Graph Algorithms

1 Applications

Problem 1: 2-SAT (DPV)

In the 2-SAT 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:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

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

1. Are there other satisfying truth assignments of this 2SAT formula? If so, find them all.
2. 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 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 $2m$ edge: 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 $\overline{\alpha} \rightarrow \beta$ or $\overline{\beta} \rightarrow \alpha$. In this sense G_I records all implications in I.

3. Carry out this construction for the instance of 2SAT given above, and for the instance you constructed in (2).
4. Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.
5. Now show the converse of (4): 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. (Hint: Assign values to the variables as follows: repeatedly pick a sink strongly connected component of G_I. Assign value true to all literals in the sink, assign false to their negations, and delete all of these. Show that this ends up discovering a satisfying assignment.)
6. Conclude that there is a linear-time algorithm for solving 2SAT.

Problem 2: Pouring water (DPV)

We have three containers whose sizes are 10 pints, 7 pints and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

1. Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

2. What algorithm should be applied to solve the problem?

2 Depth-First Search

Problem 3: Reverse of a graph (DPV)

The *reverse* of a directed graph $G = (V, E)$ is another directed graph $G^R = (V, E^R)$, on the same vertex set, but with all edges reversed; that is, $E^R = \{(v, u) : (u, v) \in E\}$.

Give a linear-time algorithm for computing the reverse of a graph in adjacency list format.

Problem 4: Satisfiability of equality constraints

For a set of variables $x_1, \cdots, x_n$, you are given some equality constraints of the form $x_i = x_j$ and some unequality constraints of the form $x_i \neq x_j$. Is it possible to satisfy all of them?

For instance, the constraints

$$x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$$

cannot be satisfied. Give an efficient algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied.

Problem 5: Ancestors and edges (DPV)

Either prove or give a counterexample: if $\{u, v\}$ is an edge in an undirected graph, and during depth-first search $\text{post}(u) < \text{post}(v)$, then v is an ancestor of u in the DFS tree.

Problem 6: Ancestry queries (DPV)

You are given a binary tree $T = (V, E)$ (in adjacency list format), along with a designated root node $r \in V$. Recall that u is said to be an *ancestor* of v in the rooted tree, if the path from r to v in T passes through u.

You wish to preprocess the tree so that queries of the form "is u an ancestor of v?" can be answered in constant time. The preprocessing itself should take linear time. How can this be done?

Problem 7: Articulation points, bridges, and biconnected components (CLRS)

Let $G = (V, E)$ be a connected, undirected graph. An *articulation point* of G is a vertex whose removal disconnects G. A *bridge* of G is an edge whose removal disconnects G. A *biconnected component* of G is a maximal set of edges such that any two edges in the set lie on a common simple cycle. We can determine articulation points, bridges, and biconnected components using depth-first search. Let $G_\pi = (V, E_\pi)$ be a depth-first tree of G.

1. Prove that the root of G_π is an articulation point of G if and only if it has at least two children in G_π.

2. Let v be a nonroot vertex of G_π. Prove that v is an articulation point of G if and only if v has a child s such that there is no back edge from s or any descendant of s to a proper ancestor of v.

3. Let
$$low[v] = \min \begin{cases} d[v] \\ d[w] : (u, w) \text{ is a back edge for some descendant } u \text{ of } v \end{cases}$$
Show how to compute $low[v]$ for all vertices $v \in V$ in $O(E)$ time.

4. Show how to compute all articulation points in $O(E)$ time.
5. Prove that an edge of G is a bridge if and only if it does not lie an any simple cycle of G.
6. Show how to compute all the bridges of G in $O(E)$ time.
7. Prove that the biconnected components of G partition the nonbridge edges of G.
8. Give an $O(E)$-time algorithm to label each edge e of G with a positive integer $bcc[e]$ such that $bcc[e] = bcc[e']$ if and only if e and e' are in the same biconnected component.

Problem 8: Broadcast Times for a Tree

Let T be a rooted binary tree whose edges are given positive real weights, representing *message delay times*. For a node $x \in T$, define the *broadcast time* for x to be the maximum over $y \in T$ of the weighted distance in T between x and y. (In other words, the broadcast time is the time before a message originating at x is received by every other processor y.) Give an efficient algorithm to compute all the broadcast times for nodes in T simultaneously.

Problem 9: Cheapest node (DPV)

You are given a directed graph in which each node $u \in V$ has an associated *price* p_u which is a positive integer. Define the array cost as follows: for each $u \in V$,

$$\text{cost}[u] = \text{price of the cheapest node reachable from } u \text{ (including } u \text{ itself).}$$

Your goad is to design an algorithm that fills the entire cost array (i.e., for all vertices).

1. Give a linear-time algorithm that works for directed *acyclic* graphs. (Hint: Handle the vertices in a particular order)
2. Extend this to a linear-time algorithm that works for all directed graphs. (Hint: Recall the "two-tiered" structure of directed graphs)

Problem 10: Hamiltonian path (DPV)

Give a linear-time algorithm for the following task.

Input: A directed acyclic graph G

Question: Does G contain a directed path that touches every vertex exactly once?

Problem 11: Find a source (DPV)

Give an efficient algorithm which takes as input a directed graph $G = (V, E)$, and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable.

Problem 12: Number of semesters (DPV)

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 w if and only if v is a prerequisite for w. 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.

3 Breadth-First Search

Problem 13: Bipartite graphs (DPV)

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).

1. Give a linear-time algorithm to determine whether an undirected graph is bipartite.
2. 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.
3. At most how many colors are needed to color in an undirected graph with exactly *one* odd-length cycle?

Problem 14: Bipartite maximum matching

Consider bipartite graphs of the following special form:

- the vertex set is $\{x_1, ..x_n\}$, $\{y_1, ...y_n\}$.
- if (x_i, y_j) is an edge, there are no edges of the form (x_u, y_v) with both $u > i$ and $v < j$.

A *matching* is a set of edges, no two of which are incident on the same vertex. Design an algorithm that finds a maximum cardinality matching in linear time.

Problem 15: Bipartite graph

Prove that a graph is bipartite if and only if it has no odd cycles.

4 Spanning Trees

Problem 16: Divide-and-conquer spanning tree (CLRS)

Professor Toole proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

Problem 17: Alternative minimum-spanning-tree algorithms (CLRS)

In this problem, we give pseudocode for three different algorithms. Each one takes a graph as input and returns a set of edges T. For each algorithm, you must either prove that T is a minimum spanning tree or prove that T is not a minimum spanning tree. Also describe the most efficient implementation of each algorithm, whether or not it computes a minimum spanning tree.

1. MAYBE-MST-A (G, w)

```
1   sort the edges into nonincreasing order of edge weights w
2   T ← E
3   for each edge e, taken in nonincreasing order by weight
4       do if T − {e} is a connected graph
5             then T ← T − e
6   return T
```

2. MAYBE-MST-B (G, w)

```
1   T ← ∅
2   for each edge e, taken in arbitrary order
3       do if T ∪ {e} has no cycles
4             then T ← T ∪ e
5   return T
```

3. MAYBE-MST-C (G, w)

```
1   T ← ∅
2   for each edge e, taken in arbitrary order
3       do T ← T ∪ {e}
4          if T has a cycle
5             then let e′ be the maximum-weight edge on c
6                  T ← T − {e′}
7   return T
```

Problem 18: Kruskal vs Prim (CLRS)

Suppose that the edge weights in a graph are uniformly distributed over the half-open interval $[0, 1)$. Which algorithm, Kruskal's or Prim's, can you make run faster?

Problem 19: Incremental spanning tree (CLRS)

Suppose that a Graph G has a minimum spanning tree already computed. How quickly can the minimum spanning tree be updated if a new vertex and incident edges are added to G.

Problem 20: Spanning tree with decreased weights (CLRS)

Given a graph G and a minimum spanning tree T, suppose that we decrease the weight of one of the edges not in T. Give an algorithm for finding the minimum spanning tree in the modified graph.

Problem 21: Second-best minimum spanning tree (CLRS)

Let $G = (V, E)$ be an undirected, connected graph with weight function $w : E \to \mathbb{R}$, and suppose that $|E| \geq |V|$ and all edge weights are distinct.

A second-best minimum spanning tree is defined as follows. Let $\mathcal{T}$ be the set of all spanning trees of G, and let T' be a minimum spanning tree of G. Then a *second-best minimum spanning tree* is a spanning tree T such that $w(T) = \min_{T'' \in \mathcal{T} - \{T'\}} \{w(T'')\}$.

1. Show that the minimum spanning tree is unique, but that the second-best minimum spanning tree need not to be unique.

2. Let T be a minimum spanning tree of G. Prove that there exist edges $(u, v) \in T$ and $(x, y) \notin T$ such that $T - \{(u, v)\} \cup \{(x, y)\}$ is a second-best minimum spanning tree of G.

3. Let T be a spanning tree of G and, for any two vertices $u, v \in V$, let $\max[u, v]$ be an edge of maximum weight on the unique path between u and v in T. Describe an $O(V^2)$-time algorithm that, given T, computes $\max[u, v]$ for all $u, v \in V$.

4. Give an efficient algorithm to compute the second-best minimum spanning tree of G.

Problem 22: Minimum product spanning tree

Let G be a connected graph whose edges have positive integral weights. A *minimum product spanning tree* is a spanning tree in G, for which the *product* of the edge weights in minimized.
Give a polynomial-time algorithm that finds a minimum product spanning trees in a given connected, weighted graph. Hint: Think logarithms, but you cannot actually use them in the final algorithm, since computers cannot exactly manipulate irrational numbers like logarithms.

Problem 23: More on Minimum Spanning Trees

The minimum spanning tree problem takes as input a connected undirected graph on n nodes whose edges have integer weights. Possible solutions are spanning trees of the graph, *i.e.*, subsets of the edges that contain no cycles but connect all n nodes. The cost of a spanning tree is the sum of the weights of its edges, and the problem is to find a spanning tree of minimal cost.

Kruskal's algorithms is a greedy algorithm for this problem where, in every phase of the algorithm, a sub-forest of the tree has been chosen. In each phase, the minimum cost edge that connects two distinct components of this forest is added to the forest.

Prim's algorithm is also greedy but is slight different. Here, at the beginning of a phase, a tree has been found that spans a subset C of the nodes. The minimum cost edge between a node in C and a node not in C is added to the tree, and the endpoint of that edge not already in is added to C.

1. You are told that the inputs for your algorithm will all be planar graphs, and you know that planar graphs have at most $3n$ edges. Which algorithm do you pick, and which data structures do you choose to implement it? What is the overall time analysis?

2. Say that you know, in addition, that all the edge weights of the graphs are integers between 1 and n. Does this change which algorithm you choose, how you would implement the algorithm, or the time analysis?

3. Say that instead you want to find the *maximum* const spanning tree. Can you use modified versions of these algorithms? Why or why not?

5 Dijkstra's Algorithm

Problem 24: Lake's algorithm (DPV)

Professor F. Lake suggests the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges: add a large constant to each edge weight so that all the weights become positive, then run Dijkstra's algorithm starting at node s, and return the shortest path found to node t.

Is this a valid method? Either prove that it works correctly, or give a counterexample.

Problem 25: Small tank (DPV)

You are given a set of cities, along with the pattern of highways between them, in the form of an undirected graph $G = (V, E)$. Each stretch of highway $e \in E$ connects two of the cities, and you know its length in miles, l_e. You want to get from city s to city t. There's one problem: your car can only hold enough gas to cover L miles. There are gas stations in each city, but not between cities. Therefore, you can only take a route if every one of its edges has length $l_e \leq L$.

1. Given the limitations on your car's fuel tank capacity, show how to determine in linear time whether there is a feasible route from s to t.

2. You are now planning to buy a new car, and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give an $O((|V| + |E|) \log |V|)$ algorithm to determine this.

Problem 26: Negative weights at the source (DPV)

Consider a directed graph in which the only negative edges are those that leave s; all other edges are positive. Can Dijkstra's algorithm, started at s, fail on such a graph? Prove your answer.

Problem 27: Dijkstra with negative weights (CLRS)

Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

Problem 28: Dijkstra with small weights (CLRS)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(WV + E)$ time.

6 Bellman-Ford Algorithm

Problem 29: Yen's improvement to Bellman Ford (CLRS)

Suppose that we order the edge relaxations in each pass of the Bellman-Ford algorithm as follows. Before the first pass, we assign an arbitrary linear order $v_1, v_2, \ldots, v_{|V|}$ to the vertices of the input graph $G = (V, E)$. Then, we partition the edge set E into $E_f \cup E_b$, where $E_f = \{(v_i, v_j) \in E : i < j\}$ and $E_b = \{(v_i, v_j) \in E : i > j\}$. (Assume that G contains no self-loops, so that every edge is in either E_f or E_b.) Define $G_f = (V, E_f)$ and $G_b = (V, E_b)$.

1. Prove that G_f is acyclic with topological sort $\langle v_1, v_2, \ldots, v_{|V|} \rangle$ and that G_b is acyclic with topological sort $\langle v_{|V|}, v_{|V|-1}, \ldots, v_1 \rangle$.

Suppose that we implement each pass of the Bellman-Ford algorithm in the following way. We visit each vertex in the order $v_1, v_2, \ldots, v_{|V|}$, relaxing edges of E_f that leave the vertex. We then visit each vertex in the order $v_{|V|}, v_{|V|-1}, \ldots, v_1$, relaxing edges of E_b that leave the vertex.

2. Prove that with this scheme, if G contains no negative-weight cycles that are reachable from the source vertex s, then after only $\lceil |V|/2 \rceil$ passes over the edges, $d[v] = \delta(s, v)$ for all vertices $v \in V$.

3. Does this scheme improve the asymptotic running time of the Bellman-Ford algorithm?

Problem 30: Optimize Bellman-Ford (CLRS)

Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all pairs of vertices $u, v \in V$ of the minimum number of edges in a shortest path from u to v. (Here, the shortest path is by weight, not the number of edges). Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes.

Problem 31: Currencies (DPV)

Shortest path algorithms can be applied in currency trading. Let $c_1, c_2, \ldots, c_n$ be various currencies; for

instance, c_1 might be dollars, c_2 pounds, and c_3 lire. For any two currencies c_i and c_j, there is an exchange rate $r_{i,j}$; this means that you can purchase $r_{i,j}$ units of currency c_j in exchange for one unit of c_i. These exchange rates satisfy the condition that $r_{i,j} \cdot r_{j,i} < 1$, so that if you start with a unit of currency c_i, change it into currency c_j and then convert back to currency c_i, you end up with less than one unit of currency c_i (the difference is the cost of the transaction).

1. Give an efficient algorithm for the following problem: Given a set of exchange rates $r_{i,j}$, and two currencies s and t, find the most advantageous sequence of currency exchanges for converting currency s into currency t. Toward this goal, you should represent the currencies and rates by a graph whose edge lengths are real numbers.

The exchange rates are updated frequently, reflecting the demand and supply of the various currencies. Occasionally the exchange rates satisfy the following property: there is a sequence of currencies $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdots r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$. This means that by starting with a unit of currency c_{i_1}, you would end up with more than one unit of currency c_{i_1}. Such anomalies last only a fraction of a minute on the currency exchange, but they provide an opportunity for risk-free profits.

2. Give an efficient algorithm for detecting the presence of such an anomaly. Use the graph representation you found above.

Problem 32: Equality constraints (CLRS)

Suppose that in addition to a system of difference constraints, we want to handle *equality constraints* of the form $x_i = x_j + b_k$. Show how the Bellman-Ford algorithm can be adapted to solve this variety of constraint system.

7 Reachability

8 Shortest Paths

Problem 33: Cheapest path with edge and vertex weights

In certain graph problems, vertices can have weights instead of or in addition to the weights of edges. Let C_v be the cost of vertex v, and $C_{(x,y)}$ be the cost of the edge (x, y). This problem is concerned with finding the cheapest path between vertices a and b in a graph G. The cost of a path is the sum of the costs of the edges and vertices encountered on the path. Suppose that each edge in the graph has a weight of zero (while non-edges have a cost of ∞). Assume that $C_v = 1$ for all vertices (i.e. all vertices have the same cost). Give an efficient algorithm to find the cheapest path from a to b and its time complexity. Now suppose that the vertex costs are not all the same (but are all positive) and the edge costs remain as above. Give an efficient algorithm to find the cheapest path from a to b and its time complexity.

Solution: Cheapest path with edge and vertex weights

Solution for $C_v = 1$:

When the cost of each vertex is equal, the goal is to minimize the number of vertices traversed when going from a to b. To find the path with the fewest number of vertices, we can use a slight variation on breadth first search. The search will start at a and will stop when the vertex b has been reached.
Inputs: adjacency list representation of graph $G = (V, E)$, vertex to start at $a \in V$ and vertex to end at $b \in V$
Outputs: the cheapest path from a to b, $(a, v_1, v_2, ..., b)$

FindCheapestPath(G,a,b)

1. for $v \in V$
2. $\quad state[v]$ = "undiscovered"; $p[v] = nil$
3. $state[a]$ = "discovered"
4. $Q = \{\}$ // initialize queue
5. $u = a$
6. while $(Q \neq \emptyset)$ and $(u \neq b)$ do
7. $\quad$ for $v \in Adj[u]$ do
8. $\quad\quad$ if $state[v]$ = "undiscovered" then
9. $\quad\quad\quad state[v]$ = "discovered"; $p[v] = u$
10. $\quad\quad\quad$ enqueue$[Q, v]$
11. $\quad state[u]$ = "completely-explored"
12. $\quad u =$ dequeue$[Q]$
13. // generate path from a to b
14. $v = b$; $path = ()$
15. while $p[v] \neq a$
16. $\quad path = p[v]$ concat $path$
17. $path = a$ concat $path$
18. return $path$

This algorithm has worst-case running time $O(m + n)$, where m is the number of edges and n is the number of vertices.

Solution for $C_v > 0$:

Breadth-first search does not work when the cost of the vertices differ. The path with the least cost might go through more vertices. It is possible to use Dijkstra's algorithm with some slight modifications.
Inputs: adjacency list representation of graph $G = (V, E)$, vertex to start at $a \in V$ and vertex to end at $b \in V$, cost C_v for all $v \in V$
Outputs: the cheapest path from a to b, $(a, v_1, v_2, ..., b)$

FindCheapestPath(G,a,b)

1. for $v \in V$
2. $\quad pathcost[v] = \infty$; $p[v] = nil$
3. $pathcost[a] = C_a$
4. for $v \in V$ // initialize priority queue
5. $\quad$ Insert v into PQ with priority = $pathcost[v]$
6. v_{next} = ExtractMin(PQ)
7. $known = \emptyset$
8. while $known \neq V$
9. $\quad known = known \bigcup \{v_{next}\}$

10. for $v \in Adj(v_{next})$ do

11. $newcost = pathcost[v_{next}] + C_v$

12. if $pathcost[v] > newcost$ then

13. $pathcost[v] = newcost; p[v] = v_{next}$

14. $v_{next} = \text{ExtractMin}(PQ)$

15. // generate path from a to b

16. $v = b$; $path = ()$

17. while $p[v] \neq a$

18. $path = p[v]$ concat $path$

19. $path = a$ concat $path$

20. return $path$

The while loop at line 8 executes n times, where n is the number of vertices in the graph. Each time the while loop executes line 14 removes the vertex with the cheapest path cost. The extraction from a priority queue implemented as a binary heap takes $O(\lg n)$. Therefore, the worst-case running time of this algorithm is $O(n \lg n)$.

Problem 34: Distance

Give an algorithm running in time $O(m + n)$ for finding the distance from s to every other vertex in an undirected graph, all of whose edges have weight 1 or 2.

Problem 35: Bitonic shortest paths (CLRS)

A sequence is *bitonic* if it monotonically increases and then monotonically decreases, or if it can be circularly shifted to monotonically increase and then monotonically decrease. For example, the sequences $\langle 1, 4, 6, 8, 3, -2\rangle$, $\langle 9, 2, -4, -10, -5\rangle$, and $\langle 1, 2, 3, 4\rangle$ are bitonic, but $\langle 1, 3, 12, 4, 2, 10\rangle$ is not bitonic.

Suppose that we are given a directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, and we wish to find single-source shortest paths from a source vertex s. We are given one additional piece of information: for each vertex $v \in V$, the weights of the edges along any shortest path from s to v form a bitonic sequence.

Give the most efficient algorithm you can to solve this problem, and analyze its running time.

Problem 36: Bottleneck shortest s-t path

If P is a path in a weighted graph G, let $maxweight(P)$ be the maximum of all the weights of the edges in P.

Give a polynomial-time algorithm to solve the following problem:

Given an undirected graph whose edges have positive integral weights, and two distinct vertices s and t, among all $s - t$ paths P find one for which $maxweight(P)$ is minimized.

Problem 37: Unique paths (DPV)

Shortest paths are not always unique: sometimes there are two or more different paths with the minimum possible length. Show how to solve the following problem in $O((|V| + |E|) \log |V|)$ time.

Input: An undirected graph $G = (V, E)$; edge lengths $l_e > 0$; starting vertex $s \in V$.

Output: A Boolean array usp[·]: for each node u the entry usp[u] should be true if and only if there is a *unique* shortest path from s to u. (Note: usp[s] = true)

Problem 38: Short paths (DPV)

Your are given a directed graph with (possibly negative) weighted edges, in which the shortest path between any two vertices guaranteed to have at most k edges. Give an algorithm that finds the shortest path between two vertices u and v in $O(k|E|)$ time.

Problem 39: Special cases of shortest paths

You need to solve all pairs longest paths (*i.e.*, total delays) for a VLSI application. The input is a directed-acyclic graph where each node has in-degree at most 2. The edge lengths are all either 1 or 2. What algorithm and which data structures would you use? What is the time complexity of the algorithm for these instances?

Problem 40: Shortest path through node (DPV)

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding shortest paths between *all pairs of nodes*, with the one restriction that these paths must all pass through v_0.

Problem 41: Number of shortest paths (DPV)

Often there are multiple shortest paths between two nodes of a graph. Give a linear-time algorithm for the following task.

Input: Undirected graph $G = (V, E)$ with unit edge lengths; nodes $u, v \in V$.

Output: The number of distinct shortest paths from u to v.

Problem 42: Building a new road (DPV)

There is a network of roads $G = (V, E)$ connecting a set of cities V. Each road in E has an associated length l_e. There is a proposal to add one new road to this network, and there is a list E' of pairs of cities between which the new road can be built. Each such potential road $e' \in E'$ has an associated length. As a designer for the public works department you are asked to determine the road $e' \in E'$ whose addition to the existing network G would result in the maximum decrease in the driving distance between two fixed cities s and t in the network. Give an efficient algorithm for solving this problem.

Problem 43: Changing road conditions (KT 4.18)

Your friends are planning an expedition to a small town deep in the Canadian north next winter break. They've researched all the travel options and have drawn up a directed graph whose nodes represent intermediate destinations and edges represent the roads between them.

In the course of this, they've also learned that extreme weather causes roads in this part of the world to become quite slow in the winter and may cause large travel delays. They've found an excellent travel Web site that can accurately predict how fast they'll be able to travel along the roads; however, the speed of travel depends on the time of year. More precisely, the Web site answers queries of the following form: given an edge $e = (v, w)$ connecting two sites v and w, and given a proposed starting time t from location v, the site will return a value $f_e(t)$, the predicted arrival time at w. The Web site guarantees that $f_e(t) \geq t$ for all edges e and all times t (you can't travel backward in time), and that $f_e(t)$ is a monotone increasing function of t (that is, you do not arrive earlier by starting later). Other than that, the functions $f_e(t)$ may be arbitrary. For example, in areas where the travel time does not vary with the season, we would have $f_e(t) = t + \ell_e$, where ℓ_e is the time needed to travel from the beginning to the end of edge e.

Your friends want to use the Web site to determine the fastest way to travel through the directed graph from their starting point to their intended destination. (You should assume that they start at time 0, and that all predictions made by the Web site are completely correct.) Give a polynomial-time algorithm to do this, where we treat a single query to the Web site (based on a specific edge e and a time t) as taking a single computational step.

Problem 44: Closest neighbor (CLRS)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$. Give an $O(VE)$-time algorithm to find, for each vertex $v \in V$, the value $\delta^*(v) = \min_{u \in V}\{\delta(u, v)\}$.

Problem 45: Number of hops (DPV)

In cases where there are several different shortest paths between two nodes (and edges have varying lengths), the most convenient of these paths is often *the one with fewest edges*. For instance, if nodes represent cities and edge lengths represent costs of flying between cities, there might be many ways to get from city s to city t which all have the same cost. The most convenient of these alternatives is the one which involves the fewest stopovers. Accordingly, for a specific starting node s, define

$$\text{best}[u] = \text{minimum number of edges in a shortest path from } s \text{ to } u$$

Give an efficient algorithm for the following problem.

Input: Graph $G = (V, E)$; positive edge lengths l_e; starting node $s \in V$

Output: The values of best$[u]$ should be set for *all* nodes $u \in V$

9 Cycles

Problem 46: Cycle containing particular edge (DPV)

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

Problem 47: Cycle detection (CLRS)

Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of $|E|$.

Problem 48: Negative cycles and Floyd-Warshall (CLRS)

How can the output of the Floyd-Warshall algorithm be used to detect the presence of a negative-weight cycle?

Problem 49: Transitive closure (CLRS)

Give an $O(VE)$-time algorithm for computing the transitive closure of a directed graph $G = (V, E)$.

Problem 50: Find negative-weight cycle (CLRS)

Suppose that a weighted, directed graph $G = (V, E)$ has a negative-weight cycle. Give an efficient algorithm to list the vertices of one such cycle. Prove that your algorithm is correct.

Problem 51: Arbitrage (CLRS)

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 46.4 Indian rupees, 1 Indian rupee buys 2.5 Japanese yen, and 1 Japanese yen buys 0.0091 U.S. dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buy $46.4 \times 2.5 \times 0.0091 = 1.0556$ U.S. dollars, thus turning a profit of 5.56 percent.

Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j.

1. Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ such that
$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$
Analyze the running time of your algorithm

2. Give an efficient algorithm to print out such a sequence if one exists. Analyze the running time of your algorithm.

Problem 52: The tramp steamer problem (DPV)

You are the owner of a steamship that can ply between a group of port cities V. You make money at each port: a visit to city i earns you a profit of p_i dollars. Meanwhile, the transportation cost from port i to port j is $c_{ij} > 1$. You want to find a cyclic route in which the ratio of profit to cost is maximized.

To this end, consider a directed graph $G = (V, E)$ whose nodes are ports, and which has edges between each pair of ports. For any cycle C in this graph, the profit-to-cost ratio is
$$r(C) = \frac{\sum_{(i,j) \in C} p_j}{\sum_{(i,j) \in C} c_{ij}}$$
Let r^* be the maximum ratio achievable by a simple cycle. One way to determine r^* is by binary search: by first guessing some ratio r, and then testing whether it is too large or too small.

Consider any positive $r > 0$. Give each edge (i, j) a weight of $w_{ij} = rc_{ij} - p_j$.

1. Show that if there is a cycle of negative weight, then $r < r^*$.

2. Show that if all cycles in the graph have strictly positive weight, then $r > r^*$.

3. Give an efficient algorithm that takes as input a desired accuracy $\epsilon > 0$ and returns a simple cycle C for which $r(C) \geq r^* - \epsilon$. Justify the correctness of your algorithm and analyze its running time in terms of $|V|$, ϵ, and $R = \max_{(i,j) \in E}(p_j/c_{ij})$.

Problem 53: Finding the shortest cycle (DPV)

Here's a proposal for how to find the length of the shortest cycle in an undirected graph with unit edge lengths.

When a back edge, say (v, w), is encountered during a depth-first search, it form a cycle with the tree edges from w to v. The length of the cycle is $\text{level}[v] - \text{level}[w] + 1$, where the level of a vertex is its distance in the DFS tree from the root vertex. This suggests the following algorithm:

- Do a depth-first search, keeping track of the level of each vertex.
- Each time a back edge is encountered, compute the cycle length and save it if it is smaller than the shortest one previously seen.

Show that this strategy does not always work by providing a counterexample as well as a brief (one or two sentence) explanation.

Problem 54: Shortest cycle (DPV)

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 so). Your algorithm should take time at most $O(|V|^3)$.

Problem 55: Shortest cycle containing an edge (DPV)

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.

Problem 56: Karp's minimum mean-weight cycle algorithm (CLRS)

Let $G = (V, E)$ be a directed graph with weight function $w : E \to \mathbb{R}$, and let $n = |V|$. We define the *mean weight* of a cycle $c = \langle e_1, e_2, \ldots, e_k \rangle$ of edges in E to be

$$\mu(c) = \frac{1}{k} \sum_{i=1}^{k} w(e_i)$$

Let $\mu^* = \min_c \mu(c)$, where c ranges over all directed cycles in G. A cycle c for which $\mu(c) = \mu^*$ is called a *minimum mean-weight cycle*. This problem investigates an efficient algorithm for computing μ^*.

Assume without loss of generality that every vertex $v \in V$ is reachable from a source vertex $s \in V$. Let $\delta_(s, v)$ be the weight of a shortest path from s to v, and let $\delta_k(s, v)$ be the weight of a shortest path from s to v consisting of *exactly* k edges. If there is no path from s to v with exactly k edges, then $\delta_k(s, v) = \infty$.

1. Show that if $\mu^* = 0$, then G contains no negative-weight cycles and $\delta(s, v) = \min_{0 \le k \le n-1} \delta_k(s, v)$ for all vertices $v \in V$.

2. Show that if $\mu^* = 0$, then
$$\max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} \ge 0$$
for all vertices $v \in V$. (Hint: Use both properties from part 1)

3. Let c be a 0-weight cycle, and let u and v be any two vertices on c. Suppose that $\mu^* = 0$ and that the weight of the path from u to v along the cycle is x. Prove that $\delta(s, v) = \delta(s, u) + x$. (Hint: The weight of the path from v to u along the cycle is $-x$.)

4. Show that if $\mu^* = 0$, then on each minimum mean-weight cycle there exists a vertex v such that
$$\max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} = 0$$
(Hint: Show that a shortest path to any vertex on a minimum mean-weight cycle can be extended along the cycle to make a shortest path to the next vertex on the cycle.)

5. Show that if $mu^* = 0$, then
$$\min_{v \in V} \max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} = 0$$

6. Show that if we add a constant t to the weight of each edge of G, then μ^* is increased by t. Use this fact to show
$$\mu^* = \min_{v \in V} \max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k}$$

7. Give an $O(VE)$-time algorithm to compute μ^*.

10 Transitive Closure

Problem 57: Transitive closure of a dynamic path (CLRS)

Suppose that we wish to maintain the transitive closure of a directed graph $G = (V, E)$ as we insert edges into E. That is, after each edge has been inserted, we want to update the transitive closure of the edges inserted so far. Assume that the graph G has no edges initially and that the transitive closure is to be represented as a boolean matrix.

1. Show how the transitive closure $G^* = (V, E^*)$ of a graph $G = (V, E)$ can be updated in $O(V^2)$ time when a new edge is added to G.
2. Give an example of a graph G and an edge e such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G.
3. Describe an efficient algorithm for updating the transitive closure as edges are inserted into the graph. For any sequence of n insertions, your algorithm should run in total time $\sum_{i=1}^{n} t_i = O(V^3)$, where t_i is the time to update the transitive closure when the ith edge is inserted. Prove that your algorithm attains this time bound.

11 Graph Theory

Problem 58: Degree of a graph (DPV)

In an undirected graph, the *degree* $d(u)$ of a vertex u is the number of neighbors u has, or equivalently, the number of edges incident upon it. In a directed graph, we distinguish between the *indegree* $d_{in}(u)$, which is the number of edges into u, and the *outdegree* $d_{out}(u)$, the number of edges leaving u.

1. Show that in an undirected graph, $\sum_{u \in V} d(u) = 2|E|$.
2. Use part 1 to show that in an undirected graph, there must be an even number of vertices whose degree is odd.
3. Does a similar statement hold for the number of vertices with odd indegree in a directed graph?

Problem 59: Number of connected components (DPV)

Show that if an undirected graph with n vertices has k connected components, then it has at least $n - k$ edges.

12 Miscellaneous

Problem 60: Degree of neighbors (DPV)

For each node u in an undirected graph, let twodegree[u] be the sum of the degrees of u's neighbors. Show how to compute the entire array of twodegree[$\cdot$] values in linear time, given a graph in adjacency list format.

Problem 61: Directed vs undirected connectivity (DPV)

1. Prove that in any connected undirected graph $G = (V, E)$ there is a vertex $v \in V$ whose removal leaves G connected. (Hint: Consider the DFS search tree for G.)

2. Give an example of a strongly connected directed graph $G = (V, E)$ such that, for every $v \in V$, removing v form G leaves a directed graph that is not strongly connected.

3. In an undirected graph with 2 connected components it is always possible to make the graph connected by adding only one edge. Give an example of a directed graph with two strongly connected components such that no addition of one edge can make the graph strongly connected.

Problem 62: Edge-disjointness (DPV)

Two paths in a graph are called *edge-disjoint* if they have no edges in common. Show that in any undirected graph, it is possible to pair up the vertices of odd degree and find paths between each such pair so that all these paths are edge disjoint.

Problem 63: Escape the maze (CLRS)

Let $G = (V, E)$ be a connected, undirected graph. Give an $O(V + E)$-time algorithm to compute a path in G that traverses each edge in E exactly once in each direction. Describe how you can find your way out of a maze if you are given a large supply of pennies.

Problem 64: Euler tour (CLRS)

An *Euler tour* of a connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once.

1. Show that G has an Euler tour if and only if in-degree(v) = out-degree(v) for each vertex $v \in V$.

2. Describe an $O(E)$-time algorithm to find an Euler tour of G if one exists (Hint: Merge edge-disjoint cycles)

Problem 65: Fuzzy Connectedness

A fuzzy undirected graph is an undirected graph with edge weights $0 \leq w_e \leq 1$. The fuzzy connectedness of a path is the minimum weight of an edge on the path. The fuzzy connectedness of two nodes is the maximum fuzzy connectedness of a path between them. Give the best algorithm you can to compute the fuzzy connectedness between two nodes in a fuzzy undirected graph.

Problem 66: Maximum matching

A *matching* is a collection of edges such that no two distinct edges have an end point in common.

A *maximum matching* is one of maximum cardinality. Prove that the following algorithm correctly finds a maximum matching in a forest F:

If F has no edges, output the empty set and halt. Otherwise, let v be a leaf in F, let u be its neighbor and let $e = \{u, v\}$. Let F' be the forest obtained by deleting both v and u (and all incident edges) from F. Recursively find a maximum matching M' in F', and output $M' \cup \{e\}$.

Problem 67: Number of different paths (DPV)

Give an efficient algorithm that takes as input a directed acyclic graph $G = (V, E)$, and two vertices $s, t \in V$, and outputs the number of different directed paths from s to t in G.

Problem 68: Odd-length cycle detection (DPV)

Give a linear-time algorithm to find an odd-length cycle in a *directed* graph. (Hint: First solve this problem under the assumption that the graph is strong connected.)

Problem 69: One-way streets in Computopia (DPV)

The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, sand there is just enough time to run a linear-time algorithm.

1. Formulate this problem graph-theoretically, and explain why it can indeed by solved in linear time.
2. Suppose it now turns out that the mayor's original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the town hall. Formulate this weaker property as a graph theoretic problem, and carefully show how it too can be checked in linear time.

Problem 70: One-Way streets

A city currently has only two-way streets in its downtown. It wants to make all of the streets one-way, while keeping any location reachable from any other location. Formally, an instance of the problem is an undirected graph. A solution is a directed graph where each edge in the undirected graph appears in exactly one direction. The constraint is that the solution must be strongly connected. Give the most efficient algorithm you can for the problem of finding a solution that is strongly connected or showing that it is impossible.

Problem 71: Reachability (CLRS)

Let $G = (V, E)$ be a directed graph in which each vertex $u \in V$ is labeled with a unique integer $L(u)$ from the set $\{1, 2, \ldots, |V|\}$. For each vertex $u \in V$, let $R(u) = \{v \in V : u \rightsquigarrow v\}$ be the set of vertices that are reachable from u. Define $\min(u)$ to be the vertex in $R(u)$ whose label is minimum, i.e., $\min(u)$ is the vertex v such that $L(v) = \min\{L(w) : w \in R(u)\}$. Give an $O(V + E)$-time algorithm that computes $\min(u)$ for all vertices $u \in V$.

Problem 72: Singly connected graph (CLRS)

A directed graph $G = (V, E)$ is *singly connected* if $u \rightsquigarrow v$ implies that there is at most one simple path from u to v for all vertices $u, v \in V$. Give an efficient algorithm to determine whether or not a directed graph is singly connected

Problem 73: Edge-disjoint paths

1. Given an undirected graph $G = V, E$ and two distinguished nodes, s and t, describe an algorithm that finds a maximum-sized set of edge-disjoint paths from s to t. (We say that two paths are $edge-disjoint$ if they don't share a common edge, even though they are allowed to go through the same vertex.) Give a time analysis for your algorithm in terms of $|V|$ and $|E|$.
2. Given a solution to (a), suppose we add one more edge to the graph. Give an efficient algorithm for updating the solution. Give a time analysis for your algorithm in terms of $|V|$ and $|E|$. (Your algorithm should be significantly faster than redoing the entire problem from scratch.)

Problem 74: Nesting Boxes (CLRS)

A d-dimensional box with dimensions $(x_1, x_2, \ldots, x_d)$ *nests* within another box with dimensions $(y_1, y_2, \ldots, y_d)$ if there exists a permutation π on $\{1, 2, \ldots, d\}$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(d)} < y_d$.

1. Argue that the nesting relation is transitive.

2. Describe an efficient method to determine whether or not one d-dimensional box nests inside another.

3. Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Describe an efficient algorithm to determine the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1 \rangle$. Express the running time of your algorithm in terms of n and d.