CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 6

Harvard SEAS - Fall 2021 *Due: Wed Oct. 27, 2021 (5pm)*

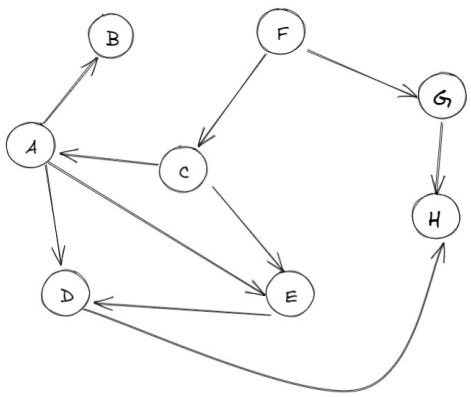
Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

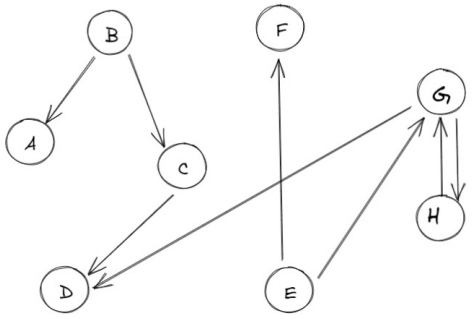
The purpose of this problem set is to provide practice with reductions, modelling problems using graphs, and analyzing examples to understand worst-case performance, all in the context of Shortest Paths, BFS, and Graph Coloring.

1. (Rotating Paths) Suppose we are given k digraphs on the same vertex set, $G_0 = (V, E_0), G_1 = (V, E_1), \ldots, G_{k-1} = (V, E_{k-1})$. For vertices $s, t \in V$, a *rotating path* with respect to $G_0, \ldots, G_{k-1}$ from s to t is a sequence of vertices $v_0, v_1, \ldots, v_\ell$ such that $v_0 = s$, $v_\ell = t$, and $(v_i, v_{i+1}) \in E_{i \bmod k}$ for $i = 0, \ldots, \ell - 1$. That is, we are looking for paths that rotate between the digraphs $G_0, G_1, \ldots, G_{k-1}$ in the edges used.

   (a) Show that the problem of finding a shortest k-rotating path from s to t with respect to $G_0, \ldots, G_{k-1}$ (if one exists) can be reduced to ordinary Single-Source Shortest Paths in a digraph G' on kn vertices and $m_0 + m_1 + \cdots + m_{k-1}$ edges, where $n = |V|$ and $m_i = |E_i|$. Deduce that the shortest k-rotating path can be found in time $O(kn + m_0 + \cdots + m_{k-1})$. To test your algorithm, try running through the example in Part 1b.

   (b) Run your algorithm from Part 1a on the following pair of graphs G_0 and G_1 to find the shortest 2-rotating path from $s = A$ to $t = H$; this will involve solving Single-Source Shortest Paths on a digraph G' with $2 \cdot 8 = 16$ vertices. Fill out the table provided below with the BFS frontier in G' at each iteration, labelling the vertices of G' as $A, B, \ldots, H, A', B', \ldots, H'$, and for each vertex v in the table, drawing an arrow in the table from v's BFS predecessor to v.





d	Frontier F_d
0	$\{A\}$
1	
2	
$\vdots$	

(c) A group of three friends decides to play a new cooperative game. They rotate turns moving a shared single piece on an $n \times n$ grid. The piece starts in the lower-left corner, and their goal is to get the piece to the upper-right corner in as few turns as possible. Many of the spaces on the grid have visible bombs, so they cannot move their piece to those spaces. Each player is restricted in how they can move the piece. Player 0 can move it like a chess-rook (any number of spaces vertically or horizontally, provided it does not cross any bomb spaces). Player 2 can move it like a chess bishop (any number of spaces diagonally in any direction, provided it does not cross any bomb spaces). Player 3 can move it like a chess knight (move to any non-bomb space that is two steps away in a horizontal direction and one step away in a vertical direction or vice-versa). Using Part 1b, show that given the $n \times n$ game board (i.e., the locations of all the bomb spaces), they can find the quickest solution in time $O(n^3)$. (Hint: give a reduction, mapping the given grid to an appropriate instance $(G_0, G_1, \ldots, G_{k-1}, s, t)$ of Shortest k-Rotating Paths.)

2. (Greedy Coloring)

(a) Show that if a graph is k-colorable, then there is an ordering of the vertices of G under which Greedy Coloring will use at most k colors. Why doesn't this imply that 3-coloring can be solved in time $O(n + m)$?

(b) For any parameter $t \in \mathbb{N}$, consider a graph $G = (V, E)$ with the $2t + 1$ vertices $V = \{v_0, v_1, v_2, \ldots, v_t, v'_1, \ldots, v'_t\}$ and the $t^2 + t$ edges

$$E = \{\{v_0, v_i\} : 1 \leq i \leq t\} \cup \{\{v_0, v'_i\} : 1 \leq i \leq t\} \cup \{\{v_i, v'_j\} : 1 \leq i, j \leq t, i \neq j\}.$$

Prove that G is 3-colorable but Greedy Coloring with BFS ordering, with an appropriate start vertex and ordering of adjacency lists, can use as many as $t + 1$ colors. (In the standard implementation of BFS, the frontier is implemented as a queue, and when a vertex is removed from the front of the queue, its outneighbors that haven't been previously visited are added to the back of the queue. Thus the start vertex and the ordering of the adjacency lists suffice to determine the order in which vertices are visit.)