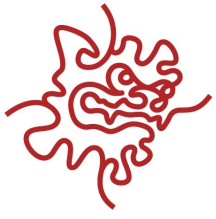
SE222: Principle of Algorithms

Dec 18, 2019



OIST

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY

Overview

Homework 1

[DPV07]. 3.7, 3.11, 3.28, 4.11, 4.12, 4.16

[Als99]. 9.18, 9.33

Homework 2

[DPV07]. 8.3, 8.7

[Als99]. 10.3, 10.5, 10.9, 10.19, 10.22

Homework 3

[DPV07]. 1.8, 1.20, 1.22, 1.31, 1.34, 1.35

Homework 4

[Als99]. 15.6, 15.10, 15.12, 15.17, 15.19, 15.27

[DPV07]. 3.7

A *bipartite graph* is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$), and 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* length cycle?

[DPV07]. 3.7

A *bipartite graph* is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$), and 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.

Solution: BFS or DFS. The graph is bipartite if and only if there are no edges between two vertices of the same color. The running time of BFS/DFS would be $O(|V| + |E|)$.

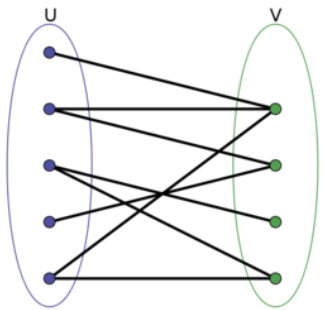


Figure: Bipartite Graph

[DPV07]. 3.7

A *bipartite graph* is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$), and 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).

(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.

Solution: $A \implies B$. Consider a cycle $C = (v_1, v_2, ..., v_{|C|-1}, v_1)$ in the graph. Suppose that $v_1 \in V_1$. Since G is bipartite, $v_3, ..., v_{2k+1}, ... \in V_1$ and $v_2, ..., v_{2k}, ...v_{|C|-1} \in V_2$.

[DPV07]. 3.7

A *bipartite graph* is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$), and 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).

(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.

Solution: $B \implies A$. Run coloring algorithm, starting at arbitrary vertex s. Let R be the set of red vertices and B be the set of blue vertices. Note that $R \cap B = \emptyset$.
Suppose for a contradiction that $r_1, r_2 \in R$ are adjacent. Since $r_1, r_2 \in R$, there must be an even length path from s to r_1, called $P(s, r_1)$, and similarly an even length path from s to r_2, called $P(s, r_2)$. Then $P(s, r_1) - P(r_1, r_2) - P(r_2, s)$ is an odd length cycle, a contradiction.

[DPV07]. 3.7

A *bipartite graph* is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$), and 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).

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

Solution: 3

[DPV07]. 3.11

Design a linear-time algorithm which, given an undirected graph G and a particular edge e in it, determine whether G has a cycle containing e.

Solution: For edge $e(u, v)$, determine whether $G(V, E - e)$ has a path from u to v. Run BFS/DFS to see if v is reachable from u, which has a linear running time.

[DPV07]. 4.12

Give an $O(|V|^2)$ algorithm for the following task.
Input: An undirected graph $G = (V, E)$; edge lengths $l_e > 0$; an edge $e \in E$.
Output: The length of the shortest cycle containing edge e.

Solution: For edge $e(u, v)$, run Dijkstra to find the shortest path from u to v in $G(V, E - e)$.

[DPV07]. 3.28

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.

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee x_4)$$

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

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

Solution: Set x_1, x_2, x_3, x_4 to true, true, false, and true, respectively.

[DPV07]. 3.28

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.

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee x_4)$$

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

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

Solution:

$$(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_3)$$

[DPV07]. 3.28

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 connnected components of a directed graph. Given an instance I 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 implication $(\bar{\alpha} \implies \beta)$ or $(\bar{\beta} \implies \alpha)$. In this sense, G_I records all implications in I.

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

Solution: If x and $\bar{x}$ are in the same strongly connnected component could we get $(x \implies \bar{x})$ and $(\bar{x} \implies x)$, that is $x = \bar{x}$.

[DPV07]. 3.28

(e) Now show the converse of (d)

Solution: Assign values to the variables as follows:

- Repeatedly pick a sink SCC of G_I.
- Assign value true to all literals in the sink, assign false to their negations and delete all of them.

Notice that an implication $(\bar{\alpha} \implies \beta)$ is always satisfied if literal β is true. Thus, setting the literals L in the sink SCC to true ensures that all implications within this SCC are satisfied.
By the symmetry of edges in G_I, for each edge $(\bar{\alpha} \implies \beta)$ in the sink SCC, there is another edge $(\bar{\beta} \implies \alpha)$ elsewhere. Thus the negations of the literals in the sink SCC form a source SCC.
Likewise, $(\bar{\beta} \implies \alpha)$ is always satisfied if $\bar{\beta}$ is false.

[DPV07]. 3.28

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

Solution:

- Build G_I.
- Run the linear-time SCC algorithm.
- For each variable x, check whether x and $\bar{x}$ are in the same SCC.

[DPV07]. 4.11

Give an algorithm that takes as input a directed graph with positive edge lengths, and returns the length of the shortest cycle in the graph(if the graph is acyclic, it should say no). Your algorithm should take time at most $O(|V|^3)$.

Solution1: Dijkstra algorithm. Pick any edge $e = (u, v)$ in the shortest cycle. Then this cycle consists of the shortest path from u to v(which can be found by Dijkstra algorithm).
A naive implementation of Dijkstra's algorithm has a running time of $O(|V|^2)$. Each node u is processed. The total running time is then $O(|V|^3)$.

Solution2: Floyd-Warshall algorithm.

[DPV07]. 4.16

Section 4.5.2 describes a way of storing a complete binary tree of n nodes in an array indexed by $1, 2, \ldots, n$.
(a) Consider the node at position j of the array. Show that its parent is at position $\lfloor j/2 \rfloor$ and its children are at $2j$ and $2j + 1$ (if these numbers are $\leq n$).

(b) What the corresponding indices when a complete d-ary tree is stored in an array?

(c) Show that the makeheap procedure takes O(n) time when called on a set of n elements. What is the worst-case input?

(d) What needs to be changed to adapt this pseudocode to d-ary heaps?

[Als99]. 9.18

An edge of a connected undirected graph G is called a bridge if its deletion disconnects G. Modify the algorithm for finding articulation points so that it detects bridges instead of articulation points.

Solution:

- Articulation Points: $low[v] \geq dfn[u]$
- Bridges: $low[v] > dfn[u]$

[Als99]. 9.33

Design an algorithm to find a cycle of shortest length in a directed graph. Here the length of a cycle is measured in terms of its number of edges.

Solution: Similar to 4.11

Have Fun!