5.22. 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 $\hat{w}(e) > w(e)$.

5.23. Sometimes we want light spanning trees with certain special properties. Here's an example.

> *Input:* Undirected graph $G = (V, E)$; edge weights w_e; subset of vertices $U \subset V$
> *Output:* The lightest spanning tree in which the nodes of U are *leaves* (there might be other leaves in this tree as well).

(The answer isn't necessarily a minimum spanning tree.)

Give an algorithm for this problem which runs in $O(|E| \log |V|)$ time. (*Hint:* When you remove nodes U from the optimal solution, what is left?)

5.24. A binary counter of unspecified length supports two operations: `increment` (which increases its value by one) and `reset` (which sets its value back to zero). Show that, starting from an initially zero counter, any sequence of n `increment` and `reset` operations takes time $O(n)$; that is, the amortized time per operation is $O(1)$.

5.25. Here's a problem that occurs in automatic program analysis. For a set of variables $x_1, \ldots, x_n$, you are given some *equality* constraints, of the form "$x_i = x_j$" and some *disequality* 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.

5.26. *Graphs with prescribed degree sequences.* Given a list of n positive integers $d_1, d_2, \ldots, d_n$, we want to efficiently determine whether there exists an undirected graph $G = (V, E)$ whose nodes have degrees precisely $d_1, d_2, \ldots, d_n$. That is, if $V = \{v_1, \ldots, v_n\}$, then the degree of v_i should be exactly d_i. We call $(d_1, \ldots, d_n)$ 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_3, d_4 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 $d_1 \geq d_2 \geq \cdots \geq d_n$ and that there exists a graph $G = (V, E)$ with degree sequence $(d_1, \ldots, d_n)$. 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 $v_2, v_3, \ldots, v_{d_1+1}$. 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 $v_2, v_3, \ldots, v_{d_1+1}$. Show that there exists $i < j \leq n$ and $u \in V$ 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 $v_2, v_3, \ldots, v_{d_1+1}$.

(c) Using the result from part (b), describe an algorithm that on input $d_1, \ldots, d_n$ (not necessarily sorted) decides whether there exists a graph with this degree sequence. Your algorithm should run in time polynomial in n and in $m = \sum_{i=1}^{n} d_i$.

5.27. Alice wants to throw a party and is deciding whom to call. She has n people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know *and* five other people whom they don't know.

Give an efficient algorithm that takes as input the list of n people and the list of pairs who know each other and outputs the best choice of party invitees. Give the running time in terms of n.

5.28. A *prefix-free encoding* of a finite alphabet Γ assigns each symbol in Γ a binary codeword, such that no codeword is a prefix of another codeword.

Show that such an encoding can be represented by a full binary tree in which each leaf corresponds to a unique element of Γ, whose codeword is generated by the path from the root to that leaf (interpreting a left branch as 0 and a right branch as 1).

5.29. *Ternary Huffman.* Trimedia Disks Inc. has developed "ternary" hard disks. Each cell on a disk can now store values 0, 1, or 2 (instead of just 0 or 1). To take advantage of this new technology, provide a modified Huffman algorithm for compressing sequences of characters from an alphabet of size n, where the characters occur with known frequencies $f_1, f_2, \ldots, f_n$. Your algorithm should encode each character with a variable-length codeword over the values $0, 1, 2$ such that no codeword is a prefix of another codeword and so as to obtain the maximum possible compression. Prove that your algorithm is correct.

5.30. The basic intuition behind Huffman's algorithm, that frequent blocks should have short encodings and infrequent blocks should have long encodings, is also at work in English, where typical words like `I`, `you`, `is`, `and`, `to`, `from`, and so on are short, and rarely used words like `velociraptor` are longer.

However, words like `fire!`, `help!`, and `run!` are short not because they are frequent, but perhaps because time is precious in situations where they are used.

To make things theoretical, suppose we have a file composed of m different words, with frequencies $f_1, \ldots, f_m$. Suppose also that for the ith word, the cost per bit of encoding is c_i. Thus, if we find a prefix-free code where the ith word has a codeword of length l_i, then the total cost of the encoding will be $\sum_i f_i \cdot c_i \cdot l_i$.

Show how to modify Huffman's algorithm to find the prefix-free encoding of minimum total cost.

5.31. A server has n customers waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. So if, for example, the customers are served in order of increasing i, then the ith customer has to wait $\sum_{j=1}^{i} t_j$ minutes.

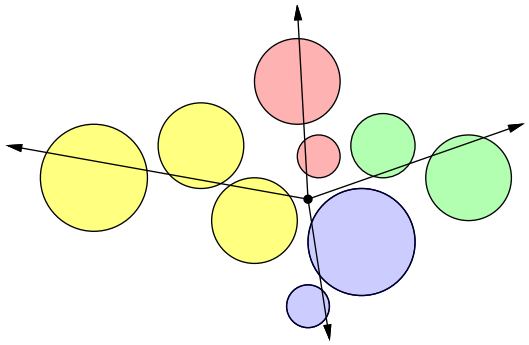
We wish to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i).$$

Give an efficient algorithm for computing the optimal order in which to process the customers.

8.

Suppose you are standing in a field surrounded by several large balloons. You want to use your brand new Acme Brand Zap-O-Matic™ to pop all the balloons, without moving from your current location. The Zap-O-Matic™ shoots a high-powered laser beam, which pops all the balloons it hits. Since each shot requires enough energy to power a small country for a year, you want to fire as few shots as possible.



Nine balloons popped by 4 shots of the Zap-O-Matic™

The *minimum zap* problem can be stated more formally as follows. Given a set C of n circles in the plane, each specified by its radius and the (x, y) coordinates of its center, compute the minimum number of rays from the origin that intersect every circle in C. Your goal is to find an efficient algorithm for this problem.

(a) Suppose it is possible to shoot a ray that does not intersect any balloons. Describe and analyze a greedy algorithm that solves the minimum zap problem in this special case. *[Hint: See Exercise 2.]*

(b) Describe and analyze a greedy algorithm whose output is within 1 of optimal. That is, if m is the minimum number of rays required to hit every balloon, then your greedy algorithm must output either m or $m + 1$. (Of course, you must prove this fact.)