CS472 Module 8 Part A - Backtracking

Athens State University

March 28, 2016

An excerpt from *The Spell-Checker Poem*

Eye have a spelling chequer,
It came with my Pea Sea.
It plane lee marks four my revue
Miss Steaks I can knot sea.
Eye strike the quays and type a whirred
And weight four it two say
Weather eye am write oar wrong
It tells me straight a weigh.
Eye ran this poem threw it,
Your shore real glad two no.
Its vary polished in its weigh.
My chequer tolled me sew.

A spell checker must:

- Scan the text being checked and extract the words contained in it
- Compare each word with a known list of correctly spelled words
- Do some form of language-dependent morphological analysis

Morphological Analysis?

- Even in a lightly inflected language like English, a spell checker has to consider different forms of the same word
  - Plurals, verbal forms, contractions, possessives

This gets worse for languages other English

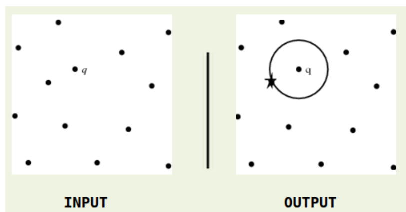
- For other languages, you have worry about
  - Complex declension and conjugation (for example, German has 9 different equivalents of the article "the" in English)
  - Agglutination: Many languages string together morphemes (smallest unit that has meaning in a language) to form new words without changing spelling or phonetics
    * Example: In Turkish, "evierinizden" means "from your houses" is formed by combining "ev-ler-iniz-den" ("house-plural-your from").
    * Japanese includes verb morphemes that indicate negation, passive voice, past tense, honorific degree, and causality

Making suggestions

- Suggesting alternative spellings require us to search for the closest word in a dictionary
- We can define "closest word" as the minimum number of character transformations (add, delete, replace) that one has to make to transform a word into a second word
- The brute force approach is start with all words that are 1 change away from our base word
- If not found, then look for all words that are 2 changes away from the base word, and so on

Nearest Neighbor Search

- Nearest-neighbor search: given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q



***kd*-trees**

- **Space partitioning**: dividing a space (like a plane) into non-overlapping regions
- The closest match problem in spell checkers is doing this in the associated dictionary
- We can use a plane to divide a space and assign points to different sets based upon which side of the plane where the point lies
- Recursively applied these scheme results in a tree structure

Building a *kd*-tree

Algorithm 1: kdtree(): builds a kd-tree

Input: A list of points *pointlist* and an integer *depth*
Output: A *kd* tree pointed at by a root node
axis ← depth MOD k;
Sort the point list and chose median point as pivot element;
Create a new tree node;
node.location ← median; beforelist ← points in pointList before median;
afterList ← points in pointList after median;
node.leftChild ← kdtree(beforeList, depth+1);
node.rightChild ← kdtree(afterList, depth+1);
return *node*;

The Nearest Neighbor Search Algorithm

- With a *kd*-tree, the NN problem can be solved by finding the point in the tree that is nearest to a given input point
  1. Traverse the tree as if attempting to insert a new point until you reach a leaf node
  2. Set the discovered leaf node as your current "best guess"
  3. Unwind the the recursion, performing the following
     (a) If current node is closer than the current best, set it to current best
     (b) Look to the other side of the plane to see if you have a better guess on that side. If so, recursively search down that branch. If not, prune that entire branch from the tree.
     (c) Search is complete when you return tot he root

Branch-and-Bound

- The NN search algorithm is an example of a branch-and-bound algorithm
  - The "look to the other side" aspect of the algorithm is the branch-and-bound step
- Similar to backtracking in that you're working with a state space tree
- Difference is that you're not limited to a particular tree traversal and b-and-b is used only for optimization problems

Bounds

- Bound computed at a node determine whether the node is promising
- Bound on the value of the solution that could be obtained by expanding beyond the code
- If bound is no better than the value of the best solution found so far, node is non-promising

The technique

- Do a BFS of the state space tree
  - Visit root first, all nodes at level 1, all nodes at level
  - Operates on a queue of nodes
- Check to see if nodes are "promising", prune paths that are not.