Design and Analysis of Algorithms
Session: Jan-May, 2020 (Course ID: CS2800)
Assignment 4: Greedy Approach

1. You wish to drive from point A to point B along a highway minimizing the time that you are stopped for gas. You are told beforehand the capacity C of you gas tank in liters, your rate F of fuel consumption in liters/kilometer, the rate r in liters/minute at which you can fill your tank at a gas station, and the locations $A = x_1, \cdots, B = x_n$ of the gas stations along the highway. So if you stop to fill your tank from 2 liters to 8 liters, you would have to stop for $6/r$ minutes. Consider the following two algorithms:

(a) Stop at every gas station, and fill the tank with just enough gas to make it to the next gas station.

(b) Stop if and only if you don't have enough gas to make it to the next gas station, and if you stop,fill the tank up all the way.

For each algorithm either prove or disprove that this algorithm correctly solves the problem. Your proof of correctness must use an exchange argument.

2. Consider the following problem. The input is a collection $A = \{a_1, \cdots, a_n\}$ of n points on the real line. The problem is to find a minimum cardinality collection S of unit intervals that cover every point in A. Another way to think about this same problem is the following. You know a collection of times (A) that trains will arrive at a station. When a train arrives there must be someone manning the station. Due to union rules, each employee can work at most one hour at the station. The problem is to find a scheduling of employees that covers all the times in A and uses the fewest number of employees.

(a) Prove or disprove that the following algorithm correctly solves this problem. Let I be the interval that covers the most number of points in A. Add I to the solution set S. Then recursively continue on the points in A not covered by I.

(b) Prove or disprove that the following algorithm correctly solves this problem. Let a_j be the smallest (leftmost) point in A. Add the interval $I = (a_j, a_j + 1)$ to the solution set S. Then recursively continue on the points in A not covered by I.

HINT: One of the above greedy algorithms is correct and one is incorrect for the other. The proof of correctness must use an exchange argument.

3. We consider a greedy algorithm for two related problems.

(a) The input to this problem consists of an ordered list of n words. The length of the ith word is w_i ,that is the ith word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line

is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty for having a line of length K is LK. *The total penalty is the* ***sum*** *of the line penalties.* The problem is to find a layout that minimizes the total penalty. Prove of disprove that the following greedy algorithm correctly solves this problem.

```
For i= 1 to n
      Place the ith word on the current line if it fits
           else place the ith word on a new line.
```

(b) The input to this problem consists of an ordered list of n words. The length of the ith word is w_i , that is the ith word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty for having a line of length K is LK. *total penalty is the* ***maximum*** *of the line penalties.* The problem is to find a layout that minimizes the total penalty. Prove of disprove that the following greedy algorithm correctly solves this problem.

```
For i= 1 to n
      Place the ith word on the current line if it fits
           else place the ith word on a new line
```

HINT: The greedy algorithm is correct for one of the above two problems and is incorrect for the other.The proof of correctness must be done using an exchange argument.

4. Consider the following problem. The input consists of n skiers with heights $p_1, \cdots, p_n$, and n skies with heights $s_1, \cdots, s_n$. The problem is to assign each skier a ski to to minimize the average difference between the height of a skier and his/her assigned ski. That is, if the ith skier is given the $\alpha(i)$th ski, then you want to minimize:

$$\frac{1}{n} \sum_{i=1}^{n} |p_i - s_{\alpha(i)}|$$

(a) Consider the following greedy algorithm. Find the skier and ski whose height difference is minimized. Assign this skier this ski. Repeat the process until every skier has a ski. Prove of disprove that this algorithm is correct.

(b) Consider the following greedy algorithm. Give the shortest skier the shortest ski, give the second shortest skier the second shortest ski, give the third shortest skier the third shortest ski, etc.Prove of disprove that this algorithm is correct.

HINT: One of the above greedy algorithms is correct and one is incorrect for the other. The proof of correctness must be done using an exchange argument.

5. We consider the following scheduling problem:

INPUT: A collection of jobs $J_1, \cdots, J_n$. The size of job J_i is x_i, which is a non-negative integer. An integer m.

OUTPUT: A nonpreemptive feasible schedule for these jobs on m processor that minimizes the total n completion time $\sum_{i=1}^{n} C_i$.

A *schedule* specifies for each unit time interval and for each processor, the unique job that that is run during that time interval on that processor. In a *feasible* schedule, every job J_i has to be run for exactly x_i time units after time 0. In a *nonpreemptive* schedule, once a job starts running on a particular processor, it has to be run to completion on that particular processor. The *completion time* C_i for job J_i is the earliest time when J_i has been run for x_i time units. So for example if $m = 2$ jobs of size $1, 4, 3$ are run in that order on the first processor, and jobs of size $7, 10$ are run on the second processor in that order, then the total completion time would be $1 + 5 + 8 + 7 + 17 = 38$.
Give a greedy algorithm for this problem and prove that it is correct.

6. Consider the following problem.

INPUT: Positive integers $r_1, \cdots, r_n$ and $c_1, \cdots, c_n$.

OUTPUT: An n by n matrix A with 0/1 entries such that for all i the sum of the ith row in A is r_i and the sum of the ith column in A is c_i , if such a matrix exists.

Think of the problem this way. You want to put pawns on an n by n chessboard so that the ith row has r_i pawns and the ith column has c_i pawns.

Consider the following greedy algorithm that constructs A row by row. Assume that the first i-1 rows have been constructed. Let a_j be the number of 1′s in the jth column in the first i-1 rows. Now the r_i columns with with maximum c_j-a_j are assigned 1's in row i, and the rest of the columns are assigned 0′s. That is, the columns that still needs the most 1's are given 1's. Formally prove that this algorithm is correct using an exchange argument.

7. Consider the following bridge crossing problem where n people with speeds $s_1, \cdots, s_n$ wish to cross the bridge as quickly as possible. The rules remain:

- It is nighttime and you only have one flashlight.
- A maximum of two people can cross at any one time
- Any party who crosses, either 1 or 2 people must have the flashlight with them.
- The flashlight must be walked back and forth, it cannot be thrown, etc.
- A pair must walk together at the rate of the slower person's pace.

Give an efficient algorithm to find the fastest way to get a group of people across the bridge. You **must** have a proof of correctness for your method.

8. The GreedySchedule algorithm we described for the class scheduling problem is not the only greedy strategy we could have tried. For each of the following alternative greedy strategies, either prove that the resulting algorithm always constructs an optimal schedule, or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control). [Hint: Three of these algorithms are actually correct.]

(a) Choose the course x that ends last, discard classes that conflict with x, and recurse.

(b) Choose the course x that starts first, discard all classes that conflict with x, and recurse.

(c) Choose the course x that starts last, discard all classes that conflict with x, and recurse.

(d) Choose the course x with shortest duration, discard all classes that conflict with x, and recurse.

(e) Choose a course x that conflicts with the fewest other courses, discard all classes that conflict with x, and recurse.

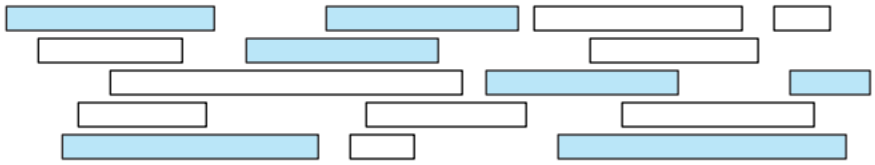
(f) If no classes conflict, choose them all. Otherwise, discard the course with longest duration and recurse.

(g) If no classes conflict, choose them all. Otherwise, discard a course that conflicts with the most other courses and recurse.

(h) Let x be the class with the earliest start time, and let y be the class with the second earliest start time.
- If x and y are disjoint, choose x and recurse on everything but x.
- If x completely contains y, discard x and recurse.
- Otherwise, discard y and recurse.

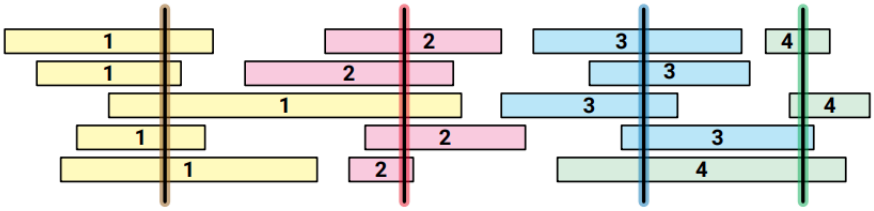
(i) If any course x completely contains another course, discard x and recurse. Otherwise, choose the course y that ends last, discard all classes that conflict with y, and recurse.

9. Let X be a set of n intervals on the real line. We say that a subset of intervals $Y \subseteq X$ covers X if the union of all intervals in Y is equal to the union of all intervals in X. The size of a cover is just the number of intervals. Describe and analyze an efficient algorithm to compute the smallest cover of X. Assume that your input consists of two arrays $L[1 \,..\, n]$ and $R[1 \,..\, n]$, representing the left and right endpoints of the intervals in X. If you use a greedy algorithm, you must prove that it is correct.



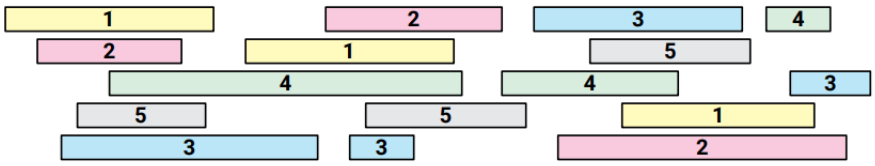
A set of intervals, with a cover (shaded) of size 7.

10. Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P. Describe and analyze an efficient algorithm to compute the smallest set of points that stabs X. Assume that your input consists of two arrays L[1 .. n] and R[1 .. n], representing the left and right endpoints of the intervals in X. As usual, If you use a greedy algorithm, you must prove that it is correct.



A set of intervals stabbed by four points (shown here as vertical segments)

11. Let X be a set of n intervals on the real line. A proper coloring of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color X. Assume that your input consists of two arrays L[1 .. n] and R[1 .. n], representing the left and right endpoints of the intervals in X. As usual, if you use a greedy algorithm, you must prove that it is correct.



A proper coloring of a set of intervals using five colors.

12. Huffman Coding:

(a) For every integer n, find a frequency array *f[1 .. n]* whose Huffman code tree has depth n-1, such that the largest frequency is as small as possible.

(b) Suppose the total length N of the unencoded message is bounded by a polynomial in the alphabet size n. Prove that the any Huffman tree for the frequencies *f[1 .. n]* has depth O(log n).

13. Call a frequency array f [1 .. n] α-heavy if it satisfies two conditions:
• f [1] > f [i] for all i > 1; that is, 1 is the unique most frequent symbol.
• f [1] $\geq \alpha \sum_{i=1}^{n} f[i]$; that is, at least an α fraction of the symbols are 1s.

Find the largest real number α such that in every Huffman code for every α-heavy frequency array, symbol 1 is represented by a single bit. [Hint: First prove that $1/3 \leq \alpha \leq 1/2$.]

14. You've been hired to store a sequence of n books on shelves in a library. The order of the books is fixed by the cataloging system and cannot be changed; each shelf must store a contiguous interval of the given sequence of books. You are given two arrays H[1 .. n] and T[1 .. n], where H[i] and T[i] are respectively the height and thickness of the ith book in the sequence. All shelves in this library have the same length L; the total thickness of all books on any single shelf cannot exceed L.

(a) Suppose all the books have the same height h and the shelves have height larger than h, so every book fits on every shelf. Describe and analyze a greedy algorithm to store the books in as few shelves as possible. [Hint: The algorithm is obvious, but why is it correct?]

(b) That was a nice warmup, but now here's the real problem. In fact the books have different heights, but you can adjust the height of each shelf to match the tallest book on that shelf. (In particular, you can change the height of any empty shelf to zero.) Now your task is to store the books so that the sum of the heights of the shelves is as small as possible. Show that your greedy algorithm from part (a) does not always give the best solution to this problem.

(c) Describe and analyze an algorithm to find the best matching between books and shelves as described in part (b).

15. A string w of parentheses (and) is balanced if it satisfies one of the following conditions:

- w is the empty string.
- w = (x) for some balanced string x
- w = xy for some balanced strings x and y

For example, the string w = (((())()())((()))() is balanced, because w = xy, where x = ((())()()) and y = (()())().

(a) Describe and analyze an algorithm to determine whether a given string of parentheses is balanced.

(b) Describe and analyze a greedy algorithm to compute the length of a longest balanced subsequence of a given string of parentheses. As usual, don't forget to prove your algorithm is correct. For both problems, your input is an array w[1 .. n], where for each i, either w[i] = (or w[i] =). Both of your algorithms should run in O(n) time.

16. One day Alex got tired of climbing in a gym and decided to take a large group of climber friends outside to climb. They went to a climbing area with a huge wide boulder, not very

tall, with several marked hand and foot holds. Alex quickly determined an "allowed" set of moves that her group of friends can perform to get from one hold to another.

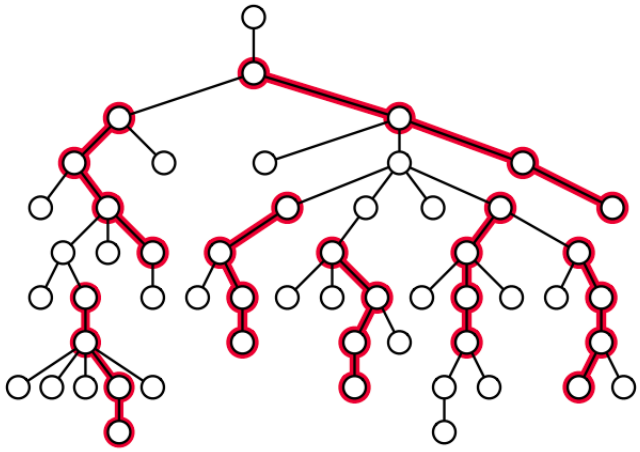
The overall system of holds can be described by a rooted tree T with n vertices, where each vertex corresponds to a hold and each edge corresponds to an allowed move between holds. The climbing paths converge as they go up the boulder, leading to a unique hold at the summit, represented by the root of T.

Alex and her friends (who are all excellent climbers) decided to play a game, where as many climbers as possible are simultaneously on the boulder and each climber needs to perform a sequence of exactly k moves. Each climber can choose an arbitrary hold to start from, and all moves must move away from the ground. Thus, each climber traces out a path of k edges in the tree T, all directed toward the root. However, no two climbers are allowed to touch the same hold; the paths followed by different climbers cannot intersect at all.

(a) Describe and analyze a greedy algorithm to compute the maximum number of climbers that can play this game. Your algorithm is given a rooted tree T and an integer k as input, and it should compute the largest possible number of disjoint paths in T, where each path has length k. Do not assume that T is a binary tree. For example, given the tree below as input, your algorithm should return the integer 8.

(b) Now suppose each vertex in T has an associated reward, and your goal is to maximize the total reward of the vertices in your paths, instead of the total number of paths. Show that your greedy algorithm does not always return the optimal reward. (

(c) Describe an efficient algorithm to compute the maximum possible reward, as described in part (b).



. Seven disjoint paths of length $k = 3$. This is *not* the largest such set of paths in this tree.

17. We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c\}$ with frequencies f_a, f_b, f_c. In each of the following cases, either give an example of frequencies (f_a, f_b, f_c) that

would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are).

(a) Code: {0, 10, 11}

(b) Code: {0, 1, 00}

(c) Code: {10, 01, 00}

18. Prove the following two properties of the Huffman encoding scheme.

(a) If some character occurs with frequency more than 2/5, then there is guaranteed to be a codeword of length 1.

(b) If all characters occur with frequency less than 1/3, then there is guaranteed to be no codeword of length 1.

19. Under a Huffman encoding of n symbols with frequencies $f_1, f_2, ..., f_n$, what is the longest a codeword could possibly be? Give an example set of frequencies that would produce this case.

20. A *feedback edge* set of an undirected graph G = (V, E) is a subset of edges E' $\subseteq$ E that intersects every cycle of the graph. Thus, removing the edges E' will render the graph acyclic. Give an efficient algorithm for the following problem:

Input: Undirected graph G = (V, E) with positive edge weights w_e.
Output: A feedback edge set E' $\subseteq$ E of minimum total weigth $\sum_{e \in E'} w_e$.

21. You are given a graph G = (V, E) with positive edge weights, and a minimum spanning tree T = (V, E') with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge e $\in$ E is modified from w(e) to a new value $\hat{w}$(e). You wish to quickly update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each case give a linear-time algorithm for updating the tree.

(a) e $\notin$ E' and $\hat{w}$(e) > w(e).

(b) e $\notin$ E' and $\hat{w}$(e) < w(e).

(c) e $\in$ E' and $\hat{w}$(e) < w(e).

(d) e $\in$ E' and $\hat{w}$(e) > w(e).

22. Graphs with prescribed degree sequences. Given a list of n positive integers $d_1, d_2, ..., d_n$, we want to efficiently determine whether there exists an undirected graph G = (V, E) whose nodes have degrees precisely $d_1, d_2, ..., d_n$. That is, if V = $\{v_1, ..., v_n\}$, then the degree of v_i should be exactly d_i. We call $(d_1, ..., d_n)$ the degree sequence of G. This graph G should not contain self-loops (edges with both endpoints equal to the same node) or multiple edges between the same pair of nodes.

(a) Give an example of $d_1, d_2, ..., d_n$ where all the $d_i \leq 3$ and $d_1 + d_2 + d_3 + d_4$ is even, but for which no graph with degree sequence (d_1, d_2, d_3, d_4) exists.

(b) Suppose that $d_1 \geq d_2 \geq ... \geq d_n$ and that there exists a graph G = (V, E) with degree sequence $(d_1, d_2, ..., d_n)$. We want to show that there must exist a graph that has this degree sequence and where in addition the neighbors of v_1 are $v_2, v_3, ..., v_{d_1+1}$. The idea is to gradually transform G into a graph with the desired additional property.

i. Suppose the neighbors of v_1 in G are not $v_2, v_3, ..., v_{d_1+1}$. Show that there exists i < j ≤ n and u ∈ V such that $\{v_1, v_i\}$, $\{u, v_j\} \notin$ E and $\{v_1, v_j\}$, $\{u, v_i\} \in$ E.

ii. Specify the changes you would make to G to obtain a new graph G' = (V, E') with the same degree sequence as G and where $(v_1, v_i) \in$ E'.

iii. Now show that there must be a graph with the given degree sequence but in which v_1 has neighbors $v_2, v_3, ..., v_{d_1+1}$.

(c) Using the result from part (b), describe an algorithm that on input $d_1, d_2, ..., d_n$ (not necessarily sorted) decides whether there exists a graph with this degree sequence. Your algorithm should run in time polynomial in n.

23. The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counter example (if it isn't correct). Always assume that the graph G = (V, E) is undirected. Do not assume that edge weights are distinct unless this is specifically stated.

(a) If graph G has more than $|V|$ -1 edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.

(b) If G has a cycle with a unique heaviest edge e, then e cannot be part of any MST.

(c) Let e be any edge of minimum weight in G. Then e must be part of some MST.

(d) If the lightest edge in a graph is unique, then it must be part of every MST.

(e) If e is part of some MST of G, then it must be a lightest edge across some cut of G.

(f) If G has a cycle with a unique lightest edge e, then e must be part of every MST.

(g) The shortest-path tree computed by Dijkstra's algorithm is necessarily an MST.

(h) The shortest path between two nodes is necessarily part of some MST.

(i) Prim's algorithm works correctly when there are negative edges.

(j) (For any r > 0, define an r-path to be a path whose edges all have weight < r.) If G contains an r-path from node s to t, then every MST of G must also contain an r-path from node s to node t.

24. 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 $O(|V|^3)$.

25. 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.

26. 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 $l_e \leq L$.

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

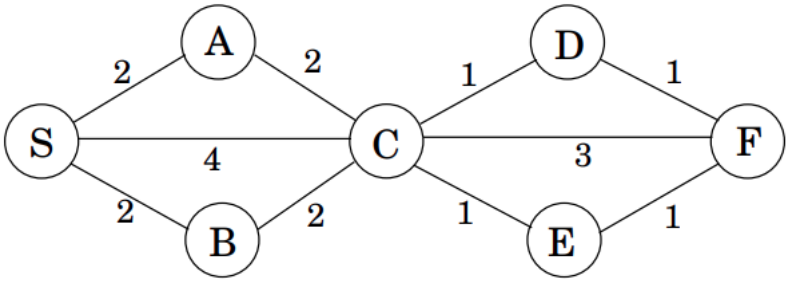
(b) You are now planning to buy a new car, and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give an O((|V| + |E|)log |V|) algorithm to determine this.

27. 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 $s \in$S .
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`.)

28. 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

`best[u]` = minimum number of edges in a shortest path from s to u.

In the example below, the best values for nodes S, A, B, C, D, E, F are 0, 1, 1, 1, 2, 2, 3, respectively.



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 $u|inV$.

29. **Generalized shortest-paths problem**. In Internet routing, there are delays on lines but also, more significantly, delays at routers. This motivates a generalized shortest-paths problem. Suppose that in addition to having edge lengths $\{l_e : e \in E\}$, a graph also has vertex costs $\{c_v : v \in V\}$. Now define the cost of a path to be the sum of its edge lengths, plus the costs of all vertices on the path (including the endpoints). Give an efficient algorithm for the following problem.
Input: A directed graph G = (V, E); positive edge lengths l_e and positive vertex costs c_v; a starting vertex $s \in V$.
Output: An array cost[·] such that for every vertex u, cost[u] is the least cost of any path from s to u (i.e., the cost of the cheapest path), under the definition above. Notice that $cost[s] = c_s$.

30. Which of the following claims are true and which are false? Justify your answer by giving a proof or by constructing a counterexample.

(a) If all arcs in a network have different costs, the network has a unique shortest path tree.

(b) In a directed network with positive arc lengths, if we eliminate the direction on every arc (i.e., make it undirected), the shortest path distances will not change.

(c) In a shortest path problem, if each arc length increases by k units, shortest path distances increase by a mUltiple of k.

(d) In a shortest path problem, if each arc length decreases by k units, shortest path distances decrease by a mUltiple of k.

(e) Among all shortest paths in a network, Dijkstra's algorithm always finds a shortest path with the least number of arcs.

31. **Most vital arc problem**. A vital arc of a network is an arc whose removal from the network causes the shortest distance between two specified nodes, say node s and node t, to increase. A most vital arc is a vital arc whose removal yields the greatest increase in the shortest distance from node s to node t. Assume that the network is directed, arc lengths

are positive, and some arc is vital. Prove that the following statements are true or show through counterexamples that they are false.

(a) A most vital arc is an arc with the maximum value of C_{ij}.

(b) A most vital arc is an arc with the maximum value of C_{ij} on some shortest path from node s to node t.

(c) An arc that does not belong to any shortest path from node s to node t cannot be a most vital arc.

(d) A network might contain several most vital arcs.

32. Describe an algorithm for determining a most vital arc in a directed network. What is the running time of your algorithm?

33. **Maximum capacity path problem**. Let $C_{ij} \geq 0$ denote the capacity of an arc in a given network. Define the capacity of a directed path P as the minimum arc capacity in P. The maximum capacity path problem is to determine a maximum capacity path from a specified source node s to every other node in the network. Modify Dijkstra's algorithm so that it solves the maximum capacity path problem. Justify your algorithm.

34. Suppose that the Floyd-Warshall algorithm terminates after detecting the presence of a negative cycle. At this time, how would you detect a negative cycle using the predecessor indices?

35. In an all-pairs shortest path problem, suppose that several shortest paths connect node i and node j. If we use the Floyd-Warshall algorithm to solve this problem, which path will the algorithm choose? Will this path be the one with the least number of arcs?

36. Show that if we use the Floyd-Warshall algorithm to solve the all-pairs shortest path problem in a network containing a negative cycle, then at some stage $d^k[i, i] < 0$ for some node i. [*Hint:* Let i be the least indexed node satisfying the property that the network contains a negative cycle using only nodes 1 through i (not necessarily all of these nodes).]

37. Suppose that a network G contains no negative cycle. Let $d^{n+1}(i, j)$ denote the node pair distances at the end of the Floyd-Warshall algorithm, when the network is initialized with $d[i, i] = \infty$ instead of $d[i, i] = 0$ for all i .Show that $min\{d_{n+1}[i, i] : 1 \leq i \leq n\}$ is the minimum length of a directed cycle in G.

38. **Sensitivity analysis**. Let d_{ij} denote the shortest path distances between the pair $[i, j]$ of nodes in a directed network G = (N, A) with arc lengths d_{ij}. Suppose that the length of one arc (p, q) changes to value $c'_{pq} < c_{pq}$. Show that the following set of statements finds the modified all-pairs shortest path distances:

if $d_{qp} + c'_{pq} < 0$, then the network has a negative cycle
else
for each pair [i, j] of nodes do
$d_{ij} := min\{d_{ij}, d_{ip} + c'_{pq} + d_{qj}\}$;

39. In question 38 we described an $O(n^2)$ method for updating shortest path distances between all-pairs of nodes when we decrease the length of one arc (p, q). Suppose that we increase the length of the arc (p, q). Can you modify the method so that it reoptimizes the shortest path distances in $O(n^2)$ time? If your answer is yes, specify an algorithm for performing the reoptimization and provide a justification for it; and if your answer is no, outline the difficulties encountered.

40. **Arc addition**. After solving an all-pairs shortest path problem, you realize that you omitted five arcs from the network G. Can you reoptimize the shortest path distances with the addition of these arcs in $O(n^2)$ time? (Hint: Reduce this problem to the one in question 38.)