CSC373S Lecture 10

- Brief review of least cost algorithms
- There are many other notable examples of the kind of DP (achieving an optimal parse) used in the iterated matrix multiplication problem. Question 3 in assignment 2 uses a similar DP. (Hint: choose an arbitrary edge of the polygon and choose a vertex to complete a triangle.) For those who have the CLRS text, there is a discussion (in section 15.5) of the optimal binary search tree problem.
- I want to continue the current discussion of DP algorithms by an application that is the stating point for a number of genomic applications as well as for spell checkers. Namely, consider the edit distance problem, which the KT text refers to as the sequence alignment problem. (I think "alignment" implies a more limited set of operations so I prefer edit distance.) We are given two strings X and Y say over a finite alphabet Σ. For a spell checker, $\Sigma = \{a, b, c, \ldots z\}$ and for genome applications $\Sigma = \{A, C.G, T\}$. As a cost/distance problem, we are trying to compute the cost of "aligning" the two strings (or more generally to me, the distance between the strings in terms of edit operations that need to be performed). This cost or distance between the strings depends on what operations are allowed and the cost of each operation. For example, the text considers a "gap cost" δ to delete a symbol and a "mismatch cost" α_{pq} to "line up" symbols p and q in Σ. There could be many other edit operations, such as transposing adjacent symbols, reversing a substring, deleting many consecutive symbols at a fixed cost + cost δ' /per delete with $\delta' < \delta$, different costs for different deletions, different costs for deletions and insertions, etc. Returning to the basic problem as defined in the text, let $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$. The DP approach is to define a semantic array $M'[i, j]$ = minimum cost to align the substrings $x_1 \ldots x_i$ and $y_1 \ldots y_j$. In this alignment problem, we are trying to define an optimal way to match symbols in X with symbols in Y. Our choices here are to match symbols or delete a symbol from one of the strings. That is, the computational array is then $M[i, 0] = i \cdot \delta$, $M[0, j] = j \cdot \delta$, and for $i, j \geq 1$, $M[i, j] = \min\{A, B, C\}$ where $A = M[i-1, j-1] + \alpha_{x_i y_j}$, $B = \delta + M[i-1, j]$, $C = \delta + M[i, j-1]$.
- We conclude our current discussion of DP algorithms by noting that sometimes they can be used to derive an exponential algorithm but still one that is better than a more naive algorithm. For example, consider the travelling salesman problem. Lets say we have a directed graph $G = (V, E)$ with $|V| = n$ and edge costs $\{c(u, v)\}$. If $(u, v) \notin E$, we can add (u, v) to E by setting $c(u, v) = \infty$. So without loss of generality we can assume all edges exist.
  The celebrated traveling salesman problem asks for a minimum cost simple cycle that includes all the vertices (such a cycle is called a Hamiltonian cycle HC). Selecting any node $u \in V$, a Hamiltonian cycle is then defined by a permutation on the remaining $n - 1$ nodes plus the edge back to u. It follows that a brute force solution that considers every HC to find the minimum cost HC would costs $O(n!)$ which is very roughly about $O(n^n)$. We can use DP to achieve complexity $O(2^n)$ which is still not practical for large n but is substantially better than n^n.

Here is the DP approach.: Fix an arbitrary starting node s. Consider the following semantic array: For each $t \in V$ and $S \subseteq V - \{s, t\}$, $H'(S, t)$ is the minimum cost of a simple path π from s to t where the set of internal nodes of π is equal to S (in some order). The desired optimum cost is then $\min_{v \neq s} H'(V - \{s, v\}, v) + c(v, s)$. We recursively compute $H'(S, t)$ for all $t \in V$ and $S \subseteq V - \{s, t\}$ by $H(S, t) = \min_{u \in S - \{s,t\}} H(S - \{u\}, u) + c(u, t)$. The basis is $H(\emptyset, t) = c(s, t)$.