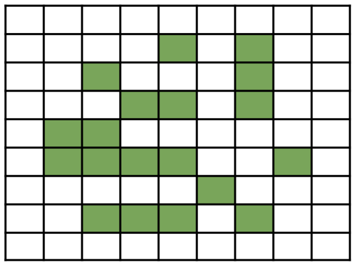
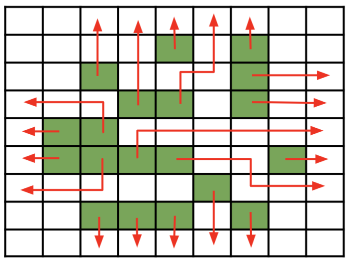
CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2019**

“Homework” #6

December 10, 2019

Problem 1. You are given an $n \times n$ matrix and a set of k cells $(i_1, j_1), \ldots, (i_k, j_k)$ on this matrix. We say that this set of cells can *escape* the matrix if: (1) we can find a path from each cell to any arbitrary *boundary cell* of the matrix (a path is a sequence of *neighboring* cells, namely, top, bottom, left, and right), (2) these paths are all *disjoint*, namely, no cell is used in more than one of these paths. See Figure 1 for an example.


(a) An “escapable” input


(b) A way of escaping the matrix

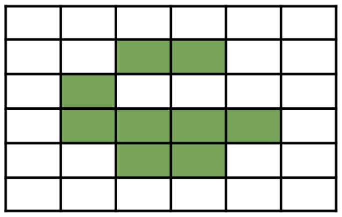

(c) A “non-escapable” input

Figure 1: The green cells correspond to the input k cells of the matrix.

Design an $O(n^3)$ time algorithm that given the matrix and the input cells, determines whether these cells can escape the matrix (together) or not[1]. **(25 points)**

Hint: Whenever you want to find a collection of paths with “bounded intersection” (totally disjoint in this problem), try network flow first.

Problem 2. A *path cover* of a directed graph $G(V, E)$ is a collection of directed paths $P_1, \ldots, P_k$ such that every vertex of v appears in *exactly* one of these paths. Our goal in this problem is to study algorithms for finding a *minimum* path cover (a path cover of minimum size).

(a) Design an algorithm that finds a path cover of a given *directed acyclic graph (DAG)* in $O(mn)$ time. **(12.5 points)**

Hint: Consider the bipartite graph obtained by making two copies of each vertex v called v^l and v^r, and adding an edge $\{u^l, v^r\}$ to this undirected graph whenever (u, v) is an edge in the original graph. Prove that the *maximum matching* size in this graph is $n - k$ if and only if the size of the minimum path cover is k, and use the algorithm for maximum matching (by network flow) to solve this problem.

(b) Prove that this problem is NP-hard in general, i.e., when G is no longer a DAG but any arbitrary directed graph. In order to do so, use a reduction from the following NP-hard problem:

- **Hamiltonian Path Problem**: Given a directed graph $G(V, E)$ is there a path in G that passes through every vertex, namely, a Hamiltonian path?

(12.5 points)

[1]This (type of) problem is used typically to connect multiple components of a circuit to the power sources that can only be placed at the boundary of the circuit – disjointness of the paths ensures that the components of the circuit do not interfere with each other.

Problem 3. Recall that in the class, we focused on *decision* problems when defining NP. Solving a decision problem simply tells us whether a solution to our problem exists or not but it does not provide that solution when it exists. Concretely, let us consider the 3-SAT problem on an input formula Φ. Solving 3-SAT on Φ would tell us whether Φ is satisfiable or not but will not give us a satisfying assignment when Φ is satisfiable. What if our goal is to actually find the satisfying formula when one exists? This is called a *search* problem.

It is easy to see that a search problem can only be "harder" than its decision variant, or in other words, if we have an algorithm for the search problem we will obtain an algorithm for the decision problem as well. Interestingly, the converse of this is also true for all NP problems and we will prove this in the context of the 3-SAT problem in this problem. In particular, we reduce the 3-SAT-SEARCH problem (the problem of finding a satisfying assignment to a 3-CNF formula) to the 3-SAT (decision) problem (the problem of deciding whether a 3-CNF formula has a satisfying assignment or not).

(a) Suppose you are given, as a black-box, an algorithm A for solving 3-SAT (decision) problem that runs in polynomial time. Design a poly-time algorithm that given a 3-CNF formula Φ and a *single* variable a in Φ, decides whether there exists a satisfying assignment of Φ in which $a =$True or not (note that this is still a decision problem). **(10 points)**

(b) Use your algorithm from the first part to find a satisfying assignment of a given 3-CNF formula Φ or output that no such assignment exists, i.e., solve the 3-SAT-SEARCH problem. Your algorithm should run in polynomial time assuming that you are given a poly-time algorithm A for solving 3-SAT decision problem. **(15 points)**

Hint: Iterate over variables of Φ one by one and use part (a) to decide whether they should be assigned True or False iteratively.

Problem 4. Prove that each of the following problems is NP-hard and for each problem determine whether it is also NP-complete or not.

(a) **7-Degree Spanning Tree:** Given an undirected graph $G(V, E)$, does G contain a spanning tree in which every vertex has a degree of *at most* 7? **(6 points)**

(b) 1/2-**Path:** Given an undirected *connected* graph $G(V, E)$, does G contain a path that passes through *more than* half of the vertices in G? **(6 points)**

(c) 4-**Coloring:** Given an undirected graph $G(V, E)$, is there a 4-coloring of G? (A 4-coloring is an assignment of colors $\{1, 2, 3, 4\}$ to vertices so that no edge gets the same color on both its endpoints). **(6 points)**

(d) **Minimum Vertex Cover:** Given an undirected graph $G(V, E)$, what is the size of minimum vertex cover in G? (A vertex cover is a set of vertices such that every edge has at least one end point in the vertex cover). **(7 points)**

You may assume the following problems are NP-hard for your reductions:

- **Undirected** s-t **Hamiltonian Path:** Given an undirected graph $G(V, E)$ and two vertices $s, t \in V$, is there a Hamiltonian path from s to t in G? (A Hamiltonian path is a path that passes every vertex).
- **Maximum Independent Set:** Given an undirected graph $G(V, E)$, what is the size of maximum independent set in G? (An independent set is a collection of vertices with no edges between them).
- **3-Coloring:** Given an undirected graph $G(V, E)$, is there a 3-coloring of G? (A 3-coloring is an assignment of colors $\{1, 2, 3\}$ to vertices so that no edge gets the same color on both its endpoints).

Fun with ~~Algorithms~~ Lower Bounds. You are given a puzzle consists of an $m \times n$ grid of squares, where each square can be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors.

It is easy to see that for some initial configurations of stones, reaching this goal is impossible. We define the Puzzle problem as follows. Given an initial configuration of red and blue stones on an $m \times n$ grid of squares, determine whether or not the puzzle instance has a feasible solution.

Prove that the Puzzle problem is NP-complete.

Challenge Yourself. The goal of this question is to give a simple proof that there are decision problems that admit *no* algorithm at all (independent of the runtime of the algorithm).

Define Σ^+ as the set of all *binary* strings, i.e., $\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$. Observe that any decision problem Π can be identified by a function $f_\Pi : \Sigma^+ \rightarrow \{0, 1\}$. Moreover, observe that any algorithm can be identified with a binary string in Σ^+. Use this to argue that "number" of algorithms is "much smaller" than "number" of decision problems and hence there should be some decision problems that cannot be solved by any algorithm.

Hint: Note that in the above argument you have to be careful when comparing "number" of algorithms and decision problems: after all, they are both infinity! Use the fact that *cardinality* of the set of real numbers $\mathbb{R}$ is larger than the cardinality of integer numbers $\mathbb{N}$ (if you have never seen the notion of cardinality of an infinite set before, you may want to skip this problem).