Reminders

When asked to design an algorithm, always assume that you need to prove its correctness and analyze its time and space requirements.

Algorithms in the Wild

**Question 1.** String Pattern-Matching

Let \( \Sigma \) be a finite alphabet (evocative synonym for ‘set’) and let \( w = \sigma_1 \sigma_2 \ldots \sigma_m \) and \( p = \tau_1 \tau_2 \ldots \tau_n \) be two finite sequences of elements of \( \Sigma \). We say that the word \( w \) contains the pattern \( p \) if \( \sigma_1, \sigma_2, \ldots, \sigma_n \) occur in \( w \) in the same order (but not necessarily consecutively) that they occur in \( p \). That is, \( w \) contains \( p \) if for some \( \tau_{i_1}, \ldots, \tau_{i_n} \) with \( i_1 < i_2 < \ldots < i_n \) we have \( \tau_{i_1} = \sigma_1, \ldots, \tau_{i_n} = \sigma_n \).

Note: Precise notation can sometimes take a while to absorb, so reread the previous sentence a few times if needed to make sure you’re comfortable with what it says!

For example, the word \( w = a \ b \ b \ a \ \underline{c} \ b \ a \ c \ d \ f \ b \ a \ c \) contains the pattern \( p = a \ c \ d \ b \) (underlined in \( w \) for emphasis).

Design an \( O(n + m) \) greedy algorithm to determine, given a pair \( w \) and \( p \), whether \( w \) contains the pattern \( p \), and prove that your algorithm correctly solves the problem in this amount of time.

Note: A correct solution needs a greedy algorithm and a “greedy stays ahead”-style proof.

**Question 2.** The Circle Game

Please answer Question 17 from Chapter 4 of your text. Try to shoot for an \( O(n^2) \) algorithm. It’s a bit of a challenge to beat \( O(n^2) \)—although if you’re feeling really ambitious, \( O(n \log n) \) time is possible. Hint: Can you use an algorithm for a related problem studied in class as a tool in this problem?

Spanning Tree Applications

**Question 3.** A feedback edge set in a graph \( G \) is a subset \( F \) of the edges of \( G \) such that every cycle in \( G \) contains at least one edge in \( F \). In other words, removing every edge in \( F \) makes the graph \( G \) acyclic. Design and analyze a fast algorithm to compute the minimum weight feedback edge set of a given edge-weighted graph. Hint. Relate this problem to some kind of spanning tree problem.

**Question 4.** In many situations, you might find yourself dealing with a dynamically changing data structure. One such simple example is that of dynamically changing edge weights in a graph. This problem asks you to consider how you would maintain a minimum-cost spanning tree in such an environment.

Suppose you are given a graph \( G \) with weighted edges and a minimum spanning tree \( T \) of \( G \).

(a) Design and analyze an algorithm to update the minimum spanning tree when the weight of a single edge \( e \) is decreased.

(b) Design and analyze an algorithm to update the minimum spanning tree when the weight of a single edge \( e \) is increased.

In both cases, the input to your algorithm is the edge \( e \) and its new weight; your algorithms should modify \( T \) so that it is still a minimum spanning tree. Hints: consider \( e \in T \) and \( e \notin T \) separately. Also, recall that we’ve proven several useful properties that relate trees, cycles, and cuts: They are useful in correctness proofs!

Theory vs Practice

You need only hand in a solution to one of the two questions below. However, you are strongly encouraged to give some thought to both of them! In particular, the Euler tour concept is likely to reappear at some later point in the semester.

---

1 Gordon Gekko, *Wall Street*

2 Tradition: Greek letters are used when denoting alphabets and their “letters”.
Question 5. An Euler tour of a graph $G$ is a circuit (that is, a path that begins and ends at the same vertex), through $G$ that traverses every edge of $G$ exactly once.

(a) Prove that a connected graph $G$ has an Euler tour if and only if every vertex has even degree.

(b) Design and analyze a linear time algorithm to compute an Euler tour in a given graph, or correctly report that no such tour exists.

Question 6 (maze generation). Kruskal’s and Prim’s algorithms take some effort to analyze for correctness, but both are conceptually simple and easy to program. They can also be applied in some unusual ways, such as for maze generation. There’s a cute Wikipedia entry on the topic:

http://en.wikipedia.org/wiki/Maze_generation_algorithm

Read the sections about randomized Kruskal’s and randomized Prim’s algorithms and implement a maze generation program using one of the algorithms. Do not read or refer to any actual implementation code on that Wikipedia page (or anywhere else) in writing your program! Your program should take two parameters, an integer width $w$ and an integer height $h$, and should generate a reasonable visualization of a $w$ by $h$ maze. The visualization needn’t be fancy: printing the maze walls as a grid of x’s, for example, is fine (although feel free to get fancy if you’d like!).

Note: Your program must be written so that I can run it easily as submitted on the departmental unix machines, and it should be written in either Java (a language I know), C or C++ (languages I used to know when I was a boy), or Python (a language I’d like to know better). Solutions should be submitted using the turnin command on one of the CS unix machines.

---

3 Named after Leonhard Euler (1707-1783) who solved part (a) in 1735
4 Typing turnin on any of the CS unix machines should give some (terse) instructions.