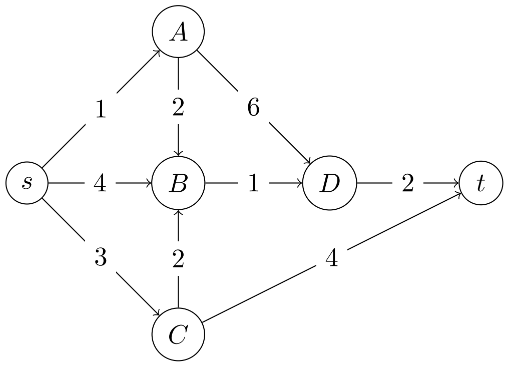
Assignment 6 solutions

Responsible TA: Claudi Lleyda Moltó

Task 1 - Bellman–Ford algorithm

Consider the following directed weighted graph



Find the shortest path from node s to node t using the Bellman–Ford algorithm. Show the step-by-step process, including the values of the distance labels at each iteration.

Solution. The Bellman–Ford algorithm will store, for every vertex v, its distance d to s and its predecessor p. We represent this with the following table, with the initialization values set.

v	s	A	B	C	D	t
d	0	∞	∞	∞	∞	∞
p	NULL	NULL	NULL	NULL	NULL	NULL

Note how there are no negative edges in the graph. Therefore there are no negative cycles and we do not need to check for them.

We begin the main body of the algorithm. When iterating over several edges, we will follow the lexical order of their vertices.

1. We take the edge (s, A) with weight $w(s, A) = 1$.

   Then $d(s) + w(s, A) = 1 < \infty = d(A)$, so we set $d(A) = d(s) + w(s, A) = 1$ and $p(A) = s$.

v	s	A	B	C	D	t
d	0	1	∞	∞	∞	∞
p	NULL	s	NULL	NULL	NULL	NULL

2. We take the edge (s, B) with weight $w(s, B) = 4$.

   Then $d(s) + w(s, B) = 4 < \infty = d(B)$, so we set $d(B) = d(s) + w(s, B) = 4$ and $p(B) = s$.

v	s	A	B	C	D	t
d	0	1	4	∞	∞	∞
p	NULL	s	s	NULL	NULL	NULL

3. We take the edge (s, C) with weight $w(s, C) = 4$.

   Then $d(s) + w(s, C) = 3 < \infty = d(C)$, so we set $d(C) = d(s) + w(s, C) = 3$ and $p(C) = s$.

v	s	A	B	C	D	t
d	0	1	4	3	∞	∞
p	NULL	s	s	s	NULL	NULL

4. We take the edge (A, B) with weight $w(A, B) = 4$.

   Then $d(A)+w(A, B) = 3 < 4 = d(B)$, so we set $d(B) = d(A)+w(A, B) = 3$ and $p(B) = A$.

v	s	A	B	C	D	t
d	0	1	3	3	∞	∞
p	NULL	s	A	s	NULL	NULL

5. We take the edge (A, D) with weight $w(A, D) = 6$.

   Then $d(A)+w(A, D) = 7 < \infty = d(D)$, so we set $d(D) = d(A)+w(A, D) = 7$ and $p(D) = A$.

v	s	A	B	C	D	t
d	0	1	3	3	7	∞
p	NULL	s	A	s	A	NULL

6. We take the edge (B, D) with weight $w(B, D) = 1$.

   Then $d(B)+w(B, D) = 4 < 7 = d(D)$, so we set $d(D) = d(B)+w(B, D) = 4$ and $p(D) = B$.

v	s	A	B	C	D	t
d	0	1	3	3	4	∞
p	NULL	s	A	s	B	NULL

7. We take the edge (C, B) with weight $w(C, B) = 2$.

   Then $d(C) + w(C, D) = 5 \geq 3 = d(C)$, so no update is made.

8. We take the edge (C, t) with weight $w(C, t) = 4$.

   Then $d(C) + w(C, t) = 7 < \infty = d(t)$, so we set $d(t) = d(C) + w(C, t) = 7$ and $p(t) = C$.

v	s	A	B	C	D	t
d	0	1	3	3	4	7
p	NULL	s	A	s	B	C

9. We take the edge (D, t) with weight $w(D, t) = 4$.

   Then $d(D) + w(D, t) = 6 < \infty = d(t)$, so we set $d(t) = d(D) + w(D, t) = 6$ and $p(t) = D$.

v	s	A	B	C	D	t
d	0	1	3	3	4	6
p	`NULL`	s	A	s	B	D

There are more iterations to be performed according to the algorithm, but the values in the table will not be relaxed further. Therefore, the algorithm has converged to the final solution, and we find that the resulting shortest path from s to t is $s \to A \to B \to D \to t$ with weight 6. $\Diamond$

Task 2 - Coin combinations

Given the currency denominations for Norwegian krone coins $C = \{1, 5, 10, 20\}$, design a dynamic programming algorithm to count in how many ways can an amount be obtained.

For example, to make 6 kroner there are two options: $1+5$ and $1+1+1+1+1+1$.

Solution. Let $n(K)$ be, for any integer K, the solution to the problem.

Note how we can solve the problem in a recursive manner. Indeed, for every positive K we have

$$n(K) = \sum_{c \in C} n(K - c),$$

where for any negative K we take $n(K) = 0$, and $n(0) = 1$.

A naïve implementation of this recursive algorithm would incur in a lot of recalculation. We propose the following algorithm, encoding the stated recursion in a dynamic programming pattern.

```
function CoinCombinations(Integer target)
    $table \leftarrow$ NewTable($target + 1$)
    $table[0] = 1$
    for $coin$ in $C$ do
        for $index$ in Range($coin, target$) do
            $table[index] \leftarrow tabe[index] + table[index - coin]$
    return $table[target]$
```

$\Diamond$

Task 3 - Grid traveling

Given an $n \times m$ grid, design an algorithm to calculate how many ways there are to travel from the top-left cell to the bottom-right cell, given you can only move right or down from one cell to another.

Solution. When traversing the grid we have, at every cell, at most two options. The number of possible paths from the cell to the bottom-right cell is equal to the sum of possible paths from the cell below and the cell to the right, if there are any.

We can calculate this with the following recursive expression.

$$n(i,j) = n(i-1,j) + n(i,j-1),$$

where $n(i,j) = 0$ if $i < 0$ or $j < 0$, and $n(0,0) = 1$.

We can write this as a dynamic programming algorithm:

```
function PATHS(Integer target, N, M)
    for $i$ in $0 \ldots N$ do
        $paths[i,0] \leftarrow 1$
    for $j$ in $0 \ldots M$ do
        $paths[0,j] \leftarrow 1$
    for $i$ in $1 \ldots N$ do
        for $j$ in $1 \ldots M$ do
            $paths[i,j] \leftarrow paths[i-1,j] + paths[i,j-1]$
    return $paths[N-1, M-1]$
```

As a note, we can also solve this problem in a combinatorial way. Indeed, any solution will involve n movements down and m movements to the right, and these can be performed in any order. We are therefore looking at how many ways we can arrange the n downwards movements amongst the $n+m$ total movements. This is exactly $\binom{n+m}{n}$. $\Diamond$