Theory Problems

Course: Coursera Algorithms Specialization
Name: Bryan Yaggi

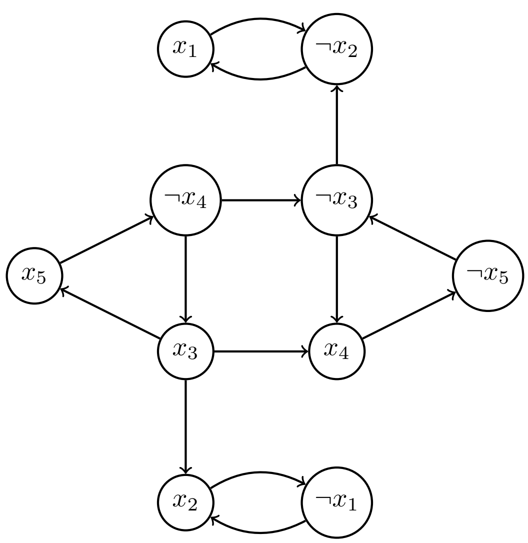
Problem 1

In the 2SAT problem, you are given a set of clauses, where each clause is the disjunction of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value "true" or "false" to each of the variables so that all clauses are satisfied — that is, there is at least one true literal in each clause. For this problem, design an algorithm that determines whether or not a given 2SAT instance has a satisfying assignment. (Your algorithm does not need to exhibit a satisfying assignment, just decide whether or not one exists.) Your algorithm should run in $O(m+n)$ time, where m and n are the number of clauses and variables, respectively. [Hint: strongly connected components.]

Each clause, a disjunction of 2 literals, can be represented as 2 implications. These implications can be used to create an implication graph. If the implication graph has any cycles containing a variable and its negation, a contradiction exists and therefore no satisfying assignment exists. Kosaraju's algorithm can be used to find the strongly connected components of a graph. It uses two depth-first searches, so it has $O(m+n)$ complexity. If a variable and its negation are present in the same strongly connected component, no satisfying assignment exists.

Example: Given $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee x_5) \wedge (\neg x_4 \vee \neg x_5) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2)$

$$\begin{aligned}
(x_1 \vee x_2) &\leftrightarrow (\neg x_1 \to x_2) \wedge (\neg x_2 \to x_1) \\
(\neg x_2 \vee x_3) &\leftrightarrow (x_2 \to x_3) \wedge (\neg x_3 \to \neg x_2) \\
(x_3 \vee x_4) &\leftrightarrow (\neg x_3 \to x_4) \wedge (\neg x_4 \to x_3) \\
(\neg x_3 \vee x_5) &\leftrightarrow (x_3 \to x_5) \wedge (\neg x_5 \to \neg x_3) \\
(\neg x_4 \vee \neg x_5) &\leftrightarrow (x_4 \to \neg x_5) \wedge (x_5 \to \neg x_4) \\
(\neg x_3 \vee x_4) &\leftrightarrow (x_3 \to x_4) \wedge (\neg x_4 \to \neg x_3) \\
(\neg x_1 \vee \neg x_2) &\leftrightarrow (x_1 \to \neg x_2) \wedge (x_2 \to \neg x_1)
\end{aligned}$$



A satisfying assignment exists.

Problem 2

In lecture we define the length of a path to be the sum of the lengths of its edges. Define the bottleneck of a path to be the maximum length of one of its edges. A mininum-bottleneck path between two vertices s and t is a path with bottleneck no larger than that of any other $s - t$ path. Show how to modify Dijkstra's algorithm to compute a minimum-bottleneck path between two given vertices. The running time should be $O(m \log n)$, as in lecture.

Answer...

Problem 3

We can do better. Suppose now that the graph is undirected. Give a linear-time $O(m)$ algorithm to compute a minimum-bottleneck path between two given vertices.

Answer...

Problem 4

What if the graph is directed? Can you compute a minimum-bottleneck path between two given vertices faster than $O(m \log n)$?

Answer...

Problem 5

Recall that a set H of hash functions (mapping the elements of a universe U to the buckets $0, 1, 2, \ldots, n-1$) is universal if for every distinct $x, y \in U$, the probability $Prob[h(x) = h(y)]$ that x and y collide, assuming that the hash function h is chosen uniformly at random from H, is at most $\frac{1}{n}$. In this problem you will prove that a collision probability of $\frac{1}{n}$ is essentially the best possible. Precisely, suppose that H is a family of hash functions mapping U to $0, 1, 2, \ldots, n-1$, as above. Show that there must be a pair $x, y \in U$ of distinct elements such that, if h is chosen uniformly at random from H, then $Prob[h(x) = h(y)] \geq \frac{1}{n} - \frac{1}{|U|}$

Answer...