Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 7 - Dynamic Programming

1. Consider the weighted directed graph $G = (V, E)$ of Figure 1. Run the Bellman-Ford algorithm to compute the value of $M[i, v]$ for every node $x \in V$. Recall that $M[i, v]$ is the cost of the minimum-cost $v \sim t$ path that uses at most i edges.

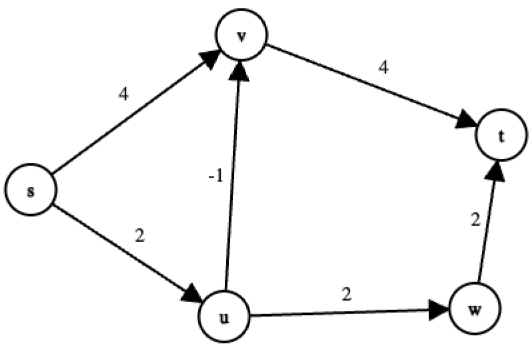


Figure 1: A directed graph with edge costs indicated. Algorithms Iluminated Example 18.2.6.

SOLUTION: We run the algorithm referring to the pseudocode presented in Lecture 20. In the initialisation step, we have $M[0, t] = 0$ and $M[0, x] = \infty$ for every other node $x \in V \setminus \{t\}$. Then we enter the nested for loops.

- Let $i = 1$. We consider the nodes one by one and compute $M[1, x]$ for each one of them. For this, we use the following recurrence relation

$$M[1, x] = \min\{M[0, x] + \min_{y \in N(x)} c_{xy} + M[0, y]\},$$

where $N(x)$ is the set of nodes that are reachable from x via a single edge. In our calculation, we observe that $M[0, x] = \infty$ for every node $x \in V \setminus \{t\}$. Therefore the nodes for which $M[1, x] \neq \infty$ (besides t) will be nodes v and w, which can reach t via a single edge. In other words, we have

$$M[1, s] = M[1, u] = \infty, M[1, t] = 0$$

For $M[1,v]$ we calculate the minimum of the expression as $M[1,v] = c_{vt} + M[0,t] = 4$. Similarly, for $M[1,w]$ we calculate $M[1,w] = c_{wt} + M[0,t] = 2$.

- Let $i = 2$. We consider the nodes one by one and compute $M[2,x]$ for each one of them. For this, we use the following recurrence relation

$$M[2,x] = \min\{M[1,x] + \min_{y \in N(x)} c_{xy} + M[1,y]\}.$$

We calculate

$$\begin{aligned}
M[2,s] &= \min\{M[1,s] + \min_{y \in N(s)} c_{sy} + M[1,y]\} = c_{sv} + M[1,v] = 4 + 4 = 8 \\
M[2,v] &= \min\{M[1,v] + \min_{y \in N(v)} c_{vy} + M[1,y]\} = M[1,v] = 4 \\
M[2,u] &= \min\{M[1,u] + \min_{y \in N(u)} c_{uy} + M[1,y]\} = c_{uv} + M[1,v] = -1 + 4 = 3 \\
M[2,w] &= \min\{M[1,w] + \min_{y \in N(w)} c_{wy} + M[1,y]\} = M[1,w] = 2 \\
M[2,t] &= 0
\end{aligned}$$

- Let $i = 3$. We consider the nodes one by one and compute $M[3,x]$ for each one of them. For this, we use the following recurrence relation

$$M[3,x] = \min\{M[2,x] + \min_{y \in N(x)} c_{xy} + M[2,y]\}.$$

Similarly to above we calculate

$$\begin{aligned}
M[3,s] &= \min\{M[2,s] + \min_{y \in N(s)} c_{sy} + M[2,y]\} = c_{su} + M[1,u] = 2 + 3 = 5 \\
M[3,v] &= \min\{M[2,v] + \min_{y \in N(v)} c_{vy} + M[2,y]\} = M[2,v] = 4 \\
M[3,u] &= \min\{M[2,u] + \min_{y \in N(u)} c_{uy} + M[2,y]\} = M[2,u] = 3 \\
M[3,w] &= \min\{M[2,w] + \min_{y \in N(w)} c_{wy} + M[2,y]\} = M[2,w] = 2 \\
M[3,t] &= 0
\end{aligned}$$

- Let $i = 4$. We consider the nodes one by one and compute $M[3,x]$ for each one of them. For this, we use the following recurrence relation

$$M[4,x] = \min\{M[3,x] + \min_{y \in N(x)} c_{xy} + M[3,y]\}.$$

It is not hard to see that in this case we will have $M[4,x] = M[3,x]$ for all $x \in V$. In the end, the $2D$ array M looks as follows.

	0	1	2	3	4
s	∞	∞	8	5	5
u	∞	∞	3	3	3
v	∞	4	4	4	4
w	∞	2	2	2	2
t	0	0	0	0	0

2. Assume that we wanted to use the Bellman-Ford algorithm to find the cost of the minimum-cost paths from a node s to all the nodes $x \in V$ in the graph G. Think about how to modify the algorithm to achieve this and run the modified algorithm on the graph of Figure 1 to compute the costs of all the minimum-cost paths from s to the nodes in V.

   Solution: The modification that we need to make is that now $M[i, x]$ will denote the cost of the minimum-cost path from node s to node $x \in V$ that uses at most i edges. In our initialisation step, we will have $M[0, s] = 0$ and $M[0, x] = \infty$ for every node $x \in V \setminus \{s\}$. Our recurrence relation will still be

$$M[i, x] = \min\{M[i-1, x] + \min_{y \in N^-(x)} c_{yx} + M[i-1, y]\},$$

   where now $N^-(x)$ denotes the set of nodes y for which there is an edge $(y, x) \in E$.

   If we run the algorithm on the graph of Figure 1, we get the following $2D$ array M:

	0	1	2	3	4
s	0	0	0	0	0
u	∞	2	2	2	2
v	∞	4	1	1	1
w	∞	∞	4	4	4
t	∞	∞	8	5	5

3. Consider the knapsack problem given by the following table, with capacity $W = 7$.

Item	Value	Weight
1	1	1
2	2	3
3	3	2
4	4	5
5	5	5

   Use the dynamic programming algorithm presented in the lectures to compute the value of the optimal solution.

   Solution: Recall that we will use a $2D$ array for which the entry $M[i, w]$ will correspond to the optimal solution using the first i intervals and capacity w. For the capacities w, we will consider all of the integers that are at most $W = 7$, i.e., $\{0, 1, 2, 3, 4, 5, 6, 7\}$.

   Our algorithm first sets $M[0, w] = 0$ for all $w \in \{1, \ldots, 7\}$. This results in our partially filled $2D$ arrray looking like:

   Then we run the first outer loop, for $i = 1$, and calculate the value of $M[1, w]$ for every $w \in \{1, \ldots 7\}$. For that, we use the recurrence relation:

$$M[1, w] = \max\{M[0, w], v_i + M[0, w - w_i]\},$$

   if $w_i > w$ (i.e., the item fits), otherwise we set $M[1, w] = M[0, w]$. Let's run this iteration explicitly:

5								
4								
3								
2								
1								
0	0	0	0	0	0	0	0	0
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

$M[1,0] = M[0,0]$ since $w_1 > 0$. When considering $w = 1$, the item now fits, as $w_1 = 1$. Therefore we have to check the recurrence relation. We have that $M[0,1] = M[0,0] = 0$, so the maximum is given by the second term and is equal to $v_i = 1$. Therefore we have $M[1,1] = 1$. Similarly, for $M[1,2]$ we have that $M[0,2] = M[0,1] = 0$, and the maximum is given again by the second term. We again have $M[1,2] = 1$. Similarly, we calculate $M[1,x] = 1$ for all $x \in \{1,\ldots 7\}$. Using this, we can populate our $2D$ array as follows:

5								
4								
3								
2								
1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Now let's consider the next iteration, when $i = 2$. We calculate the value of $M[2,w]$ for every $w \in \{1,\ldots 7\}$. For that, we use the recurrence relation:

$$M[2,w] = \max\{M[1,w], v_i + M[1, w - w_i]\},$$

if $w_i > w$ (i.e., the item fits), otherwise we set $M[2,w] = M[1,w]$. Let's also run this iteration explicitly: For $w = 0$, $w = 1$ or $w = 2$, we see that $3 = w_2 > w$, so for those cases we will have $M[2,x] = M[1,x]$ for $x \in \{0,1,2\}$. This means that $M[2,0] = 0$ and $M[2,1] = M[2,2] = 1$. For $w = 3$, we now have $w_2 = w$, and we can use the recurrence relation. We have that $M[1,3] = 1$ and $v_2 + M[1,0] = 2+0$, so we will have $M[2,3] = 2$. For $w = 4$, we have $M[1,4] = 1$ and $v_2 + M[1,1] = 2+1 = 3$, so we have $M[2,4] = 3$. Continuing like this, we can compute $M[2,x] = 3$ for all $x \in \{5,6,7\}$. Using this, we can populate our $2D$ array as follows:

5								
4								
3								
2	0	1	1	2	3	3	3	3
1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Continuing like this, in the end we complete our $2D$ array:

5	0	1	3	4	4	5	6	8
4	0	1	3	4	4	5	6	7
3	0	1	3	4	4	5	6	6
2	0	1	1	2	3	3	3	3
1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

4. Recall the following simple context-free grammar for arithmetic expressions from Lecture 21. The start symbol is Exp.

$$
\begin{array}{rcl}
\mathsf{Exp} & \to & \mathsf{Var} \;\mid\; \mathsf{Num} \;\mid\; (\;\mathsf{Exp}\;) \\
\mathsf{Exp} & \to & \mathsf{Exp} \;+\; \mathsf{Exp} \\
\mathsf{Exp} & \to & \mathsf{Exp} \;*\; \mathsf{Exp} \\
\mathsf{Var} & \to & x \;\mid\; y \;\mid\; z \\
\mathsf{Num} & \to & 0 \;\mid\; \cdots \;\mid\; 9
\end{array}
$$

   (a) How many syntax trees are there for each of the following three strings? Draw them all.

$$3 + x * y \qquad\qquad 3 + (x * y) \qquad\qquad z + 10$$

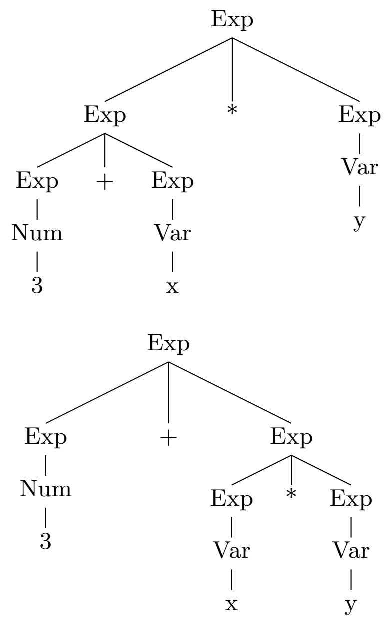
   (b) Design a new context-free grammar that generates exactly the same language as the one above, but with the property that it is *unambiguous*: every string in the language should have exactly one syntax tree. Informally, your grammar should enforce the familiar convention that * takes precedence over +. You will find it helpful to introduce some additional non-terminal symbols.

   [Hint: First try to do this for the grammar with the rule for $\mathsf{Exp} * \mathsf{Exp}$ omitted. To ensure that a string like $3 + 4 + 5$ has only one tree, you might want to draw inspiration from the grammar for comma-separated lists in Lecture 21. Then try to adapt your grammar to cater for *, building in the precedence rule.]

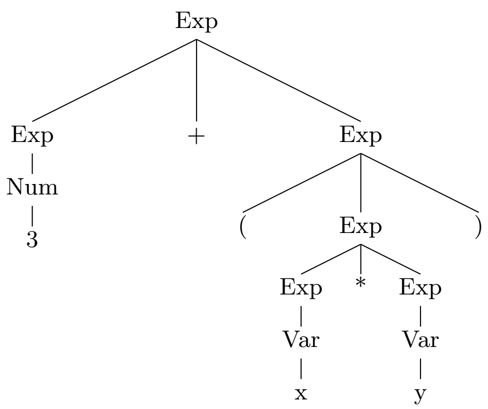
   (c) For the grammar you have designed in part (b), draw the *unique* syntax tree for any of the strings from part (a) that had more than one syntax tree with respect to the original grammar.

SOLUTION:

1. (a) 3 + x * y has two trees:



3 + (x * y) has just one tree:



z + 10 has no trees. This is not a sentence of the language: our grammar doesn't cater for multi-digit numerals like 10.

(b) First for the grammar with the clause for $*$ omitted: The key observation is that a general expression is a list of one or more 'simple expressions', separated by +. Drawing inspiration from the comma list example, the following grammar does the trick:

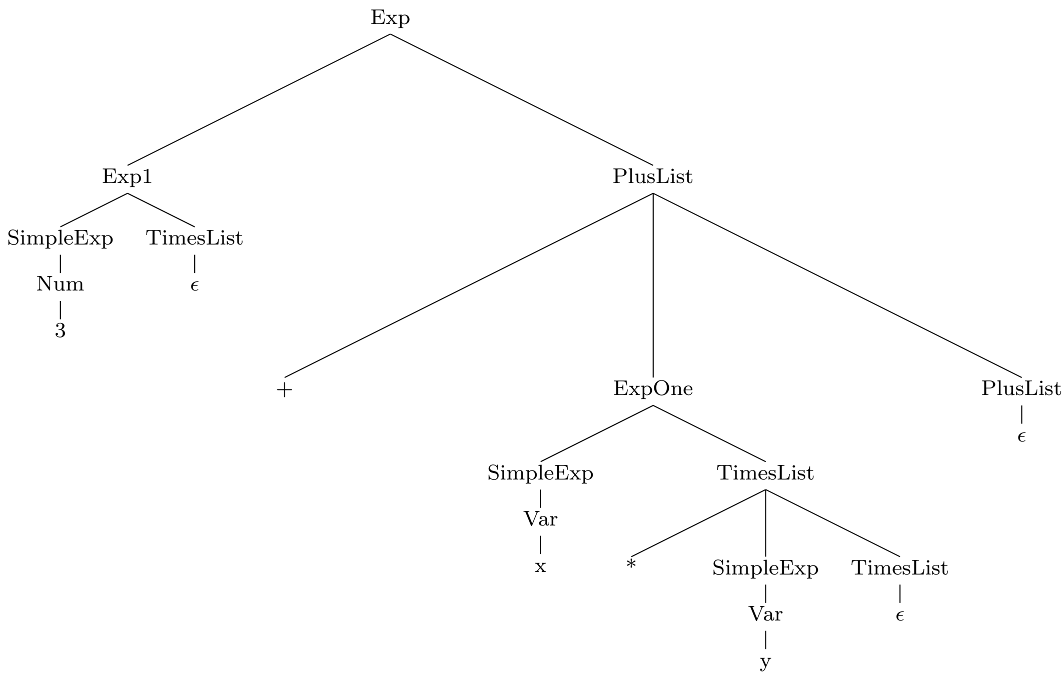
$$
\begin{array}{rcl}
\mathsf{Exp} & \to & \mathsf{SimpleExp\ PlusList} \\
\mathsf{SimpleExp} & \to & \mathsf{Var} \ | \ \mathsf{Num} \ | \ (\ \mathsf{Exp}\ ) \\
\mathsf{PlusList} & \to & \epsilon \ | \ + \ \mathsf{SimpleExp\ PlusList}
\end{array}
$$

(with the same rules for Var and Num as before). Intuitively, we here distinguish two 'levels' of expressions, corresponding to Exp and SimpleExp. To cater for $*$ as well, and to enforce the precedence rule, we can extend this idea to allow three levels:

$$\begin{array}{rcl} \mathsf{Exp} & \to & \mathsf{Exp1\ PlusList} \\ \mathsf{Exp1} & \to & \mathsf{SimpleExp\ TimesList} \\ \mathsf{SimpleExp} & \to & \mathsf{Var} \ \mid\ \mathsf{Num} \ \mid\ \mathsf{(\ Exp\ )} \\ \mathsf{TimesList} & \to & \epsilon \ \mid\ * \ \mathsf{SimpleExp\ TimesList} \\ \mathsf{PlusList} & \to & \epsilon \ \mid\ + \ \mathsf{Exp1\ PlusList} \end{array}$$

(plus the usual Var and Num rules). Other solutions are possible, but the above grammar turns out to be particularly well-adapted to 'left-to-right parsing'.)

(c) The unique syntax tree for 3 + x * y is now:



Here we include explicit ϵ's for clarity, though of course they contribute nothing to the string in question.