Darius Dolha dolha@uni-bremen.de
Mustafa Kemal Kücük mustafak@uni-bremen.com
Vasu Yaduvanshi vasu@uni-bremen.de

Übungsgruppe 51

1 Übung 6.1

1.1 Explanation

Let $x = x_1x_2...x_n$ and $y = y_1y_2...y_m$ be two words use to generate the string S (gemeinsame Teilsequenz). m,n are the sizes of those words

DP Tabelle(teilproblem):

Let $M[i, j]$ be the size of the longest common sequence of the Teilwörtern $x_1x_2...x_i$ and $y_1y_2...x_j$

Base cases:

$M[i, 0] = 0$, $\forall i \in \{0, 1, 2, ..., n\}$
$M[0, j] = 0$, $\forall j \in \{0, 1, 2, ..., m\}$

We can rely on the follwing observations to generate the recursion needed for the subproblems:

1. If $x_i = y_j$, then it means that we found a letter in common, so we will update our Longest Common Subsequence size by 1:

$$M[i, j] = 1 + M[i - 1, j - 1]$$

2. If $x_i \neq y_j$ we should choose if either we ignore the last character of the first or the second string:

2.1 if we ignore the last character of x, we will have:

$$M[i, j] = M[i - 1, j]$$

2.2 Should we ignore the last character of the second string(y_j) our size function will look like this:

$$M[i, j] = M[i, j - 1]$$

So, in this case the size of the Longest Common Subsequence of the two words would be the maximum of the functions.

$$M[i, j] = \max\{M[i - 1, j], M[i, j - 1]\}$$

1.2 Algorithm

Algorithm 1 Longest Common Subsequence (LCS)

Input: Zwei Strings $X = x_1x_2 \dots x_n$ und $Y = y_1y_2 \dots y_m$
Output: LCS(X, Y)

```
Initialisiere Array M[·,·] mit Dimension (n+1) × (m+1)
Setze M[i,0] := 0   ∀i ∈ {0,...,n}
Setze M[0,j] := 0   ∀j ∈ {0,...,m}
for i = 1,...,n do
    for j = 1,...,m do
        if x_i = y_j then
            M[i,j] := M[i-1,j-1] + 1
        else
            M[i,j] := max(M[i-1,j], M[i,j-1])
        end if
    end for
end for
return M[n,m]
```

1.3 Correctness

The algorithm is correct because:

- **Base Cases:** For empty prefixes, the LCS is 0, which is correctly handled by initializing $M[i, 0]$ and $M[0, j]$ to 0.
- **Inductive Step:**
  - If $x[i] = y[j]$, the definition correctly extends the LCS by including the matching character.
  - If $x[i] \neq y[j]$, it optimally chooses the maximum LCS length by excluding one of the characters, which is the only valid decision.

It can also be proven by an example, going step by step.

By filling the table from smaller subproblems to larger ones, we ensure every entry is computed using previously solved subproblems, adhering to the principle of dynamic programming.

Time Complexity

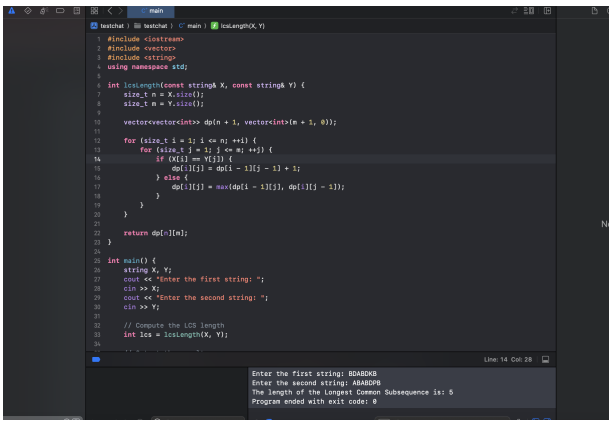
The time complexity is $O(n \cdot m)$, where:

- n: Length of x,
- m: Length of y.

This is because we compute each entry of the $(n + 1) \times (m + 1)$ table exactly once, and each entry computation takes $O(1)$.

1.4 Implementation

We can observe that the program is working properly.

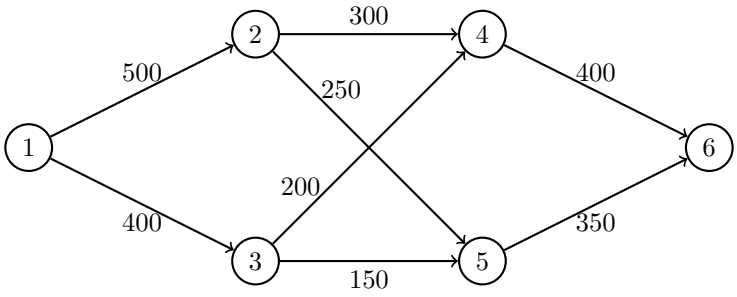


2 Übung 6.2

2.1 Graph

We are going to model the problem as a maximum flow represent every city as a node and the number of telephone connections between cities as edges. The cities have been coded as follows:

New York - 1, Chicago - 2, Memphis - 3, Denver -4, Dallas - 5, Los Angeles - 6.



2.2 Maximum Flow

In order to compute the maximul flow we are going to use the Ford-Fulkerson algorithm:

We define the residual capacity as:

$$\overline{c_f}(a) := \begin{cases} c(a) - f(a) & \text{falls } a \in A \text{ (Vorwärtskante)} \\ f(a) & \text{falls } a \in \overleftarrow{A} \text{ (Rückwärtskante)} \end{cases}$$

The flow function is defined as:

$$f(a) := \begin{cases} f(a) + \gamma & \text{falls } a \in P \\ f(a) - \gamma & \text{falls } \overleftarrow{a} \in P \\ f(a) & \text{sonst} \end{cases}$$

First we are going to choose 1 and 6 as the source and the sink. It is the only possibility. Also at the beginning $f(i) = 0$, $\forall i$.

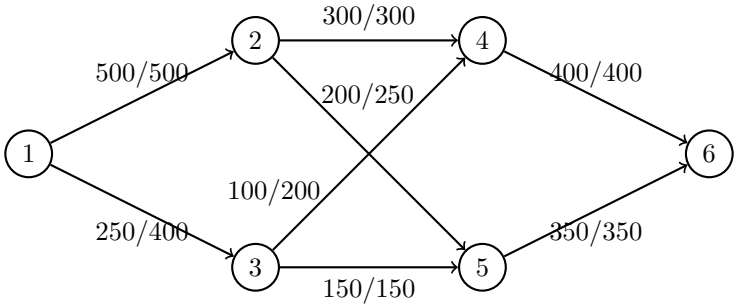
Lets choose the flowing augmented path: $1- > 2- > 4- > 6$.
Our bottleneck in this case is $\gamma = \min_{a \in P} c_{f(a)} = 300$. Our flows are: $f(1- > 2) = 300$,$f(2- > 4) = 300$, $f(4- > 6) = 300$. Now, we are going to update the residual capacity. $c_f(1- > 2) = 200$, $c_f(2- > 4) = 0$, $c_f(4- > 6) = 100$.
In the residual graph we have $c_f(2- > 1) = 300$, $c_f(4- > 2) = 300$, $c_f(6- > 4) = 300$

Then we will choose the folowing augmented path: $1- > 3- > 5- > 6$.
Our bottleneck in this case is $\gamma = \min_{a \in P} c_{f(a)} = 150$. Our flows are: $f(1- > 3) = 150$, $f(3- > 5) = 150$, $f(5- > 6) = 150$. Now, we are going to update the residual capacity. $c_f(1- > 3) = 200$, $c_f(3- > 5) = 150$, $c_f(5- > 6) = 200$.
In the residual graph we have $c_{f'}(3- > 1) = 150$, $c_f(5- > 3) = 150$, $c_f(6- > 5) = 150$.

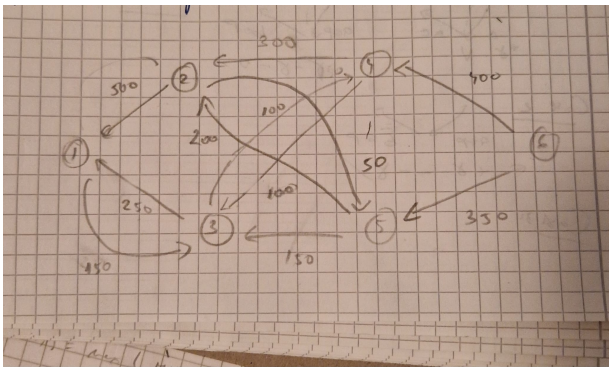
Then we will choose the folowing augmented path: $1- > 2- > 5- > 6$. Our bottleneck in this case is $\gamma = \min_{a \in P} c_{f(a)} = 200$. Our flows are $f(1- > 2) = 500$, $f(2- > 5) = 200$, $f(5- > 6) = 350$. Now, we are going to update the residual capacity. $c_f(1- > 2) = 0$, $c_f(2- > 5) = 50$, $c_f(5- > 6) = 0$.
In the residual graph we have $c_f(2- > 1) = 500$, $c_f(5- > 2) = 200$, $c_f(6- > 5) = 350$.

The last path we can choose is: $1- > 3- > 4- > 6$. ur bottleneck in this case is $\gamma = \min_{a \in P} c_{f(a)} = 100$. Our flows are $f(1- > 3) = 250$, $f(3- > 4) = 100$, $f(4- > 6) = 400$. Now, we are going to update the residual capacity. $c_f(1- > 3) = 50$, $c_f(3- > 4) = 100$, $c_f(4- > 6) = 0$.
In the residual graph we have $c_{f'}(3- > 1) = 250$, $c_f(4- > 3) = 100$, $c_f(6- > 4) = 400$.

Our total flow into the source(maximum flow will be) $f = f(4- > 6) + f(5- > 6) = 400 + 350 = 750$



In the residual graph we have. (i don't know how to draw this in overleaf)



2.3 Minimum Schnitt

After running the Ford-Fulkerson algorithm, we examine the residual network. Nodes reachable from the source $s = 1$ in the residual network form one subset U. The rest form $V \setminus U$.

Nodes in $U : \{1, 2, 3\}$ (New York, Chicago, Memphis).

Nodes in $V \setminus U : \{4, 5, 6\}$ (Denver, Dallas, Los Angeles).

The cut edges are the edges in the original network that go from U to $V \setminus U$:

$(2 \to 4)$, capacity: 300.

$(2 \to 5)$, capacity: 250.

$(3 \to 5)$, capacity: 150.

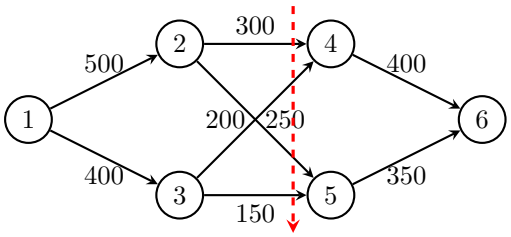
- **Value of the Minimum Cut:**

$$\text{cap}(C) = 300 + 250 + 150 = 750.$$

- **Set of Cut Edges:**

$$C = \{(2 \to 4), (2 \to 5), (3 \to 5)\}.$$

Diagram of the Network and Minimum Cut



3 Übung 6.3

We are going to use the following observation:

1. **Split Each Node (Except s and t)**: - Replace each node $v \in V \setminus \{s, t\}$ with two nodes v_{in} and v_{out}. - Add an edge between v_{in} and v_{out} with capacity $c(v)$ (the node capacity).
2. **Redirect Edges**: - For each incoming edge (u, v), redirect it to connect u to v_{in}. - For each outgoing edge (v, w), redirect it to connect v_{out} to w.
3. **Keep s and t Unchanged**: - The source s and sink t are not split, and their edges remain as in the original graph.

Consider the following example: - Nodes: $V = \{s, v_1, v_2, t\}$, - Node capacities: $c(v_1) = 3$, $c(v_2) = 2$, - Edges: $(s, v_1), (v_1, v_2), (v_2, t)$.

1. Split v_1 into $v_{1,\text{in}}$ and $v_{1,\text{out}}$ with an edge $(v_{1,\text{in}}, v_{1,\text{out}})$ of capacity 3.
2. Split v_2 into $v_{2,\text{in}}$ and $v_{2,\text{out}}$ with an edge $(v_{2,\text{in}}, v_{2,\text{out}})$ of capacity 2.
3. Redirect edges: - (s, v_1) becomes $(s, v_{1,\text{in}})$, - (v_1, v_2) becomes $(v_{1,\text{out}}, v_{2,\text{in}})$, - (v_2, t) becomes $(v_{2,\text{out}}, t)$.

—

Transformed Graph

The transformed graph is as follows:
Nodes: $\{s, v_{1,\text{in}}, v_{1,\text{out}}, v_{2,\text{in}}, v_{2,\text{out}}, t\}$,
Edges: - $(s, v_{1,\text{in}})$, capacity: ∞,
- $(v_{1,\text{in}}, v_{1,\text{out}})$, capacity: 3,
- $(v_{1,\text{out}}, v_{2,\text{in}})$, capacity: ∞,
- $(v_{2,\text{in}}, v_{2,\text{out}})$, capacity: 2,
- $(v_{2,\text{out}}, t)$, capacity: ∞.

Diagram