Introduction to Algorithms
Assignment 1

Sourabh S. Shenoy
(UIN:225009050)

September 16, 2016

1. Question 1

1.1 Rod Cutting Problem

Show by means of a counter example that the following "Greedy" strategy does not always determine an optimal way to cut the rods. Define the density of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n *cuts off a first piece of length i, where* $1 \leq i \leq n$*, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n-i.*

1.2 Solution

Following is a counterexample that proves that the suggested Greedy strategy does not always give an optimal solution:
Consider a rod of length 8 and let the following be the prices of the pieces and the price per inch, respectively:

Length of the rod, i	1	2	3	4	5	6	7	8
Price, p_i	1	4	6	12	10	18	28	8
Price per unit length, p_i/i	1	2	2	3	2	3	4	1

If we use the greedy strategy suggested in the question, we pick the piece with the highest price per unit density, which in this case is length 7. We then have a piece of length 1, which gives us a combined price of (4+1) = 5.

However, this is not the optimal solution. A closer observation reveals that on cutting the rod into 4 pieces of length 2 each, we obtain a price of 8, which is the optimal solution for this problem.

2. Question 2

2.1 Viterbi Algorithm

We can use Dynamic programming on a directed graph G=(V,E) for speech recognition. Each edge $(u,v) \in E$ is labelled with a sound $\sigma(u,v)$ from a finite set ϵ of sounds. Each path in the graph starting from a distinguished vertex $v_0 \in V$ corresponds to a possible sequence of sounds produced by the model. We define the label of a directed path to be the concatenation of the labels of the edges on that path.

a. Describe an efficient algorithm that, given an edge-labeled graph G with distinguished vertex 0 and a sequence s of sounds from ϵ, returns a path in G that begins at 0 and has s as its label, if any such path exists. Otherwise, the algorithm should return NO-SUCH-PATH. Analyze the running time of your algorithm.

b. Extend your answer to part (a) so that if a path is returned, it is a most probable path starting at 0 and having label s. Analyze the running time of your algorithm.

2.2 Solution

Idea
The problem of finding if the sequence of sounds exist, can be solved in several ways, including Brute-Force Depth First Search (DFS) and Breadth First Search (BFS) method. Here, I approach the problem using Dynamic Programming technique. The overall idea is to reduce the problem of sequence k to a smaller problem of sequence k-1, and solve it recursively to get the final solution. We construct a table of order n*k where n is the number of nodes and k is the number of sounds present in the sequence. We fill the first column and use that to compute the subsequent columns.

The recursive relation for node i and sound j would be

$$S(i,0) = 0, for 1 <= i <= n-1$$

$$VS(k, j-1), otherwise$$

While calculating probability, $S(i,j) = p(vk, vi) * S(k, j-1)$

Pseudocode
Input: The graph G=(V,E) and the desired sound sequence S=σ_1,σ_2,σ_3. Also, probabilities of taking each path is given.

Output: The path which produces the sequence S=σ_1,σ_2,σ_3. Also, we need to find out the path such that it has the maximum probability

Pseudocode:

Part a:
Step 1: Construct a matrix of order n*k where n is the number of nodes and k is the number of sounds in the desired sequence. Set all values to 0s.

Step 2: Now, consider the first sound, σ_1. Suppose the edge connects 2 vertices i and j, then store V_i at position (V_j,σ_1) in the table. This will allow us to backtrack later, if necessary.

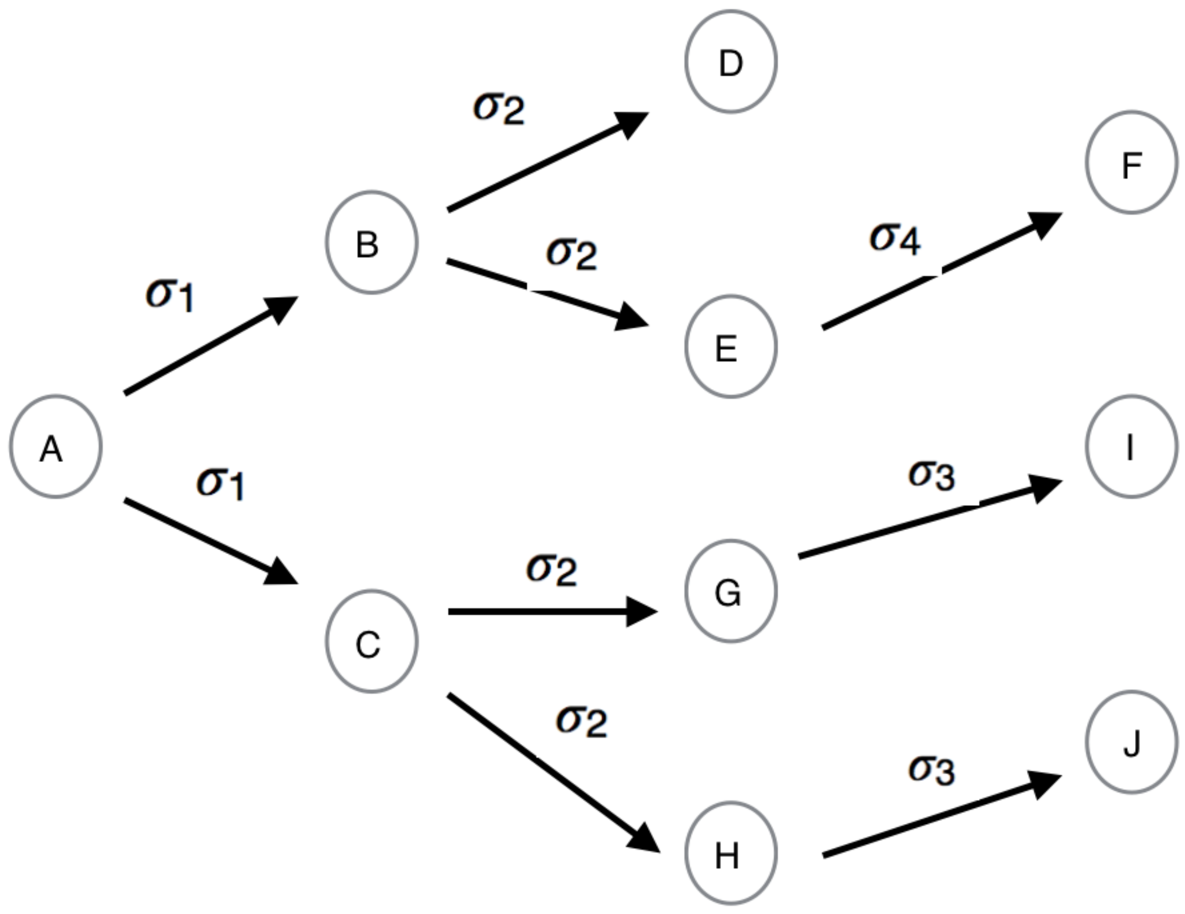
Step 3: In the subsequent steps, we use the values in the previous columns and look for edges that emanate from those vertices. Again, the starting vertex of the edge is stored in the corresponding location. This step is repeated till we complete 'k' steps.

Step 4: Traverse through the k^{th} column to check for non-zero entries. If non-zero entries exist, then backtrack to find the path and return the path and exit.

Step 5: Traverse through the k^{th} column to check for non-zero entries. If no non-zero entries exist, then return PATH-NOT-FOUND and exit.

Part b:

- To find the path with the maximum probability, calculate and store the probability value at each step.
- In the first column, the probability of the path has to be stored.
- As we progress through the columns, the probabilities of each new edge is multiplied to the existing value and stored in the appropriate location.
- To find the final path with the maximum probability, iterate through all the values in the k^{th} column to find the maximum value. This value corresponds to the desired path with the maximum probability.



Proof of Correctness
Let us consider the above graph with 'n' nodes labelled from A to J and the sound sequences as labelled in the figure. Let the required sequence be S=σ_1,σ_2,σ_3. We have matrix of order n*3. First, consider σ_1, we can see that σ_1 leads us to nodes B and C. Store the value 'A' in those respective positions. Next, repeat the same steps for σ_2. We see that σ_2 leads us to D and E from node B and to Nodes G and H from Node C. Enter the corresponding values in the table. Finally, consider σ_3. We reach I and J from G and H, via σ_3. Hence, add those values in the table. Finally, we see that there are two paths, A-C-G-I and A-C-H-J, which produce the required sequence of sounds. Suppose all entries in the last column were 0s, it implies that there exists no such path which produces the desired sound sequence.

Node	σ_1	σ_2	σ_3
A	0	0	0
B	A	0	0
C	A	0	0
D	0	B	0
E	0	B	0
F	0	0	0
G	0	C	0
H	0	C	0
I	0	0	G
J	0	0	H

To find the maximum probability path, instead of storing the node number in the table, we store the probabilities of that edge. As we proceed through the columns, we keep multiplying the probabilities of all the edges that make up the path. After the algorithms completes, we scan through the last column to find the maximum probability. The corresponding path is the path with the maximum probability.

Time Complexity

The time complexity of this algorithm, like any Dynamic Programming algorithm, can be computed as the product of the time taken to construct the table and the number of entries in the table. In this case, it would be O(nek), since we need to go through e edges to each cell. However, this unrealistically considers that the graph is completely connected, which is seldom the case. In reality, for each column we process, the edges we traverse in total for all vertices is just the edge set. In essence, we just take O(n+e) time for a single column and we only calculate for k values. So the time complexity is O(nk+ek).