Source: Stanford CS 161

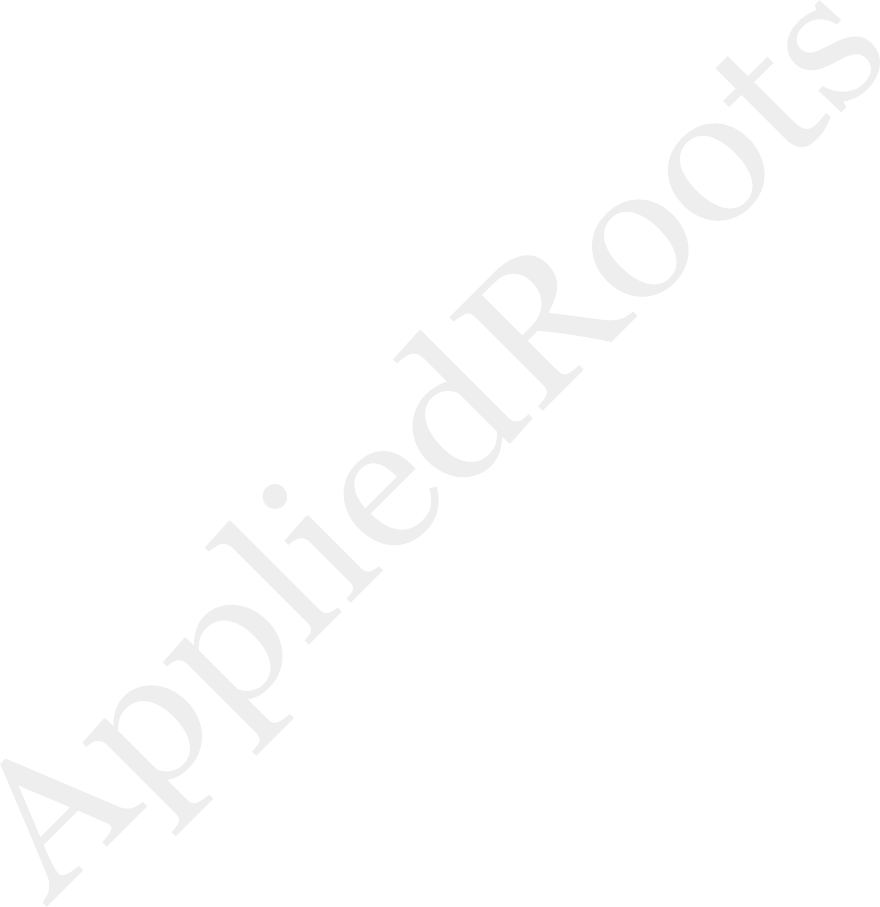
1. Dynamic Programming

The idea of dynamic programming is to have a table of solutions of subproblems and fill it out in a particular order (e.g. left to right and top to bottom) so that the contents of any particular table cell only depends on the contents of cells before it. In this lecture, we will see two examples: the Floyd-Warshall algorithm and the longest common subsequence problem.

* 1. **Dynamic Programming Algorithm Recipe**

Here, we give a general recipe for solving problems by dynamic programming. Dynamic programming is a good candidate paradigm to use for problems with the following properties:

* + - Optimal substructure gives a recursive formulation; and
    - Overlapping subproblems give a small table, that is, we can store the precomputed answers such that it

doesn’t actually take too long when evaluating a recursive function multiple times.

# Optimal Substructure

By this property, we mean that the optimal solution to the problem is composed of optimal solutions to smaller

*independent* subproblems.

For example, the shortest path from *s* to *t* consists of a shortest path *P* from *s* to *k* (for node *k* on *P*) and a shortest path from *k* to *t*. This allows us to write down an expression for the distance between *s* and *t* with respect to the lengths of sub-paths:

*d*(*s,t*) = *d*(*s,k*) + *d*(*k,t*)*,* for all *k* on a shortest *s* − *t* path

# Overlapping subproblems

The goal of dynamic programming is to construct a table of entries, where early entries in the table can be used to compute later entries. Ideally, the optimal solutions of subproblems can be reused multiple times to compute the optimal solutions of larger problems.

For our shortest paths example, *d*(*s,k*) can used to compute *d*(*s,t*) for any *t* where the shortest *s*−*t* path contains *k*. To save time, we can compute *d*(*s,k*) once and just look it up each time, instead of recomputing it.

# Implementations

The above two properties lead to two different ways to implement dynamic programming algorithms. In each, we will store a table *T* with optimal solutions to subproblems; the two variants differ in how we decide to fill up the table:

* + - 1. Bottom-up: Here, we will fill in the table starting with the smallest subproblems. Then, assuming that we have computed the optimal solution to small subproblems, we can compute the answers for larger subproblems using our recursive optimal substructure.
      2. Top-down: In this approach, we will compute the optimal solution to the entire problem recursively. At each recursive call, we will end up looking up the answer or filling in the table if the entry has not been computed yet.

In fact, these two methods are completely equivalent. Any dynamic programming algorithm can be formulated as an iterative table-filling algorithm or a recursive algorithm with look-ups.

1