1 Introduction

Dynamic programming is a powerful mathematical optimization technique that breaks the problem into subproblems of smaller dimensions, solves these subproblems only once, memorizes the result for each subproblem and then uses them to get the answer for subproblems of larger dimensions. Eventually, this leads to the solution of the initial problem. In order to get an optimal answer, each of the problems must be solved optimally.

There are two ways of how a problem can be solved with the help of dynamic programming.

Top-down approach. In this method, we break a problem into smaller subproblems, solve each subproblem and then combine the results to get the answer for the initial problem. The relation between the problem and its subproblems is represented by recurrent formulas.

Bottom-up approach. In this method, we find the answer for the current problem and then use its result to get answers for problems of larger dimensions. Our initial problem will be their subproblem.

In this homework, we will consider a top-down approach, although many problems can be solved in both ways.

Let’s show how dynamic programming works with an example.

Problem. You are given a knapsack of weight \( W \) and \( n \) items of weights \( w_1, w_2, \ldots, w_n \). You need to fill the knapsack in a such way that the total weight of all items in the knapsack is maximized and doesn’t exceed the capacity of the knapsack.

Solution. Let \( f_{i,k} \) be the optimal weight of all items in the knapsack of weight \( k \), when we can use frost \( i \) items. Here, we have defined the problem of the dimension \((i, k)\). These two parameters help us to define a problem (note that function \( f \) may also depend on variables, that have no connection with the dimension of the problem).

Let’s assume that we know optimal answers for all subproblems of smaller sizes, i.e. for all problems \((i', k')\), such that \( i' \leq i \) and \( k' \leq k \) and \((i, k) \neq (i', k')\). Now we want to find an answer for problem \((i, k)\). Let’s look at item number \( i \). There are two possible actions that we can take: we can either include item \( i \) in the knapsack, if there is enough capacity for it, or we can ignore it. If we ignore item \( i \), this means that we have to fill the knapsack of the weight \( k \) optimally without using this item. Thus, we can only include items from 1 to \( i - 1 \). Therefore the subproblem where we don’t include item \( i \) will be \((i - 1, k)\). The answer for this subproblem is \( f_{i-1,k} \).

Now let’s consider the case where we try to put item \( i \) into the knapsack. It is only possible, if the capacity of the knapsack is not less than the weight of the item \( i \), i.e. \( k \geq w_i \). If this condition is true, then we put the item in the knapsack. As regards the rest of space in the knapsack, we will try to fill it with items 1, 2, \ldots, \( i - 1 \) in the most optimal way. This subproblem will have dimension \((i - 1, k - w_i)\), where \( k - w_i \) is available capacity left in the knapsack after we put item \( i \) in it. So, all smaller subproblems of problem \((i, k)\) can be divided into two sets: when we include item \( i \) in the knapsack and when we don’t. Using all these observations we can calculate the value \( f_{i,k} \):

For \( i > 0 \):

\[
f_{i,k} = \begin{cases} 
max(f_{i-1,k-w_i} + w_i, f_{i-1,k}) & \text{if } k - w_i \geq 0 \\
 f_{i-1,k} & \text{otherwise}
\end{cases}
\]

This is a recurrent formula for dynamic programming. We can use recursion in order to compute it. An important thing to remember here is that we should calculate values for problems \( f_{i,k} \) only once. We can do this by memorizing all the computed values and just use them, if they are needed. This will improve time complexity dramatically.

Similar to mathematical induction, dynamic programming technique requires base.

As we can see the recurrent formula above makes sense only when \( i > 0 \). So the base case will be \( f_{0,k} = 0 \), where \( k \in [0, W] \).

The correctness of such recurrent equations are proven in two steps: first we prove the base case, then we prove the formula. When we prove the formula, we use the assumption that all subproblems are solved optimally.
optimally. For example, in this problem all possible valid combinations of first $i$ items can be divided into sets: when we include item $i$ into the knapsack and when we don’t. We have optimal answers for both of these case in our subproblems $(i - 1, k)$ and $(i, k - w_i)$, thus the answer for $(i, k)$ also will be optimal.

As each value $f_{i,k}$ is calculated only once, then the time complexity for this problem is $\theta(nW)$.

2 Problem 1

*Problem:* You are given a sequence of $n$ integers $a_1, a_2, ..., a_n$. Find the longest increasing subsequence $a_{k_1}, a_{k_2}, ..., a_{k_m}$, such that $k_i \in \{1, 2, ..., n\}$ for $i \in \{1, 2, ..., m\}$, $k_1 < k_2 < ... < k_m$ and $a_{k_1} \leq a_{k_2} \leq ... \leq a_{k_m}$.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

*Solution:*

1. Let $f_i$ be the length of the longest increasing subsequence, that ends with element on position $i$. Then:
   Base case:
   
   $$f_0 = 0$$
   
   For $i > 0$:
   
   $$f_i = \max(f_j + 1) \text{ for all } j: 0 \leq j < i \text{ and } a_j \leq a_i$$

   The answer for the initial problem: $f_n$

3 Problem 2

*Problem:* Find all binary sequences of the length $n$, such that no sequence has two consecutive zeros.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

*Solution:*

1. Let $f_i$ be the number of all binary sequences of the length $i$
   Base case:
   
   $$f_0 = 0, f_1 = 2$$
   
   For $i > 1$:
   
   $$f_i = f_{i-1} + f_{i-2}$$

   The answer for the initial problem: $f_n$
4 Problem 3

Problem: You are given a knapsack of weight $W$ and $n$ items with integer weights $w_1, w_2, ..., w_n$. Items have their cost $c_1, c_2, ..., c_n$. **You can take each item only once.** Your goal is to fill the knapsack with items, such that their cost is maximized.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:

1. Let $f_{i,k}$ be the optimal cost of items in the knapsack of the weight $k$, when we have only first $i$ items. Then:
   - Base case:
     \[ f_{0,k} = 0 \] for all $k \in [0,W]$
   - For other values:
     \[ f_{i,k} = \begin{cases} 
     \max(f_{i-1,k-w_i} + c_i, f_{i-1,k}) & \text{if } k - w_i \geq 0 \\
     f_{i-1,k} & \text{otherwise}
     \end{cases} \]
   The answer for the initial problem: $f_{n,W}$

5 Problem 4

Problem: You are given a knapsack of the weight $W$ and $n$ items with integer weights $w_1, w_2, ..., w_n$. Each item has its cost $c_1, c_2, ..., c_n$. **You can take each item as many times as you want.** Your goal is to fill the knapsack with items, such that their cost is maximized.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:

1. Let $f_{i,k}$ be the optimal cost of items in the knapsack of the weight $k$, when we have only first $i$ items. Then:
   - Base case:
     \[ f_{0,k} = 0 \] for all $k \in [0,W]$
   - For other values:
     \[ f_{i,k} = \begin{cases} 
     \max(f_{i-1,k-w_i} + c_i, f_{i-1,k}) & \text{if } k - w_i \geq 0 \\
     f_{i-1,k} & \text{otherwise}
     \end{cases} \]
   The answer for the initial problem: $f_{n,W}$
6 Problem 5

Problem: You are given a knapsack of the weight $W$ and $n$ items with integer weights $w_1, w_2, ..., w_n$. Each item has its cost $c_1, c_2, ..., c_n$. You can take item $i$ only $b_i$ times, where $b_i$ is a positive integer number for $i \in \{1, 2, ..., n\}$. Your goal is to fill the knapsack with items, such that their cost is maximized.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:

1. Let $f_{i,k}$ be the optimal cost of items in the knapsack of the weight $k$, when we have only first $i$ items. Then:
   Base case:
   \[ f_{0,k} = 0 \text{ for all } k \in [0,W] \]
   For other values:
   \[
   f_{i,k} = \begin{cases} 
   \max(f_{i-1,k} - l \times w_i + l \times c_i, f_{i-1,k}) & \text{for all } l: l \times k - w_i \geq 0 \text{ and } 1 \leq l \leq b_i \\
   f_{i-1,k} & \text{otherwise}
   \end{cases}
   \]
   The answer for the initial problem: $f_{n,W}$

7 Problem 6

Problem: You are given two strings $s = s_1s_2...s_n$ and $t = t_1t_2...t_m$. You may perform three type of operations on a string: delete a symbol, insert a symbol or replace existing symbol with any other possible symbol. Assume that all symbols are lowercase English letters. What is the minimum number of operations you need to transform string $s$ to string $t$?

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:

1. Let $f_{i,k}$ be the minimum number of operations needed to transform string $s_1s_2...s_i$ to string $t_1t_2...t_k$. Base case:
   \[ f_{0,k} = k \text{ for all } 1 \leq k \leq mf_{i,0} = i \text{ for all } i: 1 \leq i \leq n \]
   For other values:
   \[
   f_{i,k} = \begin{cases} 
   \min(f_{i-1,k}, f_{i,k-1}, f_{i-1,k-1}) + 1 & \text{if } s_i \neq t_k \\
   \min(f_{i-1,k} + 1, f_{i,k-1} + 1, f_{i-1,k-1}) & \text{otherwise}
   \end{cases}
   \]
   The answer for the initial problem: $f_{n,m}$
8 Problem 7

Problem: You have a board of a dimension $1 \times n$. There is a robot on cell 1. The robot can make moves from cell $i$ to cells $i + 1, i + 2, \ldots, i + k$. Moving to cell $i$ costs $c_i$ resources ($c_1 = 0$). Find the path from cell 1 to cell $n$ for a robot, that requires the minimum amount of resources.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:

1. Let $f_i$ be a minimum amount of resources required for a robot to get to cell $i$.
   
   Base case:
   
   $$f_0 = 0$$
   
   For $i > 0$:
   
   $$f_i = \min(f_{i-j} + c_i) \text{ for all } j : i - j \geq 0 \text{ and } 1 \leq j \leq k$$
   
   The answer for the initial problem: $f_n$

9 Problem 8

Problem: You are given a sequence of real positive numbers $a_1, a_2, \ldots, a_n$. At each step you may erase one number from the sequence. The cost of deleting a number $a_i$ equals $a_{i-1} \times a_{i+1}$, if $1 < i < n$. If $i = 0$, then the cost equals $a_i \times a_{i+1}$, and if $i = n$, then the cost is $a_{i-1} \times a_i$. If there is only one element $i$ in the array we spend $a_i$ resources. Erase all numbers in the array, by spending as less resources as possible.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:

1. Let $f_{i,k}$ be a minimum amount of resources required to erase subarray $a_i, a_{i+1}, \ldots, a_k$.
   
   Base case:
   
   $$f_{i,i} = a_i \text{ for all } i : 1 \leq i \leq n$$
   
   For $i < k$:
   
   $$f_{i,k} = \min(f_{i,j-1} + f_{j+1,k}) + c_j \text{ for all } j : i + 1 \leq j \leq k - 1$$
   
   The answer for the initial problem: $f_{1,n}$
10 Problem 9

Problem: You are given a board of a dimension $N \times M$. There is a robot on cell $(x_0, y_0)$ on the board. At each step robot may go from cell $(x, y)$ to cells $(x-1, y)$, $(x, y-1)$, $(x+1, y)$ and $(x, y+1)$ (all cells must be in table). Find the number of ways in which the robot may get from initial cell to final cell $(x_1, y_1)$ in $k$ steps.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:
1. Let $f_{x,y,k}$ be a number of ways for the robot to get to cell $(x,y)$ in $k$ steps from cell $(x_0, y_0)$. And let:

$$g_{i,j,k} = \begin{cases} f_{i,j,k} & \text{if } 1 \leq i \leq N \text{ and } 1 \leq j \leq M \\ 0 & \text{otherwise} \end{cases}$$

Base case:

$$f_{x,y,0} = 0 \text{ for all } (x, y) : 1 \leq x \leq N \text{ and } 1 \leq y \leq M$$

For $k > 1$:

$$f_{i,j,k} = g_{i-1,j,k-1} + g_{i,j-1,k-1} + g_{i+1,j,k-1} + g_{i,j+1,k-1}$$

The answer for the initial problem: $f_{x_1,y_1,k}$

11 Problem 10

Problem: You are given a sequence of $n$ binary digits $a_1, a_2, ..., a_n$ ($a_i \in \{0, 1\}$ for $i \in \{1, 2, ..., n\}$). Find the number of subsequences $a_{k_1}, a_{k_2}, ..., a_{k_m}$, such that $k_i \in \{1, 2, ..., n\}$ for $i \in \{1, 2, ..., m\}$, $k_1 < k_2 < ... < k_m$ and the sum of all digits in the subsequence is even.

1. Find a dynamic programming formula to solve the problem.
2. Prove its correctness.
3. Show and prove its time complexity.

Solution:
1. Let $f_i$ and $g_1$ be numbers of subsequences of a sequence $a_1, a_2, ..., a_i$ with the even and odd sums respectively. And let:

$$\sigma(x) = \begin{cases} 1 & \text{if } x \text{ is even} \\ 0 & \text{otherwise} \end{cases}$$

Base case:

$$f_0 = g_0 = 0$$

For $i > 0$: 

...
\[ f_i = \begin{cases} f_{i-1} + g_{i-1} & \text{if } \sigma(a_i) = 0 \\ f_{i-1} & \text{otherwise} \end{cases} \]

\[ g_i = \begin{cases} g_{i-1} & \text{if } \sigma(a_i) = 0 \\ f_{i-1} + g_{i-1} & \text{otherwise} \end{cases} \]

The answer for the initial problem: \( f_n \)