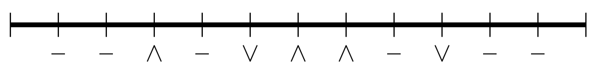
Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
Erik Demaine and David Karger

Handout 2
September 3, 2003

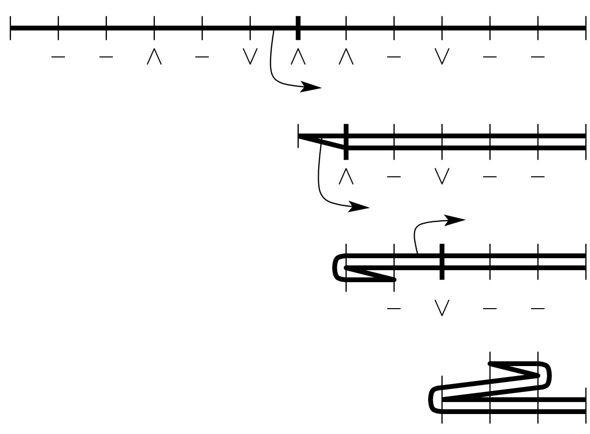
Problem Set 1

Due: Monday, September 8, 2003.

Problem 1. Suppose you are given a thin strip of paper marked with creases at regular intervals, which we can think of as a line segment divided into equal-length subsegments. Each crease point is marked as *mountain*, *valley*, or *flat* to specify the orientation of the crease ($\wedge$, $\vee$, or $-$). For example, the input might look as follows:



Your goal is to find a folding of the strip that uses all the marked creases with the specified orientations. The operations you are allowed are as follows. Given a particular crease ($\wedge$ or $\vee$), a *simple fold* rotates the portion of the segment to the left of the crease around the portion of the segment to the right of the crease. The rotation is counterclockwise for a mountain fold, and clockwise for a valley fold. When multiple layers of paper come in contact, they become inseparable; in other words, each simple fold must fold all layers of paper. For example, here is how we might fold the example above:



Notice that when a subsegment is folded, the creases turn upside-down, inverting $\wedge \leftrightarrow \vee$. Thus, for a simple fold to be *valid*, the inversions of the creases to the left must match the creases to the right.

(a) Give a linear-time algorithm that tests whether a given crease admits a valid simple fold.

(b) Prove that making a valid simple fold preserves foldability. In other words, prove that the original input is foldable by simple folds precisely if the result of a simple fold is foldable by simple folds.

(c) Design and analyze an algorithm with linear expected running time that finds a valid simple fold or reports that none exist.

(d) Design and analyzean algorithm with linear expected running time that finds a folding by a sequence of valid simple folds or reports that no such folding exists.

Problem 2. Build an uncompressed suffix tree for "banana$". Show the structure and node traversal path for each suffix insertion. Mark the suffix links that are actually used as shortcuts in the efficient construction algorithm.