IE 495 Lecture 16

October 24, 2000

Reading for This Lecture

- Primary
  - Horowitz and Sahni, Chapter 4
  - Kozen, Lecture 3
- Secondary
  - Miller and Boxer, Chapter 12 (up to page 286)

Prim's Algorithm

S is the set of nodes in the tree

```
S = {0}
for (i = 0; i < n; i++){
  SELECT i ∉ S nearest to S;
  S = UNION(S, i);
}
```

Kruskal's Algorithm

T is the set of edges in the tree

```
T = Ø

for (i = 0; i < m; i++){
  SELECT the cheapest edge e
  if (feasible(UNION(T, e)){
    UNION(T, e);

}
```

The Red and Blue Rules

- Start with all edges uncolored
- The Blue Rule:
  - Find a cut with no BLUE edges.
  - Pick an edge of minimum weight in the cut and color it BLUE.
- The Red Rule:
  - Find a cycle containing no RED edges.
  - Pick an uncolored edge of maximum weight and color it RED.
- Arbitrary application of the Red and Blue rules will result in a minimum spanning tree (blue edges).

Matroids

- A *matroid* is a pair $(S, \mathcal{I})$ where S is a finite set and $\mathcal{I}$ is a family of subsets of S such that

  (i) If $J \in \mathcal{I}$ and $I \subseteq J$, then $I \in \mathcal{I}$

  (ii) If I, J and $|I| < |J|$, then there exists and $x \in J \setminus I$ such that $I \cup \{x\} \in \mathcal{I}$

- Elements of $\mathcal{I}$ are called the *independent sets*.
- Note that all independent sets have the same cardinality.
- A *cycle* is a setwise minimal dependent set.
- A *cut* is a setwise minimal subset of S intersecting all maximal independent sets.

Matroid Examples

- Graph $G = (V, E)$
  - I is the set of forests in G.
  - I is the set of subsets E' of E such $G \backslash E'$ is connected.
- Vector space V
  - I is the set of all linearly independent subsets of V.
- Columns/rows of a matrix A
  - I is the set of all bases of A.

Importance of Matroids

- Why study matroids?
- Matroids are common mathematical structures.
- In a matroid, we can always find the minimum-weight maximal independent set using the greedy algorithm.
- Algorithm: Apply the Red and Blue rules arbitrarily.
- In fact, (S, I) satisfying property (i) is a matroid if and only if we can find a minimum-weight maximal independent set using the greedy algorithm!

Matroid duality

- The dual of a matroid (S, I) is (S, I^*) where

  $I^* = \{S' \subseteq S$ disjoint from some maximal element of $I\}$

- The maximal elements of I^* are the complements of the maximal elements of I.

- Properties

  – Cuts in (S, I) are cycles in (S, I^*).

  – The blue rule in (S, I) is the red rule in (S, I^*) with the weights reversed.

Single-source Shortest Paths

- Given an undirected graph $G = (V, E)$, a length l_e for each edge e, and a source vertex v_0.
- We are looking for the *shortest path* from v_0 to all other vertices in the graph.
- The algorithm is almost identical to Prim's MST algorithm.

Dijkstra's Algorithm

```
S is the set of nodes that have been
  examined

S = {0}

d[v] = c(0,v) ∀v∈V\S

for (i = 1; i < n; i++){
  SELECT w ∉ S with minimum d[w];
  S = UNION(S, w);
  set d[v] = min(d[v], d[w]+c(w,v));

}
```

Analysis of Dijkstra's Algorithm

- Correctness
- Optimality
- Implementation
- Complexity

Search Algorithms

The Bin Packing Problem

- We are given a set of *n* items, each with a size/weight w_i
- We are also given a set of bins of capacity *C*.
- Bin Packing Problem: Pack the items into the smallest number of bins possible.
- The total size/weight of items assigned to each bin must not exceed the capacity *C*.
- This problem is *NP*-complete.

Complexity Classes

- *P* is the class of problems for which there exists polynomial-time algorithms (on a Turing machine).
- *NP* is the set of all problems for which there exists a polynomial-time algorithm on a non-deterministic Turing machine.
- A non-deterministic polynomial-time Turing machine essentially allows "infinite parallelism".
- Hence, any problem which can be solved using a search tree of polynomial depth is in *NP*.
- Note that any problem in *P* is also in *NP*.

NP-complete Problems

- Another way to think of the class *NP* is as the class of problems for which we can verify the feasibility of a given solution in polynomial time.
- The *NP*-complete problems are the "hardest" problems in *NP*.
- If we can solve some *NP*-complete problem in polynomial-time, then *P* = *NP*.
- Note that the theory of *NP*-completeness applies only to *decision* problems.

Back to Bin-packing

- We cannot hope for a polynomial-time algorithm for this problem.
- How do we solve it?

Heuristic Methods

- Heuristic methods derive an approximate solution quickly (usually polynomial time).
- Heuristics for the Bin Packing Problem.
- Performance guarantees.