Dynamic Programming Examples

COP 4531

December 6, 2014

Here are some dynamic programming problems, with solution sketches below. We suggest you try the problems without looking at the solutions, and check your answers afterwards. You may also want to try writing pseudocode for these questions.

Shortest k-Hop Path

Suppose that we are given a graph and two nodes in the graph. let the graph be $G = (V, E)$ and the two nodes be s and t. We want to find the shortest path between the two given nodes such that it has no more than k hops or edges on it. We note that this problem cannot be solved by Djikstra 's algorithm because it does not keep track of the number of hops on a shortest path.

Coin Change Problem

The following problem is very similar to the version of the Knapsack problem that we have discussed in problem 1.

Suppose that I have the following coins with me: 25 cents, 10 cents, 5 cents, 1 cent. I have to pay a certain sum to a friend and I want to use the smallest number of coins to make the payment.This is the coin change problem.

Now consider the general setting of the above problem where there are n denominations of coins namely $D = \{d_n > d_{n-1} > \cdots > d_1\}$ $d_1 = 1$. Suppose that all the denominations are in cents. Now suppose that I want to pay a

total of m cents. Design a dynamic programming algorithm that minimizes the number of coins in the change.

Knapsack Version 1

The first problem is the Knapsack Problem. The story goes as follows: A burglar breaks into a house and he has a bag, the knapsack, with him in which to carry off the loot. The knapsack can only carry a weight of W. In the house there are n items, namely $I_1, I_2, ..., I_n$ and the items have weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$. The burglar can either take an item or decide to leave it, he cannot break up an item into smaller parts. His goal is to carry the items that will maximize the value of his loot keeping in mind that he can only carry a total weight that does not exceed W.

This problem can be written as an optimization problem with a constraint on the weight. Let us write $X_i = 1$ if item i is selected by the burglar or $X_i = 0$ otherwise. Then the Knapsack problem can be written as follows:

$$Maximize \sum_{i=1}^{n} X_i v_i$$

Subject to

$$\sum_{i=1}^{n} X_i w_i \leq W$$

$$X_i \in \{0, 1\}$$

The above formulation of the KS problem as a constrained optimization problem is known as an Integer Programming formulation. This is NP-Hard and there is no possibility of getting a polynomial time algorithm for this problem.

There are two variations of the KS problem that we have stated, which incidentally is called the 0-1 Knapsack Problem. The first variation of this problem is where there are as many copies of each item as is needed. Thus one can select the first item as many times as needed, the second item as many times as needed and so on.

Knapsack Version 2

Now we consider the second version of the Knapsack problem. In this setup there is only one copy of each item and thus we cannot choose each item more than once in order to maximize our profit/value.

Edit Distance

Edit Distance is a way to find distance between words. For example whenever we are typing and the spellchecker is at work, it needs to suggest corrections for words that are misspelt. For this it has to find words that are close to the misspelt words and thus we need to define the idea of closeness of words. Edit Distance is one way of doing that. The edit distance between words $w1$ and $w2$ is the minimum number of inserts, deletes and substitutions that are needed to convert the first word to the second. Insert is the operation of inserting a new character, it can even be a space character. Deletion is the operation of removing a character and substitution is the operation of replacing one character with another. Another way to think about edit distance is to think about lining up the two words character by character so that the words are maximally aligned in that the maximum number of characters match. Then the number of places where the characters mismatch gives us the edit distance.

Solution Sketches

Shortest k-Hop Path

Let $d(v, i)$ = shortest path length between nodes s and v such that it has no more than i edges on it. The we can write:

$$d(v, i) = min_{(u,v)\in E}(d(u, i-1) + 1)$$

We would like to compute and return $d(t, k)$

Coin change solution

Let $C(m)$ = the optimal number of coins using the set D of possible coins and change due m of the given. Note that if O is the optimal solution and the coin with denomination d_i is included in the optimal solution then we have that $C(m) = 1 + C(m - d_i)$. The problem is that we do not know d_i because we do not know O. Note that the way we are writing this the coin d_i can be included multiple times in O. So what do we do: we want to minimize and hence the most logical way to go about would be to select the d_i that gives us a minimum for $C(m) = 1 + C(m - d_i)$. Thus we have the following:

$$C(m) = 0$$

if $m = 0$. Otherwise

$$C(m) = min_{\{d_i \in D\}}\{1 + C(m - d_i)\}$$

Now we are done.

Can you see the similarity with the recurrence for problem 1 albeit a small difference in the optimization goal?

Knapsack Version 1 solution

For this version of the problem the subproblems are the ones where the weight is less that W. Let us write $k(w)$ =optimal value of items in KS with weight constraint w. Let us suppose that item I_j is selected for getting this optimal $k(w)$. Now if we remove the item number j from the KS then the remaining items give the optimal value for the KS of weight constraint $w - w_j$. Thus we can write

$$k(w) = k(w - w_j) + v_j$$

The problem here is that we assume that we know the j that achieves the maximum, which is not true because in reality we don't know the value of j.As our goal is to maximize the profit/value in the KS we need to choose a j that maximizes the above. Thus we would choose a j such that we have a maximum value for the above and hence we can write

$$k(w) = max_{1 \leq j \leq n}\{k(w - w_j) + v_j\}$$

and as we are seeking the profit/value for the weight constraint W, we would return $k(W)$ as our final answer.

Knapsack Version 2 solution

In order to solve the second version of the KS problem we will have to choose the subproblems differently. Here as we have to use each item only once, if an item has been used in the solution of a subproblem then that same item cannot be used for the solution of another subproblem. Thus apart from the weight the optimal solution should also depend on the items already used for the solution. Let $k(w, j)$ be the optimal profit with a knapsack of weight w using the items $I_i, ..., I_j$.

Now we note that there are two ways in which we can get the optimal profit using items $I_i, ..., I_j$ namely either take the item I_j or leave it. If we plan to use the item I_j then

$$k(w, j) = k(w - w_j, j - 1) + v_j$$

On the other hand if item I_j is not used then the optimal profit is given by

$$k(w, j) = k(w, j - 1)$$

As we want to maximize the profit/value the burglar should examine both these cases and decide on taking or leaving item I_j based on which one of the above two is actually larger, that is

$$k(w, j) = max\{k(w - w_j, j - 1) + v_j, k(w, j - 1)\}$$

We are looking for the value of k(W,n) in the end.

Edit distance solution

Let us try to solve the problem of finding the edit distance between two strings using a dynamic programming algorithm. Suppose that the two strings are given by $x[1....m]$ and $y[1.....n]$. The first thing that one needs to do when trying to solve a problem using dynamic programming is to define suitable sub problems. Once this is done the hard part is over. In our case let $E(i, j)$ = the edit distance between the strings $x[1....i]$ and $y[1....j]$. We are interested in $E(m, n)$. In order to get the final solution we need to define a recursion that defines $E(i, j)$ in terms of smaller sub problems. Now note that when we try to align $x[1....i]$ and $y[1....j]$ we can get one of three cases:

1. $x[i]$ lines up with a space that is it aligns with no character thus contributing to the edit distance by 1

2. $y[j]$ lines up with a space that is it aligns with no character thus contributing to the edit distance by 1

3. $x[i]$ and $y[j]$ both line up together in which case they contribute 1 to the edit distance if $x[i] \neq y[j]$ and 0 if $x[i] = y[j]$

Using the above argument we get that

$$E(i,j) = min\{E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+[x[i]=y[j]]$$

where $[x[i] = y[j]] = 1$ if $x[i] \neq y[j]$ and 0 otherwise. We also note that $E(i,0) = i$ and $E(0,j) = j$