CSC4120 Spring 2024 - Written Homework 6

Yohandi 120040025
Andrew Nathanael 120040007

March 27, 2024

Problem 1. Cycle Detection with BFS

A cycle is a path of edges from a node to itself. You are given a directed graph $G = (V, E)$, and a special vertex v. Give an algorithm based on BFS that determines in $O(|V|+|E|)$ time whether v is part of a cycle.

```
1. Initialize a queue $Q$ and an array in-degree of length $|V|$ to store the in-degree of
   each vertex containing 0 values.
2. For each edge $(u, v) \in E$, increment in-degree$[v]$.
3. For each vertex $w \in V$, if in-degree$[w] = 0$, enqueue $w$ into $Q$.
4. While $Q$ is not empty:
   (a) Dequeue an element from $Q$, let's say it is $u$.
   (b) For each $w$ such that $(u, w) \in E$: // store by adjacency list so $\mathcal{O}(|E|)$ in total.
       i. Decrement in-degree$[w]$ by 1.
       ii. If in-degree$[w] = 0$, enqueue $w$ into $Q$.
5. If in-degree$[v] = 0$, return **false**.
6. Initialize an array visited of length $|V|$ to store visited status of each vertex containing
   false values.
7. Enqueue $v$ into $Q$.
8. While $Q$ is not empty:
   (a) Dequeue an element from $Q$, let's say it is $u$.
   (b) For each $w$ such that $(u, w) \in E$: // store by adjacency list so $\mathcal{O}(|E|)$ in total.
       i. If $w = v$, return **true**.
       ii. If visited$[w]$ is false, enqueue $w$ into $Q$ and set visited$[w]$ as true.
9. Return **false**.
```

Problem 2. Reachability

We are given a directed graph $G = (V, E)$. We like to find out in time $O(|V| + |E|)$ if there is a vertex $v \in V$ that has the property that starting from v we can reach all the other vertices in V.

1. Compute the SCCs of G using Kosaraju's algorithm in $O(|V| + |E|)$ time.

2. Construct a condensation graph G' by creating a vertex for each SCC and a reversed edge for each pair of vertices if there exists an edge in G between any two vertices in the corresponding SCCs.

3. If there exists a sink in G', say it is v', then starting from any vertex in the SCC corresponding to v' in the original graph G, one can reach all other vertices.

4. Otherwise, no vertex in G can reach all others.

Problem 3.

A bipartite graph is a graph $G = (V, E)$ whose vertices can be partitioned into two sets $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$ such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between u and v).

(a) Give a linear-time algorithm to determine whether an undirected graph is bipartite.

(b) There are many other ways to formulate this property. For instance, an undirected graph is bipartite if and only if it can be colored with just two colors. Prove the following formulation: an undirected graph is bipartite if and only if it contains no cycles of odd length.

(c) At most how many colors are needed to color in an undirected graph with exactly one odd-length cycle?

(a) Let's first define a procedure `BFS-check(s)` as follows:

```
1 Initialize Q as an empty queue
2 color[s] = 1, Q.enqueue(s)
3 While Q is not empty:
    ◦ u = Q.dequeue()
    ◦ For each $(u, v) \in E$:
        ◦ if color[v] is -1: color[v] = 1 - color[u], Q.enqueue(v)
        ◦ else if color[v] is color[u]: return **false**
    ◦ return **true**
```

Our main algorithm will be:

```
1 Initialize an array color[V] with values -1.
2 For each vertex $u \in V$, if color[u] is -1, we run BFS-check(u). If the procedure
  returns **false**, then we return **false**.
3 Return **true** if the above for loop has not yet returned **false**.
```

(b) Proof:

($\Rightarrow$) Assume G is a bipartite graph and partitioned into V_1 and V_2. If G has a cycle of odd length consisting of u and other vertices and $u \in V_1$ (the case of $u \in V_2$ can be interchanged accordingly), then we will alternate between some vertices in V_1 and V_2 and eventually stop at a vertex in V_1. This contradicts the fact that G is a bipartite graph, since u and the end vertex should not be in the same set V_1.

($\Leftarrow$) Assume G has no cycles of odd length. If we use BFS to color G with two colors, starting from an arbitrary vertex, there won't be any conflict as there are no odd-length cycles. This implies that G is bipartite.

Thus, an undirected graph is bipartite if and only if it contains no cycles of odd length.

(c) At least three colors must be used to color a graph with exactly one odd-length cycle as it has been shown that it is impossible to color the graph with two colors. Claim that we can color the graph using three colors.

Suppose that the odd length cycle contains u, then for each vertex in the cycle (we move from u) we will color them alternately. Right before it reaches the last vertex, say v, that connects to u, we will color v as the third color. With this, there won't be any conflict.

The alternative coloring works as if we remove v from the graph, the graph will be bipartite and can be painted using exactly two colors.

Problem 4.

Assume that you are also given the 'reverse' edges: for each course v, the courses w that are its prerequisites.

Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course u if and only if v is a prerequisite for u. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). The running time of your algorithm should be linear.

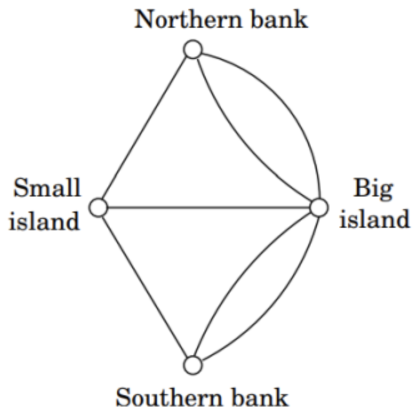
We will greedily take all courses that can be taken in the current semester. This can be done using Kahn's algorithm that combines topological sort and BFS.

1. Initialize an empty queue Q and `in-degree` as an array that maintains the number of in degrees for all vertices.
2. For each edge (u, v), we increment `in-degree`$[v]$.
3. For each course v, if `in-degree`$[v] = 0$, enqueue v into Q.
4. For `num-of-semesters` in $[1, \dots, n]$:

```
    ◦ If $Q$ is empty: return num-of-semesters $- 1$.
    ◦ While $Q$ is not empty:
        ◦ Dequeue $u$ from $Q$.
        ◦ For each edge $(u, v)$:
            ◦ Decrement in-degree[$v$] by 1.
            ◦ If in-degree[$v$] $= 0$, enqueue $v$ into $Q$.
5. return $n$ if the above for loop has not yet returned the number of semesters.
```

Assuming the representation of edges uses adjacency list, then the time complexity will be $\mathcal{O}(|V| + |E|)$.

Problem 5.



An *Eulerian tour* in an undirected graph is a cycle that is allowed to pass through each vertex multiple times, but must use each edge exactly once. This simple concept was used by Euler in 1736 to solve the famous Konigsberg bridge problem, which launched the field of graph theory. The city of Konigsberg (now called Kaliningrad, in western Russia) is the meeting point of two rivers with a small island in the middle. There are seven bridges across the rivers, and a popular recreational question of the time was to determine whether it is possible to perform a tour in which each bridge is crossed *exactly once*. Euler formulated the relevant information as a graph with four nodes (denoting land masses) and seven edges (denoting bridges), as shown above.

(a) Show that an undirected graph has an Eulerian tour if and only if all its vertices have even degree. Conclude that there is no Eulerian tour of the Konigsberg bridge problem.

(b) Can you give an analog of part (a) for directed graphs?

(a) Proof:

(⇒) Assume G has an Eulerian tour. During the tour, each time a vertex is visited, one edge is used to arrive and another to leave, except for the start/end vertex. Thus, every vertex must have an even degree to match the in-and-out pattern.

(⇐) Assume all vertices in G have even degree. Start at any vertex and follow edges until returning to the starting vertex. If all edges are used, the tour is complete. If not, since all degrees are even, we can start a new tour from an unused edge connected to the already visited path, eventually covering all edges without leaving any edge unused.

In the Konigsberg bridge problem, we have vertices' degrees: small island (3), big island (3), northern bank (3), southern bank (3). None have even degrees, so an Eulerian tour is impossible.

(b) In a directed graph has an Eulerian tour if and only if all its vertices have equal number of in-degrees and out-degrees and the graph is strongly connected. Proof:

(⇒) Assume D has an Eulerian tour. During the tour, each time a vertex is visited, one edge is used to arrive and another to leave. Thus, every vertex must have an equal number of entries (in-degrees) and exits (out-degrees). Moreover, D must also be strongly connected as a result of the tour existence.

(⇐) Assume D is strongly connected and each vertex has equal in-degree and out-degree. Start a tour at any vertex. Similar to the undirected case, one can follow the edges. The equal in-degree and out-degree ensures every time there's an entry, there's an exit, allowing a return to the start without leaving any edge unused. If all edges are not covered in the first attempt, the strong connectivity and equal degrees guarantee that remaining edges can be incorporated into the tour by starting new sub-tours and merging them.

Problem 6.

In the 2SAT problem, you are given a set of *clauses*, where each clause is the disjunction (OR) 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 example, here's an instance of 2SAT:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_4).$$

This instance has a satisfying assignment: set x_1, x_2, x_3, and x_4 to true, false, false, and true, respectively.

(a) Are there other satisfying truth assignments of this 2SAT formula? If so, find them all.

(b) Give an instance of 2SAT with four variables, and with no satisfying assignment.

The purpose of this problem is to lead you to a way of solving 2SAT efficiently by reducing it to the problem of finding the strongly connected components of a directed graph. Given an instance of 2SAT with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows:

- G_I has $2n$ nodes, one for each variable and its negation.
- G_I has $2m$ edges: for each clause $(\alpha \vee \beta)$ of I (where α, β are literals), G_I has an edge from the negation of α to β, and one from the negation of β to α.

Note that the clause $(\alpha \vee \beta)$ is equivalent to either of the implications $\alpha \Rightarrow \beta$ or $\beta \Rightarrow \alpha$. In this sense, G_I records all implications in I.

(c) Carry out this construction for the instance of 2SAT given above, and for the instance you constructed in (b).

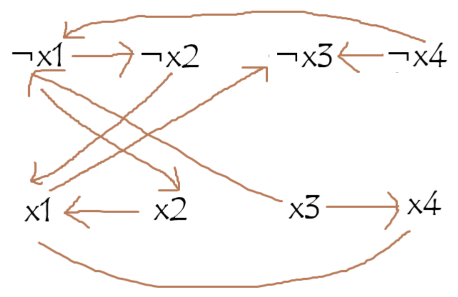
(d) Show that if G_I has a strongly connected component containing both x and $\bar{x}$ for some variable x, then I has no satisfying assignment.

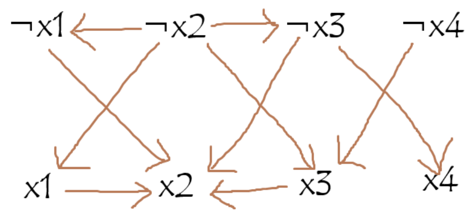
(e) Now show the converse of (d): namely, that if none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

(f) Conclude that there is a linear-time algorithm for solving 2SAT.

(a) Yes. The only other satisfying assignment: set x_1, x_2, x_3, and x_4 to true, true, false, true, respectively.

(b) $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_3 \vee x_2) \wedge (\neg x_3 \vee x_2) \wedge (x_3 \vee x_4)$





(c)

(d) The presence of both x and $\bar{x}$ within the same SCC implies there's a path from x to $\bar{x}$ and vice versa, signifying that x implies $\bar{x}$ and $\bar{x}$ implies x. In terms of logical consistency and truth values, this is impossible because it requires x to be both true and false simultaneously, which contradicts the fundamental principle of binary logic. Therefore, if such an SCC exists, I cannot be satisfied by any assignment of truth values to its variables.

In my construction of part (b), which is shown in the above figure, it can be noticed that $\neg x_2$ and x_2 form a SCC.

(e) There is no variable for which both the variable and its negation can be reached from each other, eliminating the possibility of a logical contradiction. Since G_I encapsulates all implications ($\alpha \Rightarrow \beta$ and $\beta \Rightarrow \alpha$ for each clause $(\alpha \vee \beta)$) in I, the absence of contradictory SCCs means that it is possible to assign truth values to all variables in a way that satisfies all clauses. Specifically, one can assign truth values starting from the variables in the SCCs with no outgoing edges to the rest of the graph and work backward, ensuring that all implications (and thus, all clauses) are satisfied.

(f) We can conclude that there exists a linear-time algorithm for solving 2SAT problems. The algorithm is as follows:

(a) For a 2SAT instance with n variables and m clauses, construct the directed graph G_I as described, with $2n$ nodes and $2m$ edges corresponding to the implications derived from the clauses.

(b) By using Kosaraju's algorithm, find all strongly connected components of G_I.

(c) Check for the contradictory on the SCCs to determine if any contain both a variable and its negation. If such an SCC is found, the 2SAT instance is unsatisfiable.

(d) If there is no contradictory, proceed to assign truth values to the variables in a manner consistent with the implications encoded in G_I.