the algorithm $\frac{k^k}{k!}$ times, and return YES if any run returns YES. The probability the algorithm does not return YES when there exists a k-path is at most $(1 - \frac{k!}{k^k})^{\frac{k^k}{k!}} \leq \frac{1}{e}$. And the total running time is $O(\frac{k^k}{k!} 2^k \cdot n^2)$.

3. **FIFO Queue** (20 pts) A First-In-First-Out (FIFO) queue with decimation maintains a sequence of items subject to the following operations.

   (a) PUSH(x): Add item x to the end of the sequence.

   (b) PULL(): Remove and return the item at the beginning of the sequence.

   (c) DECIMATE(): Remove every tenth element from the queue, starting at the beginning of the sequence. The details are shown in Algorithm 1.

Algorithm 1: DECIMATE()

```
1 n ← the current size of the queue;
2 for i ← 0, ..., n − 1 do
3     if i mod 10 = 0 then
4         PULL();
5     else
6         PUSH(PULL());
```

It is easy to implement a queue using a doubly-linked list, so that it uses $O(n)$ space and the worst-case time for PUSH and PULL operations is $O(1)$.

Prove *using the potential method* that in any intermixed sequence of PUSH, PULL, and DECIMATE operations, the amortized cost of each operation is $O(1)$.

Answer: Let the potential function $\Phi(D_i)$ be the size of the queue times some constant c so that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$. We have

(a) For PUSH(x), the amortized cost $c'_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + c$.

(b) For PULL(), the amortized cost $c'_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 - c$.

(c) For DECIMATE(), the amortized cost $c'_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \leq 2n - 0.1cn$.

Let $c = 20$, we prove that the amortized cost of each operation is $O(1)$.

4. **Maximum Cut** (30 pts) Given a graph G with n nodes and m edges, we want to find the maximum cut of the graph, i.e., each node is assigned one of the two colors, and we want to maximize the number of edges with two different colors.

   (a) (10 pts) Design a polynomial-time randomized algorithm that finds a cut with $\frac{m}{2}$ edges in expectation.

   (b) (20 pts) Derandomize your algorithm using the method of conditional expectation so that it guarantees to find a cut with at least $\frac{m}{2}$ edges in polynomial time. You need to describe your algorithm, state how to compute the conditional expectations, analyze the running time, and prove the correctness of your algorithm.

   Answer: The randomized algorithm is to assign each node one of the two colors uniformly and independently. For each edge, the probability that the two nodes have different colors is $\frac{1}{2}$. Therefore, by linearity of expectation, the expected number of edges in the returned answer is $\frac{m}{2}$.

   To derandomize the algorithm, we apply the method of conditional expectation. Given the colors for the first i nodes, the conditional expectation is (# of edges with two different colors so far) + $\frac{1}{2}$(# of edges with at least one node not yet colored). Therefore, for the color of the $(i+1)$-th node, we can calculate the conditional expectations for both colors and set the color to be the one having larger expectation.

   We claim that the algorithm always finds a maximum cut with at least $\frac{m}{2}$ edges. The number of edges, which equals the conditional expectation given the colors for all n nodes, is larger than the conditional expectation given the colors for the first $n-1$ nodes, and etc. Finally, it is larger than the conditional expectation given no color, which is $\frac{m}{2}$.

   For the running time, we need to color n nodes, and for each node, it takes $O(n)$ time to compute the conditional expectations and determine the color. So the total running time is $O(n^2)$.