CS2010 Semester 1 2012/2013

Data Structures and Algorithms II

Tutorial 09 - Dynamic Programming 2

For Week 11 (29 October - 02 November 2012)

Released: Wednesday, October 24, 2012

Document is last modified on: October 31, 2012

1 Introduction and Objective

Quiz 1 and 2 are over by now. We are now left with the third and last part of CS2010 that will only be examined during final exam.

In the previous tutorial, we discuss Fibonacci and Coin Change. Both are DP problems involving only a single parameter to describe the state/subproblem. In this tutorial, we will increase the level of difficulty of DP problems. We will give you problems with longer story and you have to identify the correct state/subproblem. The number of parameters that are used to describe a state/subproblem is limited to maximum two.

Finally, we discuss PS7 Subtask 1-2, re-visiting lecture 10 materials if necessary.

2 Tutorial 09 Questions

Q1. Which of the following statement(s) is/are false:

1. SSSP can be solved on a general graph with -ve weighted cycles if we limit the total number of nodes traversed from source to destination.
2. In DP, other than getting the recurrence relations and the base cases, we still need to store the solutions to smaller sub-problems.
3. We can solve a DP problem without computing the topological sort of the underlying DAG first.

Ans: None of the above is false.
Reasons:
1). Already shown in Lecture 10 ('traveling salesman with m hops'),
2). Yes, if we only have recurrence + base cases, we only have recursive backtracking (can be slow). By storing the solutions of the smaller sub-problems, we ensure that each subproblem is only computed once. These are the two ingredients for DP to be useful,
3). Yes, we can use the top-down DP instead (recursion). The recursion will first check if the current state has been computed before or not and will not re-compute if it has been computed before. This way, we do not need to use the topological order of the underlying DAG (bottom-up DP).

Q2. In a futuristic society, country X is being invaded by country Y to its east. In order to defend against country Y, country X has set up a defense system along its eastern border. It has basically converted its entire eastern border into a grid like system, where any place not on the grid is blocked by a force field which cannot be crossed. Thus the enemy can only invade country X by using the grid lines.

However, a battery of lasers of different types and with different fire power are placed at intersections points on the grid, and will fire upon the enemy when they are detected. The lasers can only fire in at most 3 direction - east, north and south. Note that adjacent lasers cannot both fire towards the other due to uncontrolled "chaining" effect (to be described shortly) that could destroy both lasers. Due to budget constraints not all intersection points in the grid are placed with a laser.

An illustration of the grid and laser system is given below:

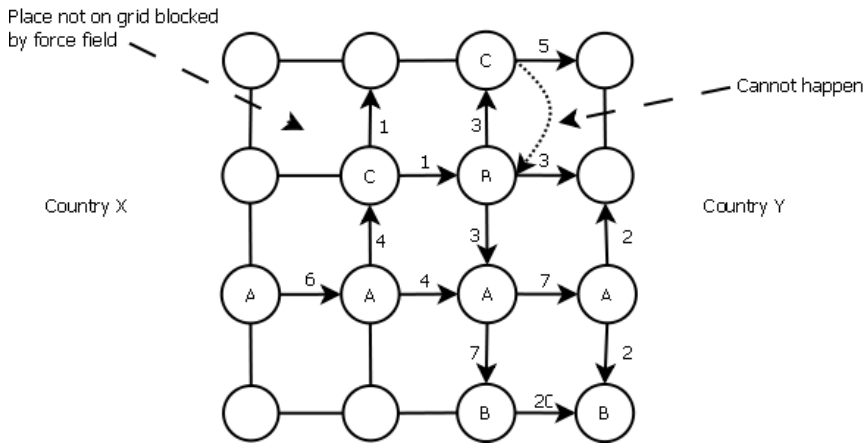

Figure 1:

Vertices with no labels are intersection points that are empty. The label on a labeled vertex tells the type of the laser. The directions a laser can fire are indicated by the directed edges. The number on the directed edge indicates the fire power from the laser.

Sensors attached to each laser allow a base radar station performing a periodic sweep of the whole grid to detect if enemies are currently occupying an intersection point where a laser is located. Once an enemy is detected, it is too late for the laser lying on the intersection point occupied by the enemy to fire. However a series of lasers can be **chained** along the grid to fire upon the occupied intersection point starting from a source laser. The final fire power unleashed on the enemy is the sum of the edges along the path from the source laser to the destination laser. An illustration is given below (suppose the enemy is detected at bottommost rightmost cell):

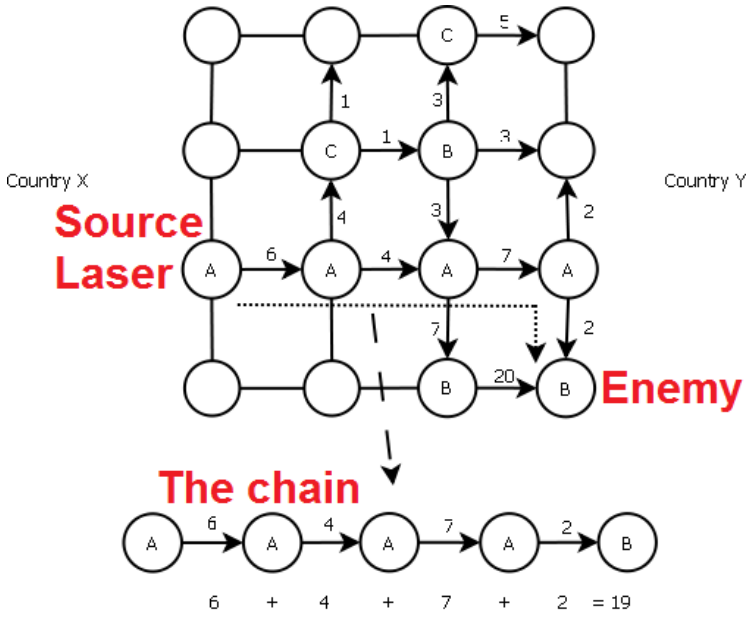

Figure 2:

After setting up this system, the scientists now need an algorithm to find the source laser and the best chain of lasers to fire when given enemy location, such that the fire power reaching the enemy location is maximized. Help the scientists design such an algorithm. You can assume that there is always at least 1 possible chain of lasers that can reach the enemy location.

Ans:
Actually, this long problem is just about finding the longest path on a DAG. We have discussed the bottom-up/top-down DP solution to find this longest path on a DAG during Lecture 09. For this one, we can set the source as the position of the given enemy, reverse the direction of all edges (the graph is still a DAG), and run longest path on this reversed DAG. For Figure 2, the longest path is actually: 6+4+1+3+7+20 = 41, which is this path: A (go right) → A (go up) → C (go right) → B (go down) → A (go down) → B (go right) → B (enemy)).

Q3. A subset of nodes S ⊂ V is an independent set of graph $G = (V, E)$ if there are no edges between them. For instance, in the example below, nodes {1,5} form an independent set, but nodes {1,4,5} do not. The largest independent set are {2,3,6}, {1,4,6}, {2,4,6}, or {1,3,5}, all with size 3. Give a DP solution for finding the size of the largest independent set of G when G is a tree. Hint: Do you need to attach an extra parameter to each vertex?

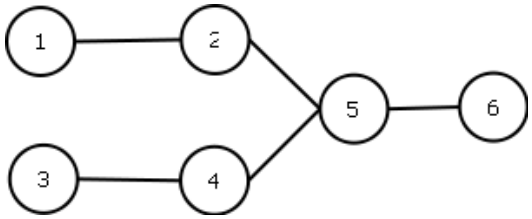

Figure 3:

Ans: Here we have a choice to pick or not pick a vertex. Also once a vertex is picked, we cannot pick its children (otherwise the set will not be independent). We can formulate the recurrence as follows:

$I(v, 0)$ is the largest independent set for the tree rooted at vertex v when v is not taken, while $I(v, 1)$ is the case when vertex v is taken.

If v is not taken, we can choose to take or not take its children and we will take the max among the two options, thus:
$I(v, 0) = \sum Max(I(c, 0), I(c, 1))$, for all children c of v.

If v is taken, we definitely cannot choose its children without violating the independent set requirement, thus:
$I(v, 1) = 1 + \sum I(c, 0)$, for all children c of v. (notice +1 because we take vertex v).

Base cases:
$I(v, 0) = 0$ for all v, where v are leaves in G.
$I(v, 1) = 1$ for all v, where v are leaves in G.

The pseudo code is now simple if we have these recurrences, try to come up with it by yourself. We need $2 \times V$ memo table because we duplicate each vertex of the tree inside this computation DAG. The total number of operations needed is $O(3 \times E) = O(3 \times (V - 1)) = O(V)$ as the graph is a tree.

Problem Set 7

Q4. Discussion of PS7 Subtask 1-2 (recap of Lecture10)

For Subtask 1, the supermarket is a small supermarket and everything there have to be grabbed/bought. $1 \leq K = N \leq 9$. Therefore, brute force (backtracking) is still possible as $(9 + 1)! = 10!$ is doable. In Lecture 10, Steven has shown how to do recursive backtracking by resetting the visited flag inside DFS routine. Tutor will discuss this again if students are still not clear with this.

For Subtask 2, the supermarket is slightly bigger than in Subtask 1 and everything there have to be grabbed/bought. $1 \leq K = N \leq 15$. This is the typical range of DP-TSP as outlined in Lecture 10. Backtracking is now too slow. Students just need to understand the lecture 10 material and applied it here. Tutor will discuss this again if students are still not clear with this, especially the bit manipulation part.

If still have time, a very general ideas (involving shortest paths) on how to deal with the more difficult Subtask 3-4-5 can be discussed.