Algorithms & Theory Qual Spring 2019

Part I: Algorithms

March 4, 2019, Monday, 9am–5pm, SC 4405

Version: 1.0

Print ID:

1 Color collector.

Let $G = (V, E)$ be a directed graph. Let $R \subset V$ and $B \subset V$ be two disjoint non-empty sets of nodes (red and blue nodes). Given a walk W in the graph let r_W of times a red node is visited by W (same node can be visited multiple times and will be counted repeatedly) and similarly let b_W be the number of blue nodes visited by W. Call a graph G *interesting* if for any $i, j \geq 0$ there is a walk W in G such that $r_W \geq i$ and $b_W \geq j$. In other words G has walks that visit red and blue nodes infinitely often.

1.A. [**30 Points**] Describe an efficient (polynomial-time) algorithm that given G, R, B checks whether it is interesting.

1.B. [**20 Points**] A graph is *fascinating* if for any integer $u \geq 1$, there is a walk $w_1, w_2, \ldots, w_t$ in G, such that is has a color alternating subsequence of length at least u. Formally, there are indices $i_1 < i_2 < \ldots < i_u$, such that $w_{i_{2j-1}} \in R$ and $w_{i_{2j}} \in B$, for all j. Describe an example of a graph that is interesting but not fascinating.

1.C. [**50 Points**] Now consider the more general setting where G has nodes with k different colors $C_1, C_2, \ldots, C_k$ (non-empty and disjoint). Now G is interesting if there is a walk that visits nodes in all colors infinitely often. Describe an algorithm that for any fixed k checks whether G is interesting.

2 Rainbow flow.

Let $G = (V, E)$ be a directed graph with integer edge-capacities $c(e), e \in E$ and let $s, t \in V$ be source and sink nodes. Maximum flow is typically formulated as a linear program using edge flow variables $f(e)$. However there are some advantages to viewing flow via path variables. We will explore this. Suppose some of the edges are colored red and some are colored blue and the rest are uncolored; let $R \subset E$ and $B \subset E$ be the red and blue edges respectively (disjoint sets). Let k_r and k_b be non-negative integers and let $\mathcal{P}$ be the set of all s-t paths in G that have at most k_r red edges and at most k_b blue edges. We are interested in a maximum s-t flow where the flow is non-zero only on paths from $\mathcal{P}$: more formally we want to maximize $\sum_{p \in \mathcal{P}} f(p)$ subject to the usual constraint that the total flow on each edge is at most $c(e)$; here $f(p)$ is a variable for amount of flow on path p.

2.A. [**20 Points**] Write down the formal LP formulation for the problem.

2.B. [**20 Points**] Argue why the LP has an optimum solution that can be written in space that is polynomial in the graph size even though the number of variables is potentially exponential.

2.C. [**30 Points**] Write the dual of the LP.

2.D. [**30 Points**] Describe a polynomial-time separation oracle for the dual LP.

3 Making strong connected components faster.

3.A. [**20 Points**] Let G a directed graph G with n vertices and m edges. Consider any **DFS** of G. Prove the following lemma.

Lemma 0.1. *The last vertex in any* **DFS** *postordering of G lies in a source strong connected component of G.*

3.B. [**20 Points**] Using the above, sketch an algorithm, with running time $O(n+m)$, for computing the strong connected components of G.

3.C. [**50 Points**] Assume you are given a data-structure, which can maintain under insertion and deletion a set of vertices $X \subseteq V(G)$, such that given a query vertex u, the data-structure reports a single edge from u to a vertex in X in G (if it exists) – if there are several such edges, it reports arbitrarily one of them. The data-structure also supports the reverse queries, where given u, it reports an edge from a vertex of X to u (if such an edge exists).

Assume that insertion, deletion, query, and reverse query operations all take $O(\log n)$ time. Describe an algorithm with near linear running time (i.e., $O(n\text{polylog}n)$) that computes all the strong connected components of G.

3.D. [**10 Points**] Using the same setting as the previous part, describe an algorithm that computes a set $Y \subseteq V(G)$ of minimum cardinality, such that all the vertices of G are reachable from Y (formally, for any $v \in V(G)$, there exists a $y \in Y$, such that there is a path from y to v in G). The running time of the algorithm has to be $O(n\text{polylog}n)$.

4 Green bits and hamming distance.

For two strings $x = a_1 \cdots a_n \in \Sigma^*$ and $y = b_1 \cdots b_n \in \Sigma^*$, the *Hamming distance* $d_H(x, y)$ is defined as the number of positions i such that $a_i \neq b_i$.

4.A. [**30 Points**] Pick a random function $h : \Sigma \to \{1, \ldots, k\}$ for a given integer k. For two strings $x = a_1 \cdots a_n \in \Sigma^*$ and $y = b_1 \cdots b_n \in \Sigma^*$, define new strings $h(x) = h(a_1) \cdots h(a_n) \in \{1, \ldots, k\}^*$ and $h(y) = h(b_1) \cdots h(b_n) \in \{1, \ldots, k\}^*$.

For a fixed pair of strings x and y and a random h, prove that the probability that $d_H(h(x), h(y)) \geq (1 - 2/k)d_H(x, y)$ is at least $1/2$.

[Hint: first compute the expected value of $d_H(x, y) - d_H(h(x), h(y)) = |\{i \in \{1, \ldots, n\} : a_i \neq b_i \text{ and } h(a_i) = h(b_i)\}|$.]

4.B. [**70 Points**] In the *all-pairs Hamming distance* problem, we are given a set S of n length-n strings over a finite alphabet Σ, and we want to compute the Hamming distance $d_H(x, y)$ for all pairs of strings $x, y \in S$. It is known that this problem can be solved exactly in $O(|\Sigma| n^{2.373})$ time (which beats the naive $O(n^3)$-time algorithm but only when $|\Sigma|$ is not too big). Using this result as a subroutine, describe a randomized approximation algorithm, which computes a $(1 - \varepsilon)$-factor approximation to $d_H(x, y)$ for all pairs $x, y \in S$, in $O((1/\varepsilon)n^{2.373} \log n)$ time. The error probability of your algorithm should be at most $1/n^c$ for an arbitrarily large constant c.