Other directions for consideration

Types of directions:
- New criteria for assessment
- New data structures for common ADTs
- New ways of using ADTs together
- New ADTs for different situations

New criteria for assessment
- Amortized analysis (cost per operation over a sequence of operations)
- Use of space (e.g. succinct data structures)
- Support for change or multiple versions of data
New data structures for ADT Dictionary: skip list

Ideas:

- Combines strong points of linked structures (easy to modify) and array (binary search)
- List of nodes of different “heights”, allows quick search between tall nodes.
- Uses randomization to keep nice distribution of heights even as changed by addition and deletion.
New ways of using ADTs together

**Minimum spanning tree**: Given a weighted graph, choose edges to form a tree including all vertices in the graph such that the sum of the weights is minimized.

**Total order**: Given a list of tasks and information about which should be done before which, determine an ordering.

**Connectivity**: Given a graph, determine the **connected components** (sets of vertices such that there is path between each pair of vertices in the set).

**Single-source shortest paths**: Given a weighted graph and a source vertex, determine the minimum costs of paths from the source to each other vertex.
New ADTs for graph problems

ADT Disjoint Set

- CreateSet()
- FindRep(S) - returns an element of the set
- Union(S, T) - forms one set out of two

Extensions to Priority Queue:

- Mergeable or Meldable Heap: Add an operation that allows two structures to be joined into one.
- Binary heap: Also add delete and decrease key operations.
New ADTs for different situations

Problem domains:
- Computational geometry
- Strings (including problems in bioinformatics)

ADTs:
- Structures storing the location of points or objects in space
- Structures based on each input being a sequence of symbols taken from a finite set: tries, suffix trees
Multi-dimensional data

ADT Range
Data: \(d\)-tuples of numbers

Operations:

- \texttt{CreateRange()}
- \texttt{IsEmptyRange(R)}
- \texttt{LookUpRange(R, small1, big1, \ldots, smalld, bigd)}
- \texttt{AddToRange(R, item)}
- \texttt{DeleteFromRange(R, item)}
One-dimensional range search in a BST

Range from 30 to 80

- **inside node**: it and its descendants are all in the range
- **outside node**: it and its descendants are all outside the range
- **boundary node**: it has descendants both in and out of the range
One-dimensional range search

Alternatives include:

- At each node, store the largest value in its left subtree.
- Using in-order, “thread” nodes.
- Store values in leaves, “thread” leaves.
Quadtrees

Idea:
- Represent a square region as a tree
- Root represents entire region
- Leaves represent the points being stored
- One child for each quadrant that contains a point

Creating a quadtree:
- Starting with the whole region, split into four.
- Omit nodes for empty pieces.
- Repeat recursively.

Extensions:
- Fix maximum depth and allow multiple points per leaf.
- Octtrees for 3-dimensional data.
- Poor generalization for higher dimensions.
k-d trees

- Binary trees for k-dimensional data
- Nodes = regions that are not necessarily square
- Splits depend on distributions of points in a region

Creating a k-d tree

- Find dimension which represents biggest range in values.
- Split along this dimension (equal number of points in each part), one right at the split as the parent.
- Logarithmic depth.
Strings

Types of problems:

- Coding and decoding
- Text compression

Observations:

- A string can be viewed as a sequence of symbols.
- The symbols can be used to handle a set of strings in ADT Dictionary in a digital way.
- With each symbol taken from a fixed size set, there is a bound on the number of choices.
A **trie** stores strings as leaves in an ordered tree, where each node can have at most one child for each possible symbol, and the leaf storing a string can be found from the root by processing each symbol in turn, taking the child with that symbol at each step.

The term trie is pronounced “tree” for “reTRIEval”.

A **suffix tree** is a trie used to store all the suffixes of a string, and can be used for pattern matching.
Future directions

To use what you have learned:

- Planning - get practice solving problems from lots of application areas.
- Coding - write implementations of many variants discussed but not coded in class.

To learn more:

- CS 231 (Spring 2018 and on) on algorithms
- CS 338 on managing large amounts of data
- Courses in application areas of computer science