In previous class, we were discussing about KMP algorithm for string matching. Here are a few more details of the same.

In the pseudo-code for compute_Π, variable ‘k’ represents how far you have gone in the second best. Here, k increases by 1 at most m times. Also, k decreases at the step k ← Π(k) step but cannot be negative. Hence the running time for the algorithm is evaluated as O(m). This analysis is called Amortized analysis.

Dynamic Programming.

We wish to introduce concepts in dynamic programming by analyzing a few examples.

Problem: Shortest path in a directed acyclic graph (DAG)

A graph here is a set of nodes and edges. The first node is the source and the last node is the destination. Each node has 2 edges.

Graphs can be linearized. The following is the linearized version of the above graph. However, it is important to note that in a linearized graph, every edge goes from left to right.
The only way to get to E is from D or B. If we knew the shortest paths to B and D, we can say that
\[ \text{dist}(E) = \min \{ \text{dist}(B) + 2 ; \text{dist}(D) + 1 \} \]

Where function dist(E) implies the shortest path to E, dist(B) implies the shortest path to B and so on.

This brings us to an important feature of dynamic programming. The optimal solutions to the shorter problems (sub-problems) combine to create the final solution to the main problem. This feature clearly provides a recursive algorithm. But this can also lead to repetitive calculations and can make the algorithm highly inefficient when formulated.

The solution to the above problem lies in storing the solutions to the sub-problems in a table, thereby reducing the number of calculations to just once.

Algorithm:

\[
\begin{align*}
\text{dist}(.) & \leftarrow \infty \text{ (for every node)} & \text{represents an array} \\
\text{dist}(S) & \leftarrow 0 \\
\text{for } \forall v \in V - \{S\} \text{ in linearized order} & \\
\text{dist}(v) & = \min \{ \text{dist}(u) + w(u, v) \} \\
\end{align*}
\]

\(w(u, v)\) represents the weight between nodes u and v. For eg, \(w(S, A) = 1\).

The above algorithm can be executed in \(O(|E|)\) or \(O(|V|^2)\) complexity, since every edge is touched only once.

**Longest Increasing Subsequence**

Consider the following:

5 2 8 6 3 6 9 7  

The above diagram can be converted to a DAG as follows:

\[ \text{and so on.} \]
String Alignment:

Suppose we wish to convert the string TRAIN to CRANE. We allow only 3 operations on the string, namely:

- Substitution.
- Insertion
- Deletion. (all 3 operations on a single character in the string)

Our aim here is to convert string1 (i.e. TRAIN) to string2 (i.e. CRANE) using minimum number of operations.

The following is one way of doing it:

1) Start with TRAIN. Substitute T with C. This gives us CRAIN.
2) Delete I. This gives us CRAN.
3) Insert E at the end. This gives us CRANE.

Another way of representing is as given below:

\[
\begin{array}{c}
T \\
R \\
A \\
I \\
N \\
\_ \\
C \\
R \\
A \\
_ \\
N \\
\_.
\end{array}
\]

If a similar score of 1 is given to each of the substitutions, insertions and deletions, then we’d get an overall score of 3 in the above alignment. However, this may not be the case always. Substitutions, insertions and deletions may be given different scores. Moreover, different types of substitutions may be given different scores too.

For the same problem, if we consider a space of all solutions, starting from first letter and going step by step, we get:

And so on…
In the first stage, there are 3 alignments. The next stage, there are $3^2$ alignments and so on. For a string of length ‘n’, there are at least $3^n$ alignments to choose from. Thus $\Omega(3^n)$ is the space of solutions.

For the alignment steps which have been marked with bold arrows, any alignment that follows is better than the ones that stem from it.

The alignments can thus be represented in the form of graphs, as follows:

Where X and Y are the 2 sequences and ‘i’ from X, ‘j’ from Y implies that we are aligning ‘i’ characters from X and ‘j’ characters from Y.

The above diagram can be converted to a DAG and the shortest path be computed. This can also be converted to a matrix.