CSC373S Lecture 4

- We now return to the material in the text and Jepson notes.

  In the Greedy algorithms notes (pages 17-22) and text (section 4.2), a minimum (max) lateness problem is discussed. The input is a set of jobs, each having a processing time and deadline; that is $J(i) = (t_i, d_i)$. All jobs must be scheduled on one machine (with no overlap). If a job is started at some time s_i then it will run (uninterupted) and finish at time $f_i = s_i + t_i$. The lateness of job $J(i)$ is defined as $L_i = \max\{0, f_i - d_i\}$ and the goal is to minimize the maximum lateness $L = \max_i L_i$.

  In the scheduling literature, when jobs run without interuption, this is called "no preemption". Scheduling problems also consider "preemption" (at various costs).

  The notes give a number of plausible greedy algorithms for the min lateness problem and then show that the fixed order $d_1 \leq d_2 \ldots \leq d_n$ results in an optimal greedy algorithm. Note that in this problem, once the ordering is fixed, the decision is mandated. See page 21.

  It is easy to show that this is optimal by an exchange argument. Namely, given any solution, it can be transformed into a solution satisfying the ordering by deadlines. Formally, this is an induction on the number of adjacent transpoitions in a solution or a proof by contradiction (if one asserts that there is no optimal solution using this order). But informally, one just finds an adjacent pair "out of order" and then show that transposing these jobs cannot increase the maximum lateness. For assignment question 4, you can use the same kind of exchange argument.

- In the Greedy graph algorithms notes and the text, two important graph problems are discussed. The first problem is the "single source shortest distance" problem. I would rather refer to it as the single source "least cost" problem since "distance" can be confused with graph distance (i.e. length of a path) as opposed to the cost of a path.

  In any case, we are given an edge weighted directed graph $G = (V, E)$ with weight function $w : E \to \Re^+$; note all edge weights are positive or non negative. We let $w(e)$ denote the weight of edge e. We also assume that every vertex is reachable (by at least one path) from s. (See notes pages 2-9.) The goal is to find the least cost path from a distinguished vertex s to all nodes $v \in V$. Dijkstra's algorithm depends on the edge costs being non-negative. Later we will use dynamic programming (DP) to deal with negative cost edges. (There are ways to run Dijkstra's alg iteratively so as to deal with negative edges but we won't be discussing that.)

  Dijkstra's algorithm appears on page 5 of the Greedy graph algorithms notes.

  Things to observe:

  1. Dijkstra's algorithm is the basis for GPS navigation systems. One can think of the destination as the "source" and the current location as the target.

2. Since all edges are non negative, the least cost path is a simple path.
3. We will see that the solution (ie the set of paths) becomes a directed tree with root s.
4. Dijkstra's algorithm (for single source to all nodes) can be viewed as a greedy algorithm where the choice of the next item (e.g. edge to choose or vertex to extend) is chosen adaptively (i.e. based on what has been decided this far). Aside: for the problem of computing an optimal $s-t$ path for a fixed target t, I would no longer consider Dijkstra to be a greedy algorithm but rather better viewed as a DP.
5. The proof is by induction showing that for any $v \in S_i$, we have computed the least cost path to v where S_i is the set of nodes reached by the end of the i^{th} iteration. In the proof there is a claim and the first inequality in this claim is an equality.

- The last question on the problem set shows that Dijkstra's algorithm can be generalized to work for other path problems.
- The other graph theoretic problem in the notes is the minimum spanning tree MST problem. The notes mention three optimal algorithms, namely Kruskals algorithm, Prims algorithm and the Reverse-Delete algorithm.

  Things to observe:

  1. Kruskal's algorithm applied to any graph will determine the connected compenents of the graph and the MST for each component. That is, Kruskal's alg computes a min cost spanning forest, with an MST for each component. Prim's algorithm will find the MST of the connected component in which the starting node s occurs. It can be run a number of times to find all connected components. The reverse-delete algorithm is not well defined if G is not connected but can be defined to work with all graphs. It can be shown that this algorithm will always select the same set of edges as Kruskals algorithm (when the MST is not unique). The reverse delete alg does not seem easy to implement efficiently.
  2. Kruskal's algorithm can be veiwed as a fixed order greedy algorithm where items are edges sorted by $c(e_1) \leq c(e_2) \ldots \leq c(e_m)$ and edges taken greedily in the sense that whenever the current edge being considered connects two components, it is taken. Prim's algorithm can be viewed as an adaptive greedy algorithm in that the next edge to be chosen depends on the current component being constructed.
  3. Unlike Dijkstra's least cost algorithm, the MST algorithm works (computes an optimal MST) when there are negative edges. This implies that we can also use the MST algorithm to compute the maximum cost spanning tree (or max cost spanning forest) by taking the negative of each edge. More directly, we can use the ordering $c(e_1) \geq c(e_2) \ldots \geq c(e_m)$ and again accept greedily (never forming a cycle).

4. Kruskal's greedy algorithm for MST can be abstracted to a wider class of problems when one realizes that tree edges $T \subseteq E$ in a graph determine an independence set system called *matroids*. Matroids also abstract (and this is the initial motivation) linear indepedence in a vector space. The key property (leading to many other facts) is that any independent set of elements can be extended to a *basis*. In a connected graph, the edges in a spanning tree is a basis (i.e. has maximal number of independent elements). This standard greedy alg of ordering the elements by largest weight is an optimal alg for optimally computing a linear function over any matroid.
5. The proof of Kruskal's algorithm that I like can be phrased as a "promising argument"; namely, the partial solution S_i at the end of the i^{th} iteration can be extended to an optimal solution OPT_i. The essense of this argument is that if one adds an edge to any spanning tree, a unique cycle C will be formed. If the current edge e that Kruskal's algorithm is adding to S_i is not in OPT_i then there has to be an edge e' in the cycle (that is not in the Greedy solution) with weight $w(e') \geq w(e)$ and it can be removed to form a new spanning tree which is at least as good. This proof then abstracts to the more abstract setting of matroids mentioned above.
6. The application to clustering data is worth noting. There are many reasonable ways to define the objective function in a clustering. Some defintions lead to NP hard optimization problems but usually ones that in some sense can be computed "practically". In general, "clustering" is an informal concept and there is no one best definition but it is an important concept used in many applications.

- Next time we will present one more greedy algorithm, the online greedy approximation for the makespan problem. See text section 11.1 and Approximation algorithm notes.