1. Compute the value of f(6,7) by the DP method using only integer 2D table of size 7×8.

a) **f(x,y) =**

$$\begin{cases} \mathbf{0} & \text{if } x=0 \text{ or } y=0 \\ \mathbf{f(x-1,\,y-1)+ f(x-1,\,y) + f(x,\,y-1) + 1} & \text{else} \end{cases}$$

Hint/solution Create a 2D array P of size 7×8. Fill first (index 0) row and column with zeros. In the order of nondecreasing values of indices x and y fill consequent cells in the array using the formula
P[x][y] = P[x-1][y-1] + P[x-1][y] + P[x][y-1] +1
The cell P[6][7] contains value f(6, 7) = 9912.

b) **f(x,y) =**

$$\begin{cases} \mathbf{0} & \text{if } x=0 \text{ or } y=0 \\ \mathbf{\max\{ f(x-1,\,y-1),\ f(x-1,\,y)\} + f(x,\,y-1) + 1} & \text{else} \end{cases}$$

Hint/solution Analogous to 1 a), the only change is the formula
P[x][y] = max(P[x-1][y-1], P[x-1][y]) + P[x][y-1] +1
The cell P[6][7] contains value f(6, 7) = 1715 .

2. 1. Compute the value of f(3, 4, 2) by the DP method using only integer 3D table of size 4×5×3.

a) **f(x,y,z) =**

$$\begin{cases} \mathbf{0} & \text{if } x=0 \text{ or } y=0 \text{ or } z=0 \\ \mathbf{f(x-1,\,y-1,\,z-1)+ f(x-1,\,y,\,z) + f(x,\,y-1,\,z) + f(x,\,y,\,z-1)+1} & \text{else} \end{cases}$$

Hint/solution Create a 3D array P of size 4×5×3. Fill with zeros all cells which have at least one index equal to 0. In the order of nondecreasing values of indices x and y and z fill consequent cells in the array using the formula
P[x][y][z] = P[x-1][y-1][z-1] + P[x-1][y][z] + P[x][y-1][z] + P[x][y][z-1] + 1
The cell P[3][4][2] contains value f(3, 4, 2) = 215.

b) **f(x,y) =**

$$\begin{cases} \mathbf{0} & \text{if } x=0 \text{ or } y=0 \text{ or } z=0 \\ \mathbf{3*f(x-1,\,y-1,\,z) - f(x,\,y-1,\,z) - f(x-1,\,y,\,z-1) + 1;} & \text{else} \end{cases}$$

Hint/solution Analogous to 2 a), the only change is the formula
P[x][y][z] = 3*P[x-1][y-1][z] - P[x][y-1][z] + P[x-1][y][z-1] + 1
The cell P[3][4][2] contains value f(3, 4, 2) = 10.

3. Pascal triangle:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
  1   7  21  35  35  21   7   1
.. .. .. .. .. .. .. .. .. .. ..
```

Pascal triangle contains binomial coefficients Bin(n,k). Recall that for positive values of n and k the recursive relation always holds:
Bin(n,k) = Bin(n-1,k) + Bin(n-1,k-1)
Now, define a simple recursive function which returns the binomial coefficient

$$\begin{cases} \mathbf{1} & \text{if } k = 0 \text{ or } n = k \end{cases}$$

Bin(n,k) =

Bin(n-1, k) + Bin(n-1, k-1) if n > k > 0.

Find out, how many times is this function called when command **x = Bin(6,4);** is issued.

Hint/solution If n = 0 or n = k then the function Bin returns 1 and terminates the recursive descent. When both parameters are positive and not equal then the total number of calls of function Bin(.,.) during execution of the call Bin(n,k) is equal to the number of calls during execution of Bin(n-1,k) plus the number of calls during execution of Bin(n-1, k-1) plus one because we have to count the call Bin(n,k) as well.
Denote by symbol NC(n, k) the total number of calls of Bin(.,.) during execution of the call Bin(n,k).
We arrive to the recurrence:
NC(n, k) = 1 for k = 0 or n = k,
NC(n, k) = NC(n-1, k) + NC(n-1, k-1) +1 for n > k > 0.
This recurrence differs form the binomial coefficient recurrence only by additional 1, therefore we can utilise the same triangle scheme which we fill with values of NC:

```
                1
              1   1
            1   3   1
          1   5   5   1
        1   7  11   7   1
      1   9  19  19   9   1
    1  11  29  39  29  11   1
1  13  41  69  69  41  13   1
```

We conclude PV(6, 4) = 29.
Note that studying closely both Bin a nd NC tables we arrive to hypothesis NC(n, k) = 2*Bin(n, k)-1.
Prove it by induction!

4. Ackermann function A(n, m) (see bellow) is defined for any two non-negative integer parameters and its values can be stored in a 2D table. Fill first few lines and columns of this table by the corresponding values A(n, m) using the DP approach and avoid thus any recursive calls.
Can you determine the value of A(4,4) or maybe of A(5, 3)?

A(n, m) =
m+1 n=0
A(n-1, 1) n>0, m=0
A(n-1, A(n, m-1)) n>0, m>0

(Note that the second function parameter in the third line is again a result of function call A(n, m-1).)

Hint/solution Suppose that n is a row index. Let the table be potentialy infinite in both dimensions with for positive n and m. The definition of A(n, m) consists of three cases, we present the corresponding methods for each case bellow:
1. Fill the 0-th row with values 1, 2, 3, ...

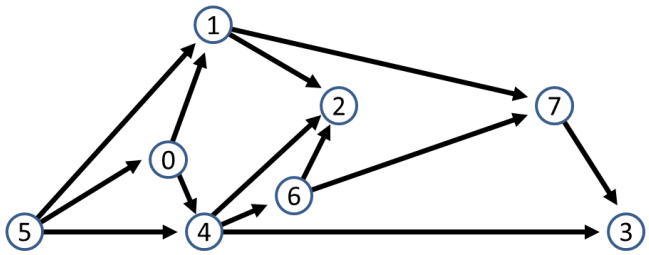
2. The first value (column 0) in each row is equal to the second value (column 1) in the immediately previous row.

3. The value Y in each next cell in the row with index n≥1 is equal to the the value X in the previous row (row n-1) and in some column. Which column? The column which index is equla to the value written in the cell immediately to the left of Y, i.e. cell in row n and column m-1.

The value A(4,4) can be relatively easily determined to be 2^(2^(2^(2^(2^(2^2))))-3
= 2^(2^(2^65536)) - 3 (do not try to evaluate it exactly numerically). But for example, another "close" value A(5,3) is, somehow strangely out of reach....

5A. Determine some topological order of nodes of DAG G1. The nodes of G1 are labeled 0,1, 2, ..., 7 and the list of edges of G1 is given:
E(G1) = {(0, 1), (0, 4), (1, 2), (1, 7), (4, 2), (4, 3), (4, 6), (5, 0), (5, 1), (5, 4), (6, 2), (6, 7), (7, 3)}.

Hint/solution Each topological order can be easily deduced form the picture. There are exactly 9 of them. Each topological order respects the order of nodes on the longest path 5 -- 0 -- 4 -- 6 -- 7 -- 3. Imagine this longest path to be a kind of "backbone" and then there remain nodes 1 and 2 to be fitted in some appropriate places in it.



5B. There exist more different topologicals orders of nodes of G1 in 5A problem. When we print out labels of nodes in some order the result is formally an 8D vector of integers. Find such topological order of G1 in which the corresponding vector of node labels is lexicographically as small as possible.

Hint/solution The picture helps us to determine the order 5 0 1 4 6 2 7 3.

6. Suppose that each edge in connecte DAG has some real valued weight. There is a standard (see lecture notes) DP algorithm which finds the path with maximum weight in this DAG. Modify this algorithm in such way that it solves modified problems described bellow:

A. Supose that each node has positive weight and the edges have no weights.
B. Supose that each node has arbitrary weight (positive or negative) and the edges have no weights.
C. Supose that each node and each edge has arbitrary weight.

Hint/solution
A 1. Convert the a problem to the old and solved one. Add one artificial leaf and add an edge from each original leaf to the artificial one. Now write the value of each original node on each outgoing edge from that node. This modification results in a standard edge-weighted DAG and the standard algorithm of finding longest paths can be applied to it. The longest path in the resulting DAG corresponds to the longest path in the original graph. If the longest path in the new DAG contains an added edge, just remove that edge from the path, otherwise no action is needed.
2. Direct solution. The algorithm is analogous to the longest path DP algorithm. In each node x we will keep value D(x) equal to the length of the longest path which ends in x. When we come to x we will examine all direct predecessors of x (connected by an edge to x) and choose the one which has maximu value of D. Let it be node y. Then we set D(x) = D(y) + weight(x) and move to the next node in the topological order.
The system of predecessors and final path reconstruction remains the same as it is in the edge-weighted DAG.

We do not have to include the compensation condition for the negative weights. The condition says that if D(x) is negative then it is set to D(x) = 0 to avoid the negative influence of the negative-weighted edges which precede x on any potential longest path.

B. Completely analogous to A, only the compensation condition must be included now.

C. 1. Analogous to A1 with the only difference that in the initial phase we *add* each node weigth to all weights of its outgoing edges.
2. Analogous to A2 with the only difference: D(x) = D(y) + weight((y, x)) + weight(x) and y is chosen so that the term D(y) + weight((y, x)) is maximum possible.

7. Describe a method based on DP which finds the number of all paths in unweighted DAG which length is exactly 3. Do not generate or traverse separately each of those paths.
(Hint: Register in each node how many paths of length 1, of length 2and of length 3 end in that node.)

Hint/solution Denote by symbols P(x,0) resp. P(x, 1) resp. P(x, 2), resp P(x, 3) the number of paths which end in node x and which length is exactly 0, resp. 1, resp 2, resp 3.
Set P(x, 0) = 1 for each node x. (Path of length 0 starts and ends in x, but it nonetheless exists.)
Initialize P(x, 1) = P(x, 2) = P(x, 3) = 0 for each node x.
We visit all nodes of DAG in ascending topological order and consider the following:
Let x be the currently processed node and let y be its predecessor, i.e. edge (y, x) exists. If there is a path of exactly length D which ends in y then this paths can be extended along the edge (y, x) to new length D+1.
This idea leads to the recurrences:
P(x, 0) = 1.
P(x, 1) = sum {P(y,0) | y is a predecessor of x , i.e. edge (y, x) exists}.
P(x, 2) = sum {P(y,1) | y is a predecessor of x , i.e. edge (y, x) exists}.
P(x, 3) = sum {P(y,2) | y is a predecessor of x , i.e. edge (y, x) exists}.

The implementation of the recurrences is straighforward, we visit each edge (y, x) exactly once and during that visit update the three values P(x, 1), P(x, 2), P(x, 3) in constant time. The resulting complexity is then at most linear with respect to the number of edges, namely it is $\Theta(|E(G)| + |V(G)|)$.

8. Describe a method based on DP which finds the total number of all paths in DAG, i.e. the number of paths of all possible lengths.

Hint/solution The solution is analogous to the solution of the previous problem. We define analogously the values P(x, 1), P(x, 2), ..., P(x, N-2), P(x, N-1) and fill the table in the same manner. The operational complexity of the solution rises to $\Theta(|V(G)| \cdot |E(G)|)$ and the memory complexity raises to o $\Theta(|V(G)|^2)$ as we have to store N values in each of N nodes.

9. We want to know the number of all binary vectors of length N which do not contain two immediately neighbouring 1's. (E.g. vector 0100100101 is acceptable, vectors 01100, 111011 are not acceptable).
Let symbol P(N, 0) denote the number of all such vectors which have length N and their last digit is 0.
Analogously, let P(N, 1) denote the number of all such vectors which have length N and their last digit is 1.

A. Formulate recurrences which will express exactly how the values P(N, 0) a P(N, 1) depend on the values P(N-1, 0) and P(N-1, 1) .
B. Apply the DP strategy to compute the value P(12, 0) + P(12, 1) according to the recurrences derived in A.

Hint/solution
A.
1. P(1, 0) = 1.
2. P(1, 1) = 1.
3. P(N, 0) = P(N-1, 0) + P(N-1, 1).
4. P(N, 1) = P(N-1, 0).
Substitute 4. to 3. to obtain
3'. P(N, 0) = P(N-1, 0) + P(N-2, 0)
which is by the way the same recurrence formula as the formula which defines the Fibonacci sequence
0, 1, 1, 2, 3, 5, 8, ...

B.
P(12, 0) + P(12, 1) = P(13, 0) = Fib15 = 377.

10. Each edge in a given DAG is colored green or blue. Describe a method based on DP which finds the longest path in DAG with the additional property that the colors of each two neighbouring (incident) edges in the path are different.

Hint/solution Define in each node x two values: Db(x) and Dg(x).
Db(x) represents the length of the longest path which satisfies the problem conditions and which ends in x and whixh last edge is blue.
Dg(x) represents the length of the longest path which satisfies the problem conditions and which ends in x and which last edge is green.

For each node x it holds:

Db(x) = 0	if no blue edge ends in x,
Db(x) = 1 + max{Dg(y) | edge (y, x) exists and it is blue }	otherwise.

Dg(x) = 0	if no green edge ends in x,
Dg(x) = 1 + max{Db(y) | edge (y, x) exists and it is green }	otherwise

The graph is then traversed in topological order, the values Db(x) and Dg(x) are computed and the longest path is reconstructed in the same manner as it is done in the standard DP solution which finds the plain longest path in DAG.