MaxFlow & MinCut

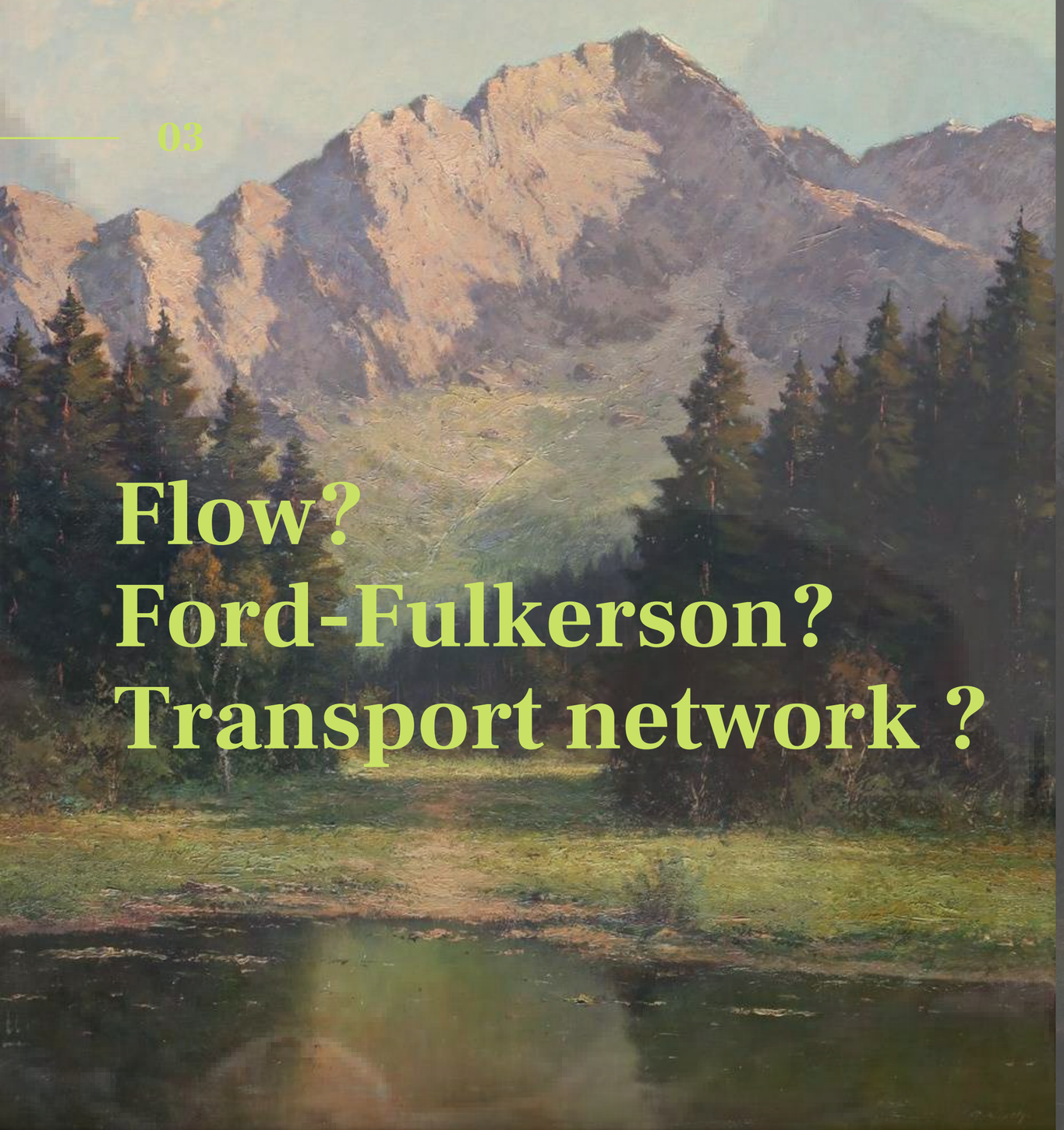
preapared by DormLads

What is MaxFlow & MinCut ?

In optimization theory and graph theory, the maximum flow problem is to find such a flow over a transport network that the sum of the flows from the source, or, equivalently, the sum of the flows to the drain is the maximum.

The maximum flow problem is a special case of more difficult problems, such as the circulation problem.

In 1955, Lester Ford and Delbert Fulkerson first constructed an algorithm specifically designed to solve this problem. Their algorithm is called the Ford-Fulkerson algorithm.



03

Flow?
Ford-Fulkerson?
Transport network ?

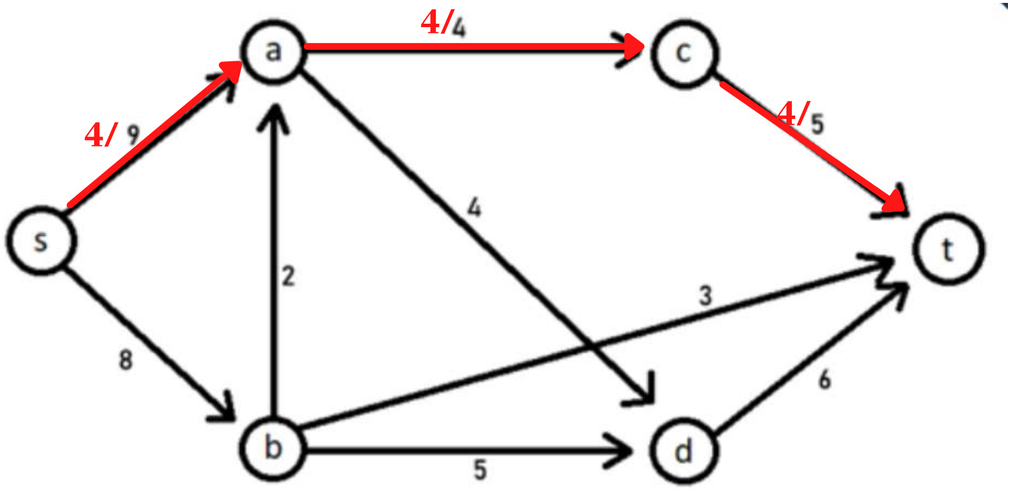
Yes, perhaps, at first glance, everything seems incomprehensible . Let's first understand the concepts of flow and transport network and understand what we are talking about .

For example , imagine that you are a Kazakh merchant and are going to sell horse meat to Bishkek, logistics is not yet developed and you will have to take cargo transportation into your own hands

The red arrows are railways, and the numbers above them are their capacity .Our goal is to pass through these railway connections as much of our goods as possible .

05

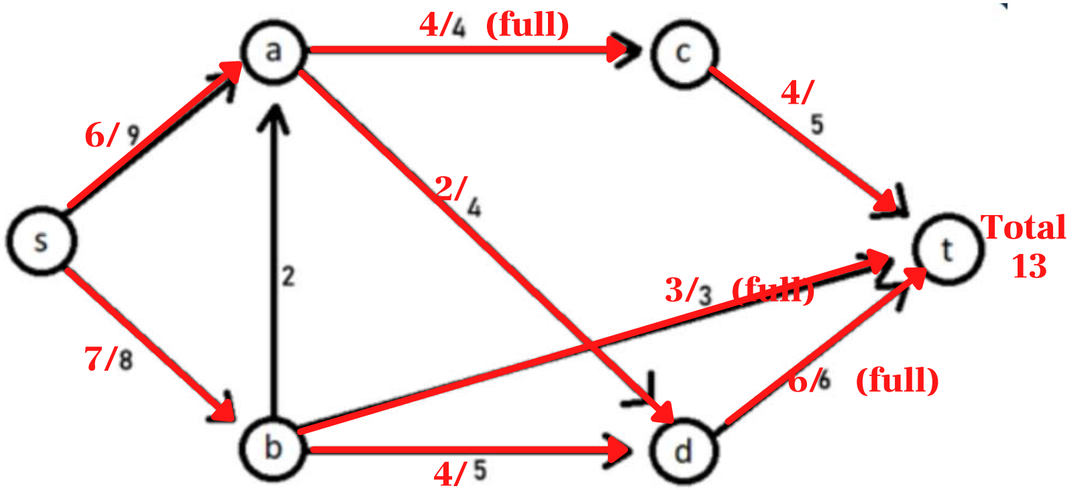
Note that we cannot push more goods than the possible capacity of the railway connection at any given time .In this case, we sum up in advance the volume of the stream that we are going to send through the extreme possibilities .But not in all cases we can not send as much horse meat as we want, so in some networks we will be limited .This in turn will lead us to the maximum flow , the minimum cut of our maximum flow that we will be able to send .From the very beginning, it will be limited to the minimum cut that we can find in this transport network .We'll get to that , but for now we're just trying to find the optimal flow, and then we're going to draw those relationships later.



For a visual example, we have translated our map to this " drawing".

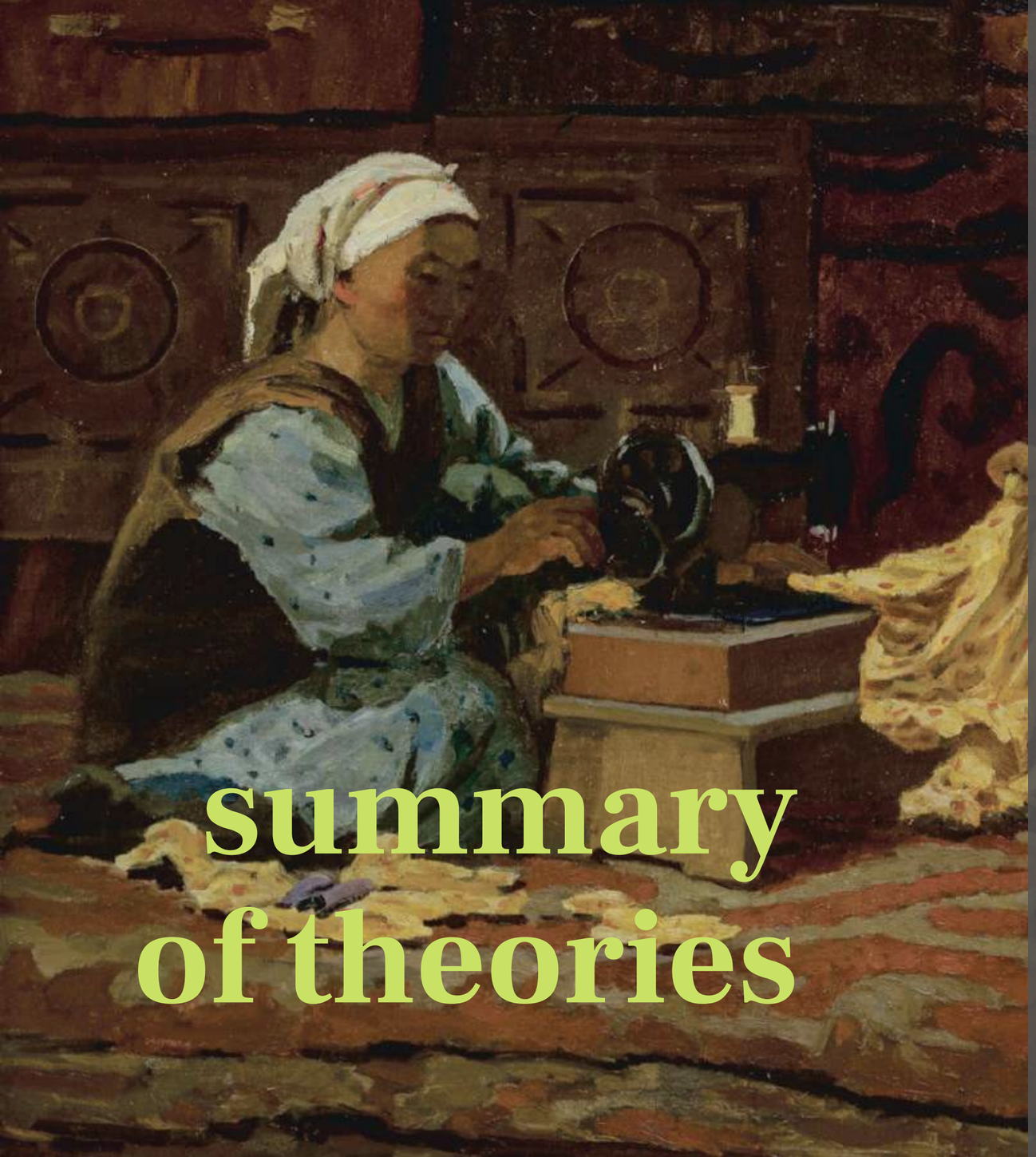
First we choose the path s-a-c-t. On this path, the minimum capacity is 4, and imagine that we sent 4 tons of horse meat.

Next, we chose the s-a-d-t path , and sent 2 tons . Also on the way s-b-t , and sent 3 tons, and on the way s-b-d-t 4 tons .



At the end, we received 13 tons of horse meat in Bishkek ,which is the maximum flow.We were so smart ,in the first time we distributed the cargo correctly, and we became a kind of math guy who beat Ford and Fulkerson by time .

Now, anyone who has watched closely may notice that **the sum of the minimum cut is equal to the maximum flow**. That is, if we add a-c to b-t and to the path d+t (4+3+6= 13) we get 13 .



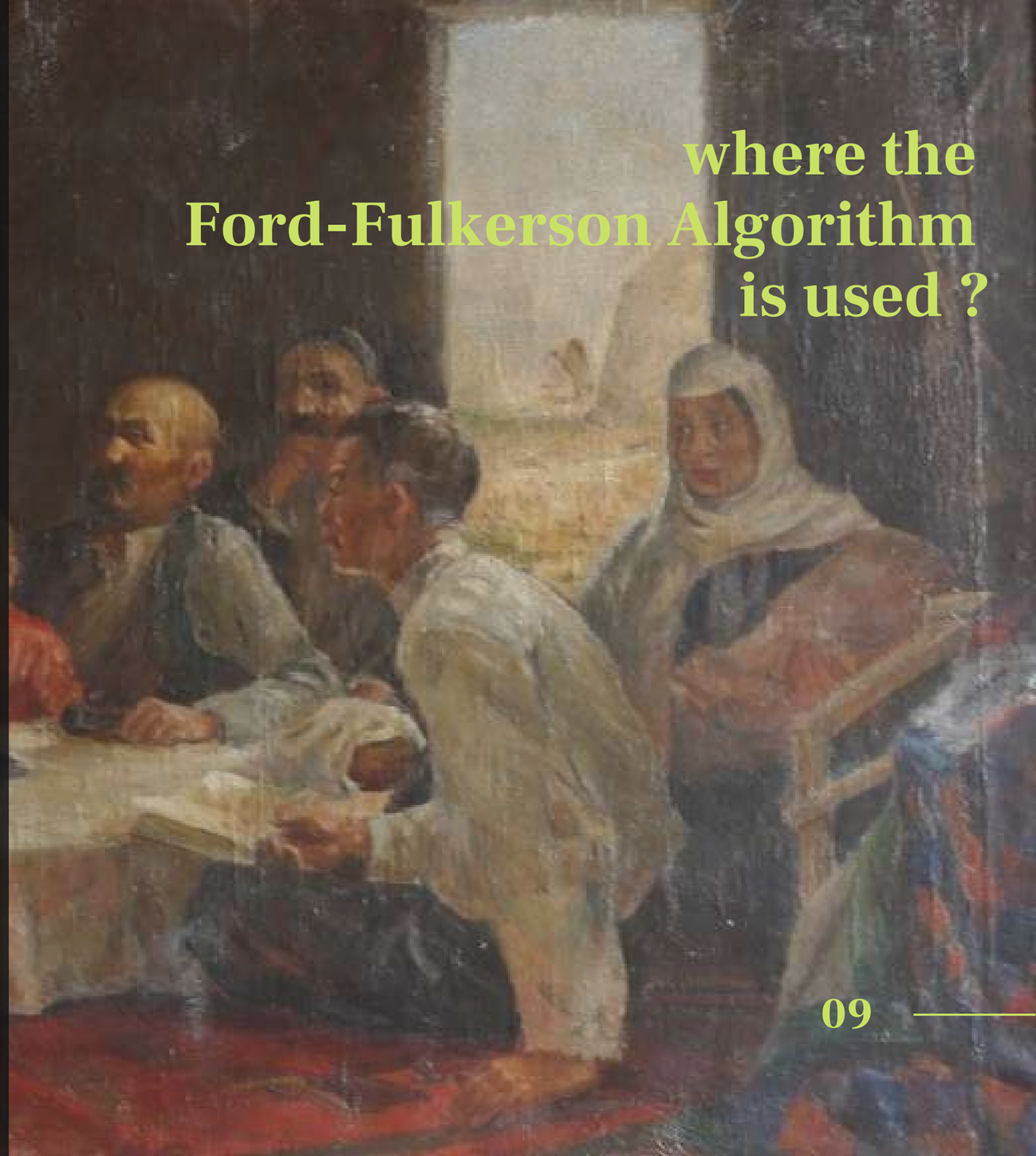
summary of theories

So what does this give you, a brilliant businessman from Aktobe ?

1. If you are a pessimist, the maximum amount of goods that you can send
2. If you are an optimist, it gives you information where you can expand the railway path to send even more goods

where the Ford-Fulkerson Algorithm is used ?

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.

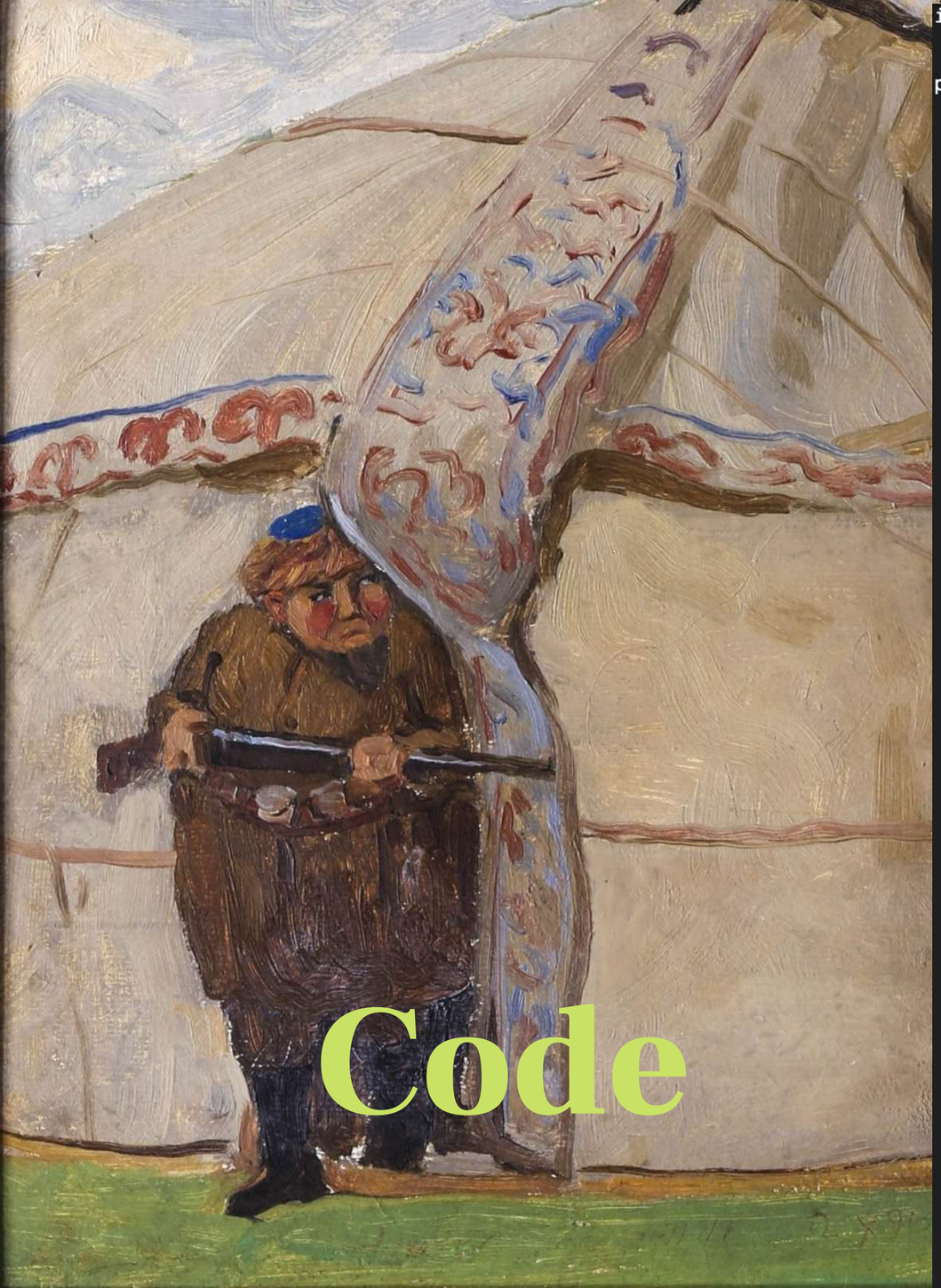


Time Complexity Analysis

Time Complexity Analysis

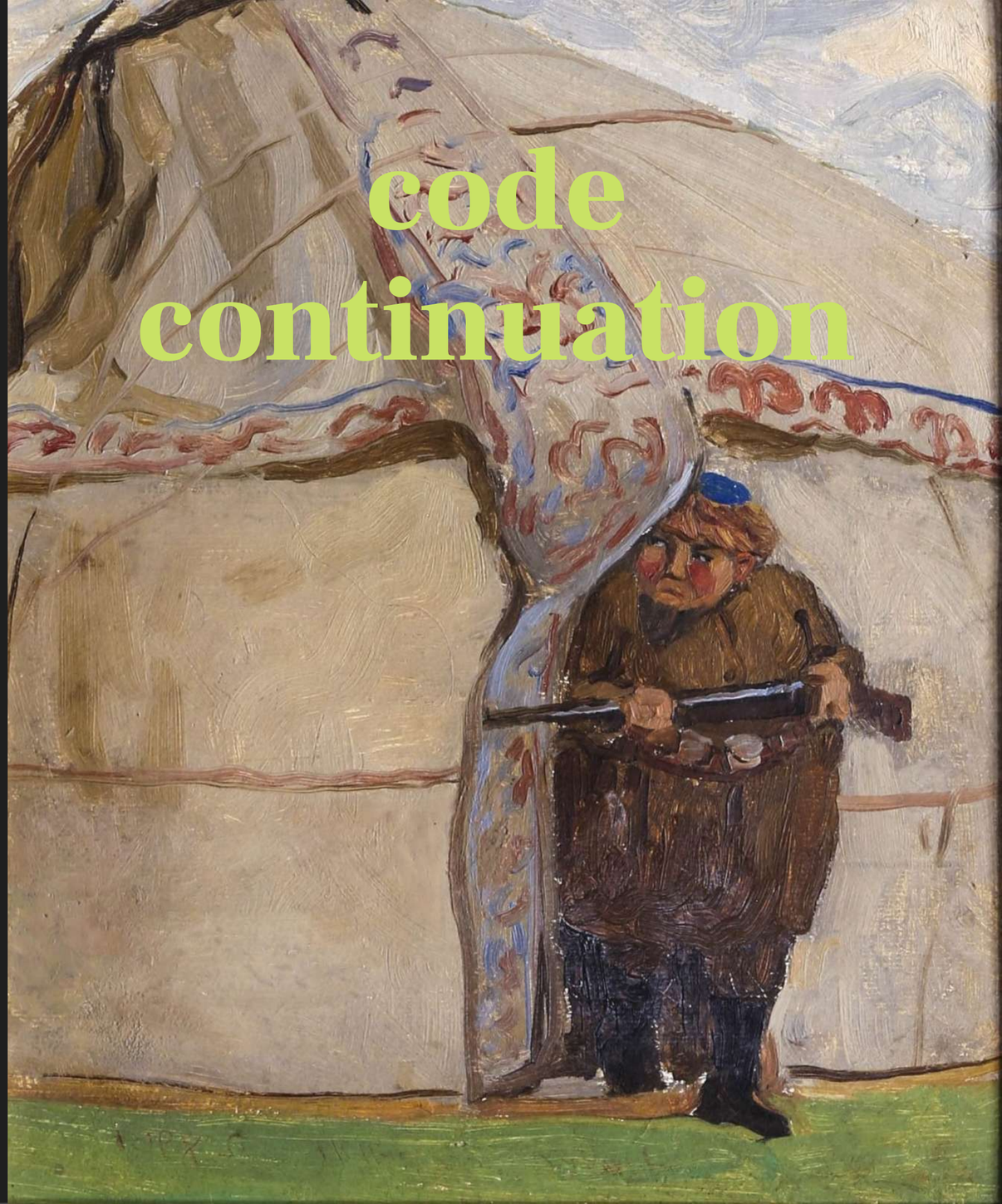
The Ford-Fulkerson algorithm depends heavily on the method used to find an augmented path. An augmented path can be found using a Breadth-first search (BFS) or Depth-first search (DFS). If we choose an augmented path using BFS or DFS, the algorithm runs in polynomial time.

The execution time of BFS is equal to O(E'), where E' is the number of edges in the residual graph G_f. For each edge, the flow will be increased, and in the worst case, it reaches its maximum flow value f*. Therefore the algorithm will be iterated at most f* times. Hence, the overall time complexity of the Ford-Fulkerson algorithm would be O(f* × E').

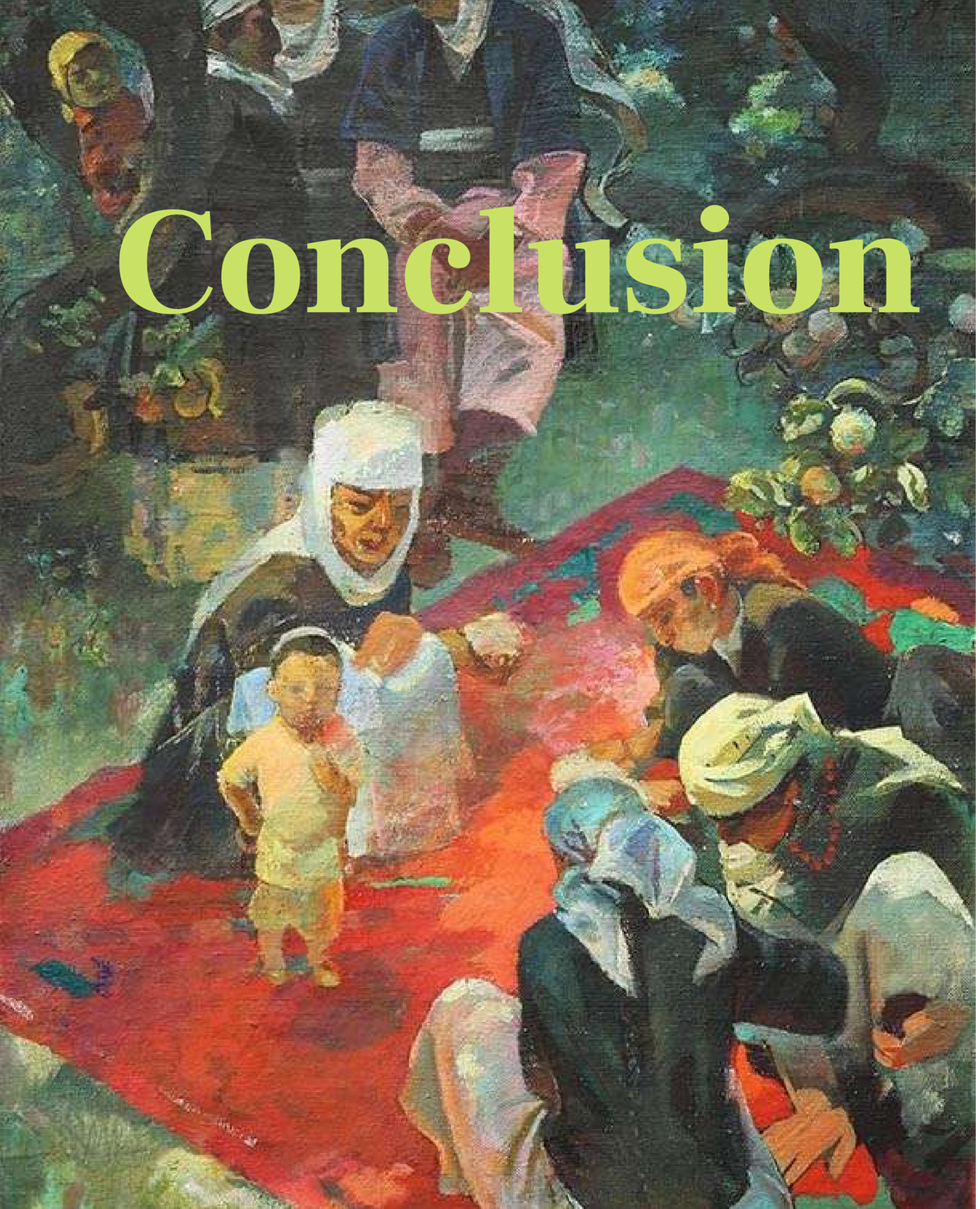



```java
import java.util.*;


public class Main {
    private static boolean bfs(int[][] residualGraph, int start, int goal, int[] parent,int numberOfVertices) {
        boolean[] visited = new boolean[numberOfVertices + 1];
        Queue<Integer> queue = new LinkedList<Integer>();
        boolean pathFound = false;
        int destination, element;
        for (int vertex = 1; vertex <= numberOfVertices; vertex++)
        {
            parent[vertex] = -1;
            visited[vertex] = false;
        }
        queue.add(start);
        parent[start] = -1;
        visited[start] = true;

        while (!queue.isEmpty())
        {
            element = queue.remove();
            destination = 1;
            while (destination <= numberOfVertices)
            {
                if (residualGraph[element][destination] > 0 &&  !visited[destination])
                {
                    parent[destination] = element;
                    queue.add(destination);
                    visited[destination] = true;
                }
                destination++;
            }
        }

        if (visited[goal])
        {
            pathFound = true;
        }
        return pathFound;
    }

    private static void dfs(int[][] rGraph, int start,
                            boolean[] visited) {
        visited[start] = true;
        for (int i = 0; i < rGraph.length; i++) {
            if (rGraph[start][i] > 0 && !visited[i]) {
                dfs(rGraph, i, visited);
            }
        }
    }

    private static void MaxFlowMinCut(int[][] graph, int start, int goal,int numberOfVertices) {
        int u, v;
        int maxFlow = 0;
        int pathFlow = Integer.MAX_VALUE;
        int[][] residualGraph = new int[graph.length][graph.length];
        for (int i = 0; i < graph.length; i++) {
```

```
        for (int j = 0; j < graph.length; j++) {
            residualGraph[i][j] = graph[i][j];
        }
    }


    int[] parent = new int[graph.length];

    while (bfs(residualGraph, start, goal, parent.numberOfVertices)) {


        for (v = goal; v != start; v = parent[v]) {
            u = parent[v];
            pathFlow = Math.min(pathFlow, residualGraph[u][v]);
        }

        for (v = goal; v != start; v = parent[v]) {
            u = parent[v];
            residualGraph[u][v] = residualGraph[u][v] - pathFlow;
            residualGraph[v][u] = residualGraph[v][u] + pathFlow;
        }
        maxFlow += pathFlow;
    }

    boolean[] isVisited = new boolean[graph.length];
    dfs(residualGraph, start, isVisited);

    System.out.println("MinCut is: ");
    for (int i = 0; i < graph.length; i++) {
        for (int j = 0; j < graph.length; j++) {
            if (graph[i][j] > 0 && isVisited[i] && !isVisited[j]) {
                System.out.println(i + " -> " + j);
            }
        }
    }
    System.out.println("MaxFlow is: " + maxFlow);
}


public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Напишите сколько вершин в вашем графе");
    int NumVertex = sc.nextInt();
    int[][] graph = new int[NumVertex][NumVertex];
    for(int i =0;i<NumVertex;i++){
        for(int j =0;j<NumVertex;j++){
            graph[i][j] = sc.nextInt();
        }
    }

    MaxFlowMinCut(graph, 0, NumVertex-1, NumVertex-1);
}
}
```



The above implementation of Ford Fulkerson Algorithm is called Edmonds-Karp Algorithm. The idea of Edmonds-Karp is to use BFS in Ford Fulkerson implementation as BFS always picks a path with minimum number of edges. When BFS is used, the worst case time complexity can be reduced to O(VE2). The above implementation uses adjacency matrix representation though where BFS takes O(V2) time, the time complexity of the above implementation is O(EV3).
This is an important problem as it arises in many practical situations. Examples include, maximizing the transportation with given traffic limits, maximizing packet flow in computer networks.



Conclusion

In this presentation, we discussed how a simple example (which is very necessary in life) can explain the minimum cut, the maximum flow of the graph, and how to find them . We have presented the Ford-Fulkerson algorithm for solving the maximum flow problem in a graph. We tested the Ford-Fulkerson algorithm by example and analyzed the time complexity. Implemented an improved algorithm code .

And yes, there is a lot of national painting in this presentation, as it was made during the month of Nauryz . (Design author of the presentation K.Kadyrov .)


Happy New Year !
Thank you for your attention!