Introduction to Dynamic Programming

Dynamic programming offers an alternative to solving optimization problems that constructs solutions to problems utilizing a method that is similar to two previous methods discussed in this course:

- **Divide-and-conquer:** Solutions are constructed by solving subproblems and then combining to determine the solution to the global problem. This idea is used in dynamic programming in that we require optimal solutions to subproblems when we make choices at the global level. Dynamic programming differs from divide-and-conquer in that dynamic programming algorithms usually do not solve problems recursively because subproblems may be considered multiple times (although, they can be made recursive by using a technique called "memoization").
- **Greedy choice:** Solutions are constructed by making locally optimal choices under the assumption that these choices are independent of previous solutions. Greedy strategies depend upon the principle of optimality in that they depend on previous local choices being optimal. This idea is used in dynamic programming in that we still make a greedy choice among a set of alternatives. Dynamic programming differs from the greedy strategy in that, based on the way the subproblems are created, the decision are no longer local.

Typically, a dynamic programming algorithm is defined by following four steps:

1. Characterize the structure of an optimal solution (applying the principle of optimality).
2. Recursively define the value of an optimal solution (i.e., define the Bellman equation).
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from the computed value function (i.e., Bellman equation).

Given this procedure, we have two key ingredients to dynamic programming problems—optimal substructure and overlapping subproblems. Optimal substructure arises when an optimal solution to a problem contains within it the optimal solutions to its subproblems. The algorithm must then consider a set of possible "splits" of the problem (like divide-and-conquer) and choose the best split based on the values of those subproblems. Since we must select the appropriate split, this suggests there exists overlap among the subproblems such that the split cannot be made without considering those values. As a result, it is preferable that the space of subproblems be kept small (i.e., polynomial in the input size) to manage complexity. Also, because of the overlap of subproblems, a simple recursive approach naturally leads to resolving previously-solved problems unless information obtained from previous solutions is stored in a table for simple lookup. In fact, it is customary to solve dynamic programming in a bottom-up fashion to build this table and avoid the problems arising from recursion.

The main idea that captures both optimal substructure and overlapping subproblems is a mathematical form of the value function for a given subproblem. This mathematical form was first suggested by Richard Bellman in 1957 when Bellman invented the dynamic programming method. As a result, this mathematical form has since become known as the "Bellman form" or the "Bellman equation." A typical form of a Bellman equation is as follows:

$$V_{ij} = \min_{i \leq k < j} \{c_{ijk} + V_{ik} + V_{kj}\}.$$

In this form, note that we are minimizing (or maximizing) over a choice of split defined by the value k. The actual value function takes into account the value of optimal subproblem solutions V_{ik} and V_{kj} and adds into those values the current cost of the decision c_{ijk}.

Note that the actual Bellman form can vary substantially from this general form, and the challenge to finding dynamic programming solutions to problems is coming up with this equation. Once the equation is found, the algorithm itself generally follows in a straightforward fashion.