CS333 Project

Hasan Can Sevindir

Introduction

In this project we have the role of the CEO of a start-up company and we are expected to make some important decisions using our networkFlow knowledge and some network flow algorithms. We are given lists of: ventures, prerequisite, outcomes of the venture projects and are expected to find the list venture projects with the total of maximum profit.

Problem Statement

I used **MaximumFlow** approach with Ford-Furkerson's algorithm with **BFS**(breathe first search travercing methodology) in order to solve and come up with a feasible solution for this problem.

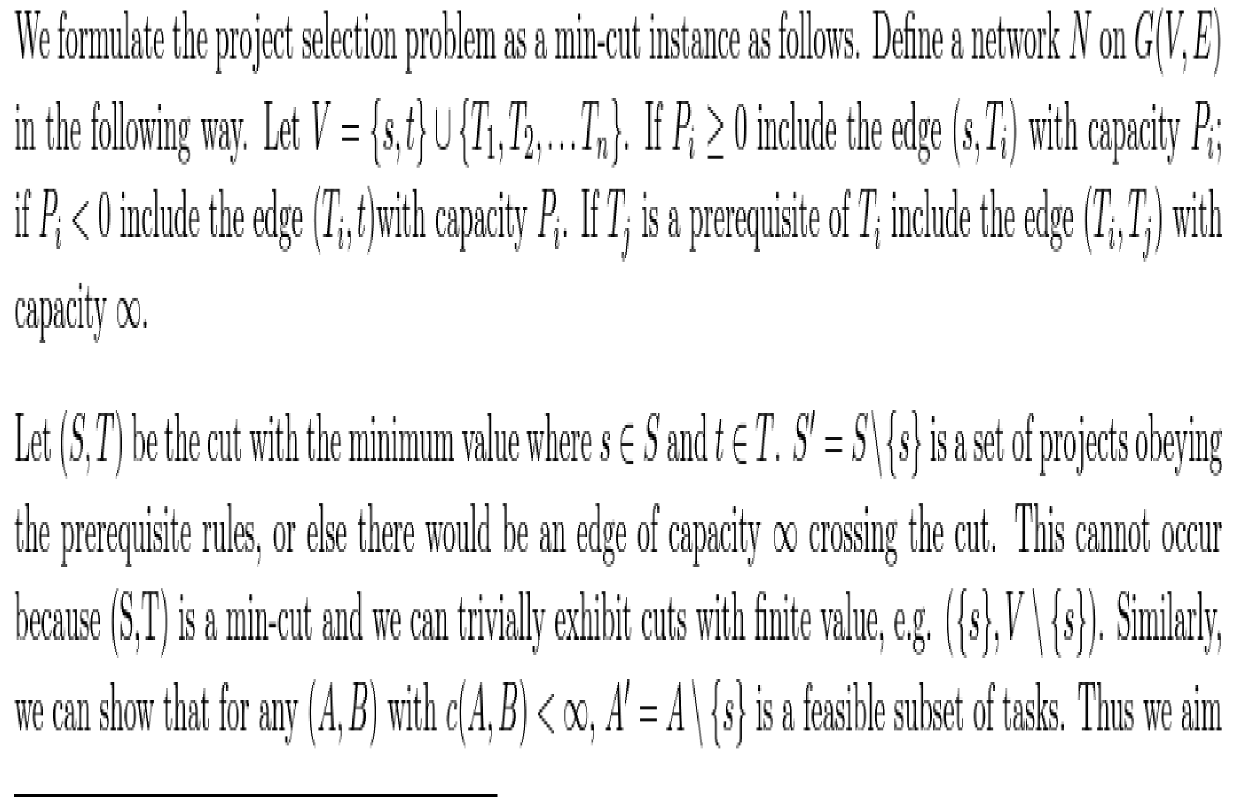
The **Ford-Fulkerson** algorithm for the maximum flow problem can be applied to solve the maximum profit problem. The goal is to create a graph with the venture projects represented as vertices between the **source** and **sink**, with the source denoting the beginning of the project selection process and the sink denoting its conclusion.

The number of projects that may be chosen is represented by the **capacity** of the edges from the source to the venture projects, and the **profit** of each project is represented by the **capacity** of the edges from the venture projects to the sink. Project i cannot be picked without also selecting project j, and vice versa, as shown by the capacity of the edges between the venture projects.

We then apply the Ford-Fulkerson technique to discover the maximum flow in the network in order to determine the subset of projects to execute and the maximum profit. The **subset** of projects to be executed found by following the residual graph's route from the source back to the sink. The maximum flow shows the maximum number of **projects** that can be chosen. The total of the profits from the chosen projects may then be used to compute the maximum profit.

By choosing the subset of projects that maximizes the profit while meeting all project dependencies, this method enables us to identify the best solution to the maximum profit problem.

Mathematical explanation of the project using the algorithm:



We formulate the project selection problem as a min-cut instance as follows. Define a network N on $G(V, E)$ in the following way. Let $V = \{s, t\} \cup \{T_1, T_2, \ldots T_n\}$. If $P_i \geq 0$ include the edge (s, T_i) with capacity P_i; if $P_i < 0$ include the edge $(T_i, t)$with capacity P_i. If T_j is a prerequisite of T_i include the edge (T_i, T_j) with capacity ∞.

Let (S, T) be the cut with the minimum value where $s \in S$ and $t \in T$. $S' = S \setminus \{s\}$ is a set of projects obeying the prerequisite rules, or else there would be an edge of capacity ∞ crossing the cut. This cannot occur because (S,T) is a min-cut and we can trivially exhibit cuts with finite value, e.g. $(\{s\}, V \setminus \{s\})$. Similarly, we can show that for any (A, B) with $c(A, B) < \infty$, $A' = A \setminus \{s\}$ is a feasible subset of tasks. Thus we aim to show that finding the min-cut is equivalent to maximizing total profit, i.e. we show that the total profit of S' is greater than or equal to the total profit of A'. Let $P^+ = \sum_i P_i \mathbb{I}(P_i \geq 0)$ and $P^- = \sum_i P_i \mathbb{I}(P_i < 0)$, where $\mathbb{I}(\cdot)$ is the indicator function. It is easy to show that:

$$c(A, B) = P^+ - \sum_{P_i \in A} P_i \mathbb{I}(P_i \geq 0) - \left(P^- - \sum_{P_i \in A} P_i \mathbb{I}(P_i < 0) \right).$$

Thus $c(S, T) \leq c(A, B)$ implies that

$$P^+ - P^- - \left(\sum_{P_i \in S} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in S} P_i \mathbb{I}(P_i < 0) \right) \leq P^+ - P^- - \left(\sum_{P_i \in A} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in A} P_i \mathbb{I}(P_i < 0) \right)$$

or equivalently

$$\sum_{P_i \in S'} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in S'} P_i \mathbb{I}(P_i < 0) \geq \sum_{P_i \in A'} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in A'} P_i \mathbb{I}(P_i < 0)$$

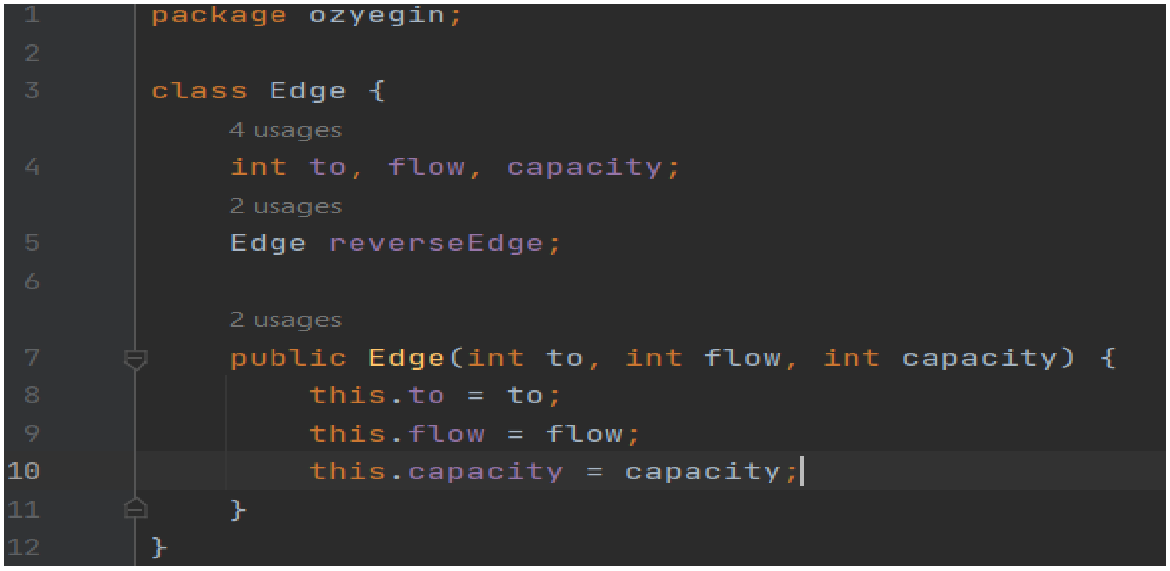
Note that the left hand side is the total profit of S' and the right hand side is the total profit of A'. Thus we have shown that any other feasible solution has profit at most equal to the profit of S'.

Implementation Details

- I implemented the project without using any external library otherthan JDK itself, and the program is dynamic so that we can add as much as **n** values as we want, so it's not limited to 20, we can have big amount of project lists and the program will solve for them.

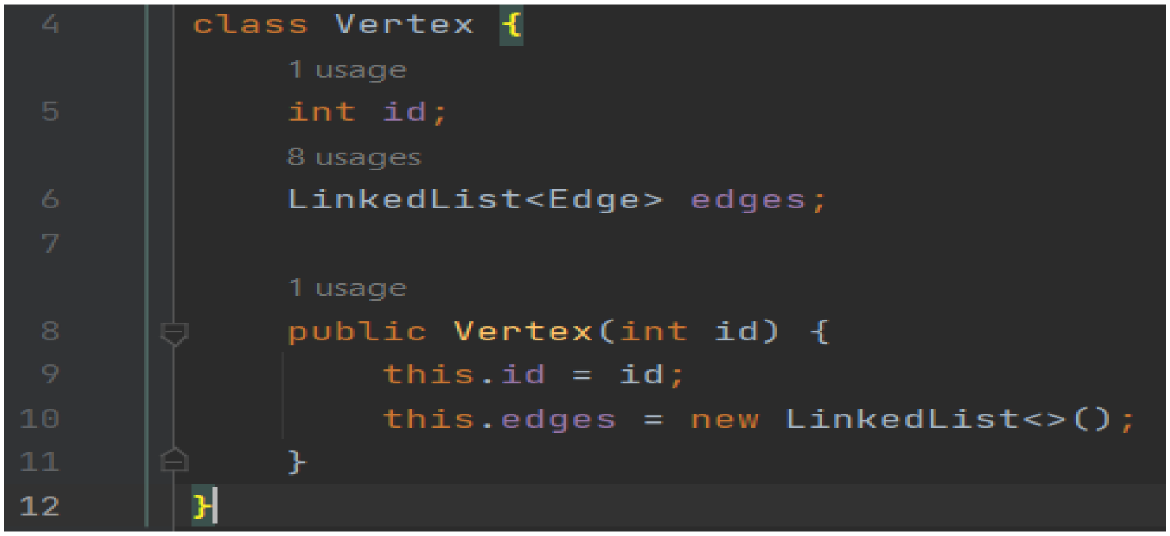
- I implemented custom graph in order to use it throughout the implementation.

- **Edge class:**
  - This class acts like an Edge of a graph, stores the Edge related information such us capacity of an Edge in a graph and flow and to.



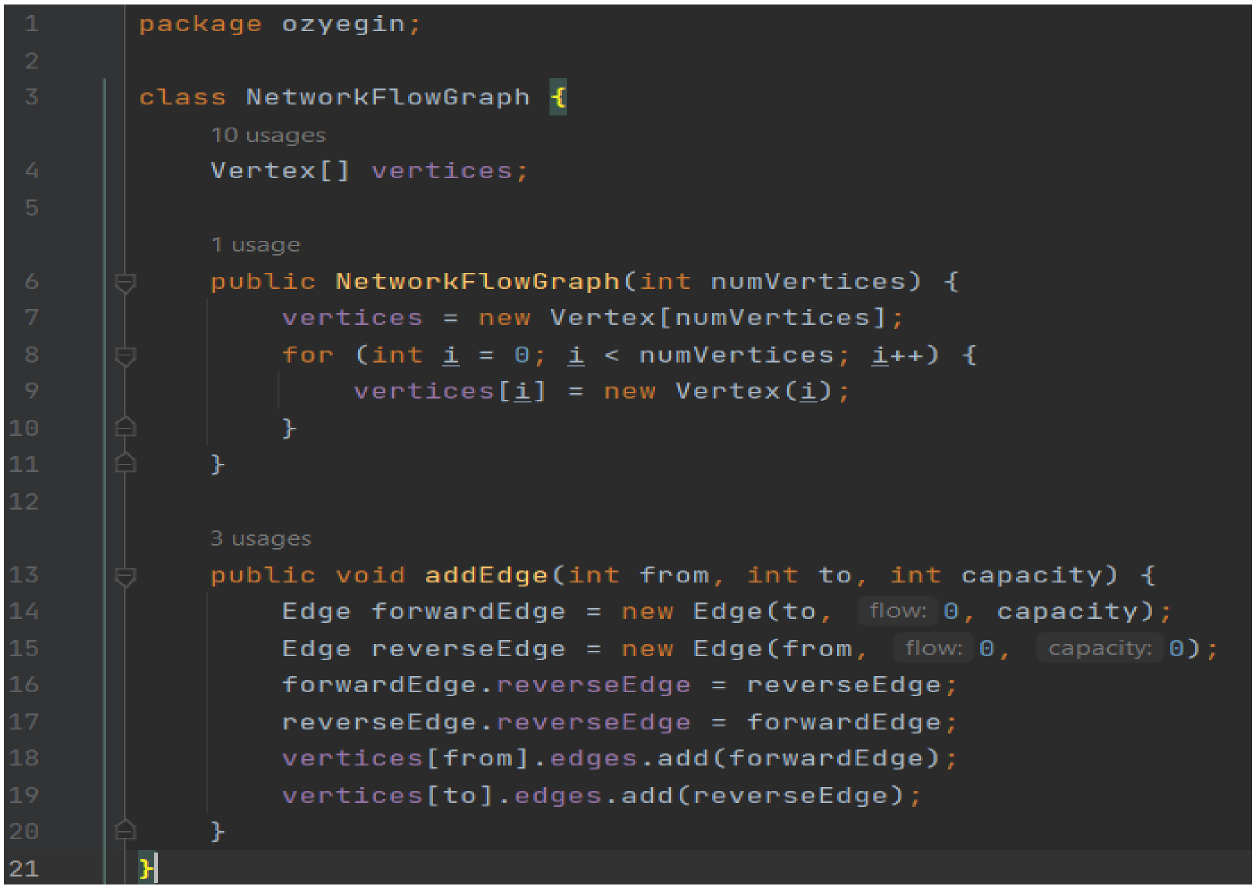
```
package ozyegin;

class Edge {
    4 usages
    int to, flow, capacity;
    2 usages
    Edge reverseEdge;

    2 usages
    public Edge(int to, int flow, int capacity) {
        this.to = to;
        this.flow = flow;
        this.capacity = capacity;
    }
}
```

- **Vertex class:**
    - This class class act like vertex of a graph in data structures, it has edges and id to keep track of it.



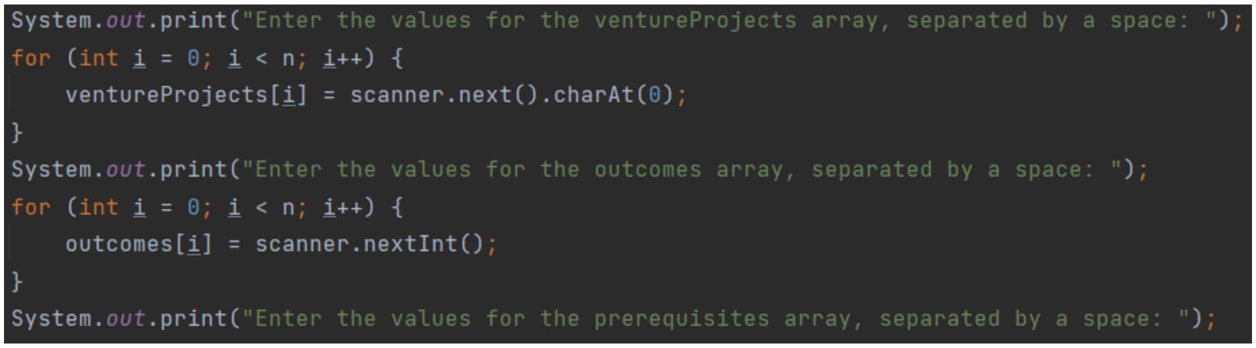
```
class Vertex {
    1 usage
    int id;
    8 usages
    LinkedList<Edge> edges;

    1 usage
    public Vertex(int id) {
        this.id = id;
        this.edges = new LinkedList<>();
    }
}
```

- **NeworkGraph class:**
    - This class acts like the main graph used in the project, it has vertex list and addEdge methos which addes new edge into the graph with corresponding vertex.



```
package ozyegin;

class NetworkFlowGraph {
    10 usages
    Vertex[] vertices;

    1 usage
    public NetworkFlowGraph(int numVertices) {
        vertices = new Vertex[numVertices];
        for (int i = 0; i < numVertices; i++) {
            vertices[i] = new Vertex(i);
        }
    }

    3 usages
    public void addEdge(int from, int to, int capacity) {
        Edge forwardEdge = new Edge(to, flow: 0, capacity);
        Edge reverseEdge = new Edge(from, flow: 0, capacity: 0);
        forwardEdge.reverseEdge = reverseEdge;
        reverseEdge.reverseEdge = forwardEdge;
        vertices[from].edges.add(forwardEdge);
        vertices[to].edges.add(reverseEdge);
    }
}
```

- **Driver class:**
    - This is main class where we get the user inputs and initialize the required data structures and call the main algorithm to solve the desired problem.
    - The program start with getting inputs from the console



```
System.out.print("Enter the values for the ventureProjects array, separated by a space: ");
for (int i = 0; i < n; i++) {
    ventureProjects[i] = scanner.next().charAt(0);
}
System.out.print("Enter the values for the outcomes array, separated by a space: ");
for (int i = 0; i < n; i++) {
    outcomes[i] = scanner.nextInt();
}
System.out.print("Enter the values for the prerequisites array, separated by a space: ");
```

- Below we construct the graph and fill it’s content with the proper content in order to feed it to our maximum flow finder algorithm.

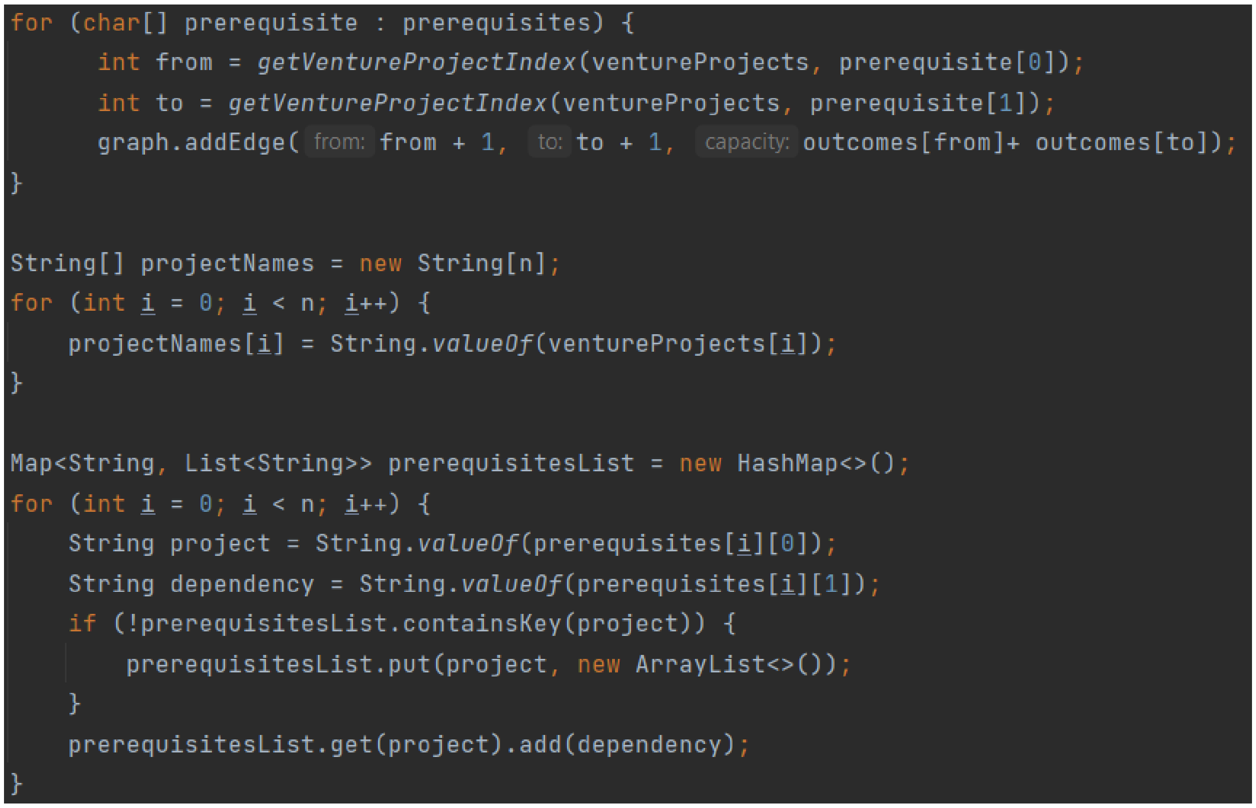
- We first add edges between venture projects based on the prerequisites input. It is doing this in several steps:
    - Iterate through the prerequisites array: in the for loop we iterate through the prerequisites array, which is a 2D array containing pairs of venture projects that have a prerequisite relationship.

    - Find the indices of the venture projects: then we use getVentureProjectIndex method to find the indices of the venture projects in the ventureProjects array. The getVentureProjectIndex method searches the ventureProjects array for the given venture project and returns its index if found, or -1 if not found.

    - Add an edge between the venture projects: following by that we then add the edge between the venture projects in the adjacency matrix by calling the addEdge method of the graph object. The addEdge method takes three arguments: the index of the source node, the index of the destination node, and the capacity of the edge. In this case, the source and destination nodes are the indices of the venture projects, and the capacity of the edge is the sum of the outcomes of the two venture projects.

- Create the projectNames array: then we create an array called projectNames which is an array of strings containing the names of the venture projects. This array is created by converting the characters in the ventureProjects array to strings.

- Create the prerequisitesList map: later we create a map called prerequisitesList which is a mapping from venture projects to lists of their dependencies. The code does this by iterating through the prerequisites array and adding the dependencies to the appropriate lists in the map. If a project does not yet have an entry in the map, the code adds a new entry with an empty list of dependencies.



```
for (char[] prerequisite : prerequisites) {
      int from = getVentureProjectIndex(ventureProjects, prerequisite[0]);
      int to = getVentureProjectIndex(ventureProjects, prerequisite[1]);
      graph.addEdge( from: from + 1,  to: to + 1,  capacity: outcomes[from]+ outcomes[to]);
}

String[] projectNames = new String[n];
for (int i = 0; i < n; i++) {
    projectNames[i] = String.valueOf(ventureProjects[i]);
}

Map<String, List<String>> prerequisitesList = new HashMap<>();
for (int i = 0; i < n; i++) {
    String project = String.valueOf(prerequisites[i][0]);
    String dependency = String.valueOf(prerequisites[i][1]);
    if (!prerequisitesList.containsKey(project)) {
        prerequisitesList.put(project, new ArrayList<>());
    }
    prerequisitesList.get(project).add(dependency);
}
```

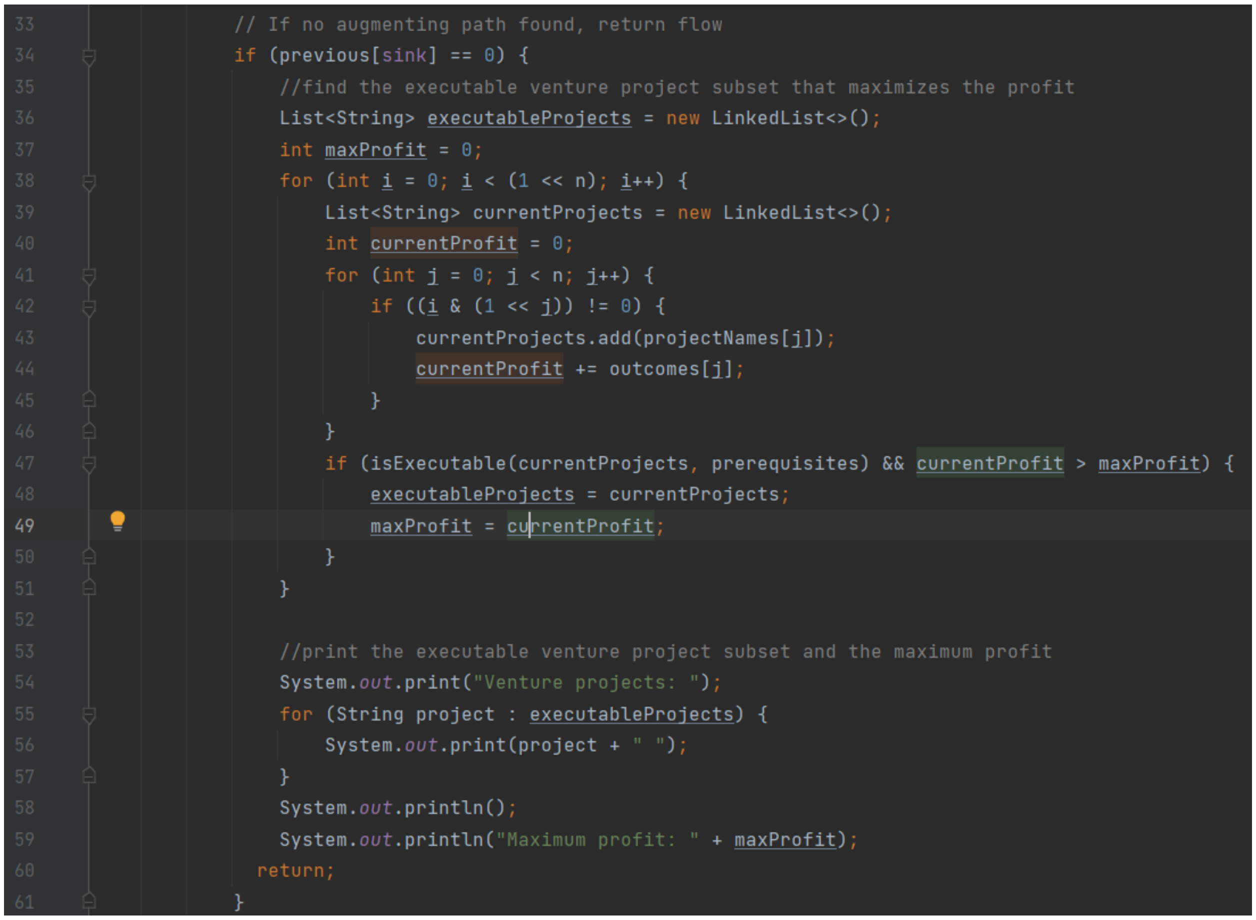
- **MaximumFlow class:**
    - This class is core and main logic and backbone of the projec where we implement the maximumflow ford-forkerson and bfs approach in order to solve the problem.
    - public void getMaximumFlow(int n,int[] outcomes, String[] projectNames, Map<String, List<String>> prerequisites)
    - The method getMaximumFlow is a function that takes in four arguments:
    - **n**: an integer representing the number of venture projects
    - **outcomes**: an array of integers representing the outcomes of the venture projects
    - **projectNames**: an array of strings representing the names of the venture projects
    - **prerequisites**: a map from strings (project names) to lists of strings (project dependencies)
    - Here we use Ford-Fulkerson algorithm to calculate the maximum flow in a network flow graph. The graph consists of a source vertex, a sink vertex, and

vertices representing the venture projects. The edges in the graph represent the dependencies between the venture projects.

- The method begins by initializing the flow to 0. It then enters a while loop that will continue until an augmenting path cannot be found. Within the loop, the method uses a breadth-first search (BFS) to find an augmenting path from the source to the sink. If no augmenting path is found, the loop is exited and the flow is returned.

- If an augmenting path is found, then we calculate the minimum flow through the path and updates the flow and residual capacities of the edges in the graph accordingly. The minimum flow is then added to the overall flow.

- After the while loop has exited, the method finds the subset of executable venture projects that maximizes the profit. It does this by iterating through all possible subsets of the venture projects and calculating the profit for each subset. If a subset is executable (i.e., all of its dependencies are satisfied) and has a higher

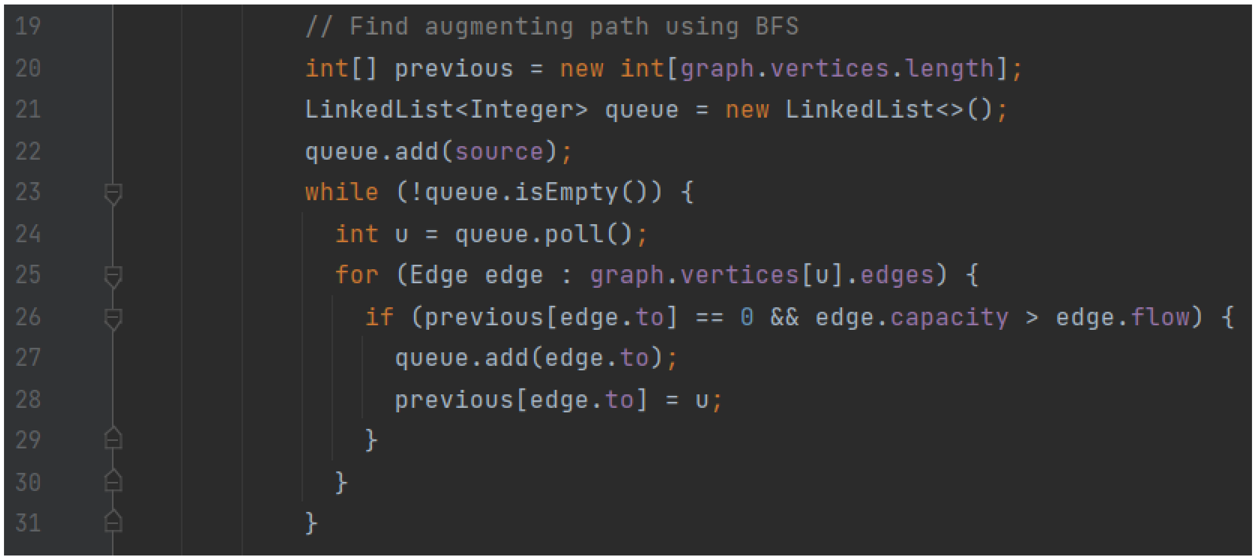


```java
        // If no augmenting path found, return flow
        if (previous[sink] == 0) {
            //find the executable venture project subset that maximizes the profit
            List<String> executableProjects = new LinkedList<>();
            int maxProfit = 0;
            for (int i = 0; i < (1 << n); i++) {
                List<String> currentProjects = new LinkedList<>();
                int currentProfit = 0;
                for (int j = 0; j < n; j++) {
                    if ((i & (1 << j)) != 0) {
                        currentProjects.add(projectNames[j]);
                        currentProfit += outcomes[j];
                    }
                }
                if (isExecutable(currentProjects, prerequisites) && currentProfit > maxProfit) {
                    executableProjects = currentProjects;
                    maxProfit = currentProfit;
                }
            }

            //print the executable venture project subset and the maximum profit
            System.out.print("Venture projects: ");
            for (String project : executableProjects) {
                System.out.print(project + " ");
            }
            System.out.println();
            System.out.println("Maximum profit: " + maxProfit);
          return;
        }
```

profit than the current maximum, the maximum is updated.

    - Finally, the method prints the subset of executable venture projects and the maximum profit.

- **BFS part:**



```
            // Find augmenting path using BFS
            int[] previous = new int[graph.vertices.length];
            LinkedList<Integer> queue = new LinkedList<>();
            queue.add(source);
            while (!queue.isEmpty()) {
              int u = queue.poll();
              for (Edge edge : graph.vertices[u].edges) {
                if (previous[edge.to] == 0 && edge.capacity > edge.flow) {
                  queue.add(edge.to);
                  previous[edge.to] = u;
                }
              }
            }
```

- The breadth-first search (BFS) algorithm is a graph traversal algorithm that explores vertices in the order of their distance from the source vertex. BFS is used in the getMaximumFlow method to find an augmenting path in the network flow graph.

- The BFS algorithm works as follows:

- The source vertex is added to a queue.
- The vertex at the front of the queue is removed and all of its neighbors are added to the queue.
- Steps 2 and 3 are repeated until the queue is empty.
- In the getMaximumFlow method, the BFS algorithm is implemented as follows:
- source vertex is added to the queue.
- While the queue is not empty:
    - vertex at the front of the queue is removed and stored in a variable u.
    - For each edge in the list of edges for vertex u:
    - If the edge's capacity is greater than its flow and the edge's destination vertex has not been visited, the destination vertex is added to the queue and its previous vertex is set to u.
- If the sink vertex has not been visited, no augmenting path was found and the loop is exited. Otherwise, an augmenting path has been found and the loop continues.
- The BFS algorithm is used in the getMaximumFlow method to find an augmenting path because it is able to efficiently explore the graph and find the shortest path between the source and sink vertices. This is important because the Ford-Fulkerson algorithm requires that the augmenting path be the shortest path in order to guarantee the correctness of the maximum flow calculation.

Complexity Analysis

getMaximumFlow():

The complexity of the **getMaximumFlow()** method depends on the size of the input network flow graph and the number of times the loop iterates.

The size of the input network flow graph is determined by the number of vertices n and the number of edges m. The getMaximumFlow method has a time complexity of O(n + m) because it loops through all of the vertices and edges of the graph once.

The number of times the loop iterates depends on the structure of the input graph and the value of the maximum flow. In the worst case, the loop will iterate n - 1 times, where n is the number of vertices in the graph. This occurs when the maximum flow is equal to the minimum cut, which is the smallest possible cut that separates the source from the sink. The minimum cut is equal to the number of vertices minus one because at least one vertex must be on the source side of the cut and at least one vertex must be on the sink side of the cut.

Therefore, the overall time complexity of the **getMaximumFlow()** method is **O(n^2 + nm)**, which is determined by the O(n^2) time complexity of the loop and the **O(n + m)** time complexity of the **BFS** algorithm.

The space complexity of the **getMaximumFlow()** method is **O(n + m)** because it stores the graph in an adjacency list, which requires space proportional to the number of vertices and edges. The method also stores the previous vertices and the queue in memory, which also require space proportional to the number of vertices.

Program Execution:

I have test the program with 5 different input sets and gor the results

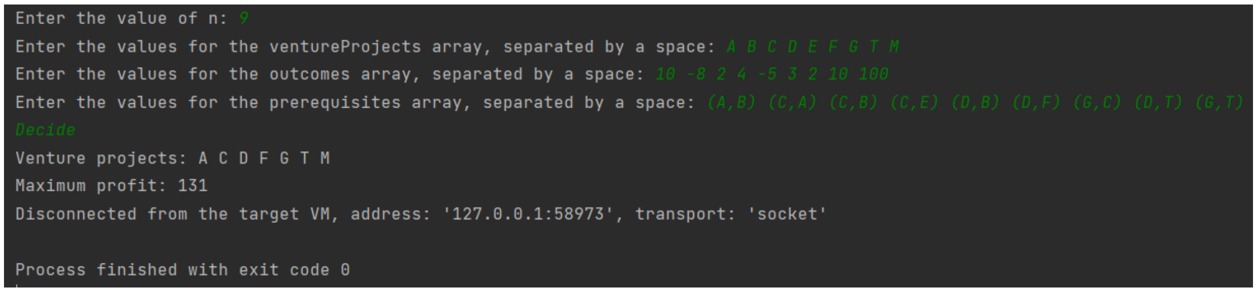
Input sets:

****

TEST INPUTS 1 >>

9
A B C D E F G T M
10 -8 2 4 -5 3 2 10 100
(A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T)
Decide

****



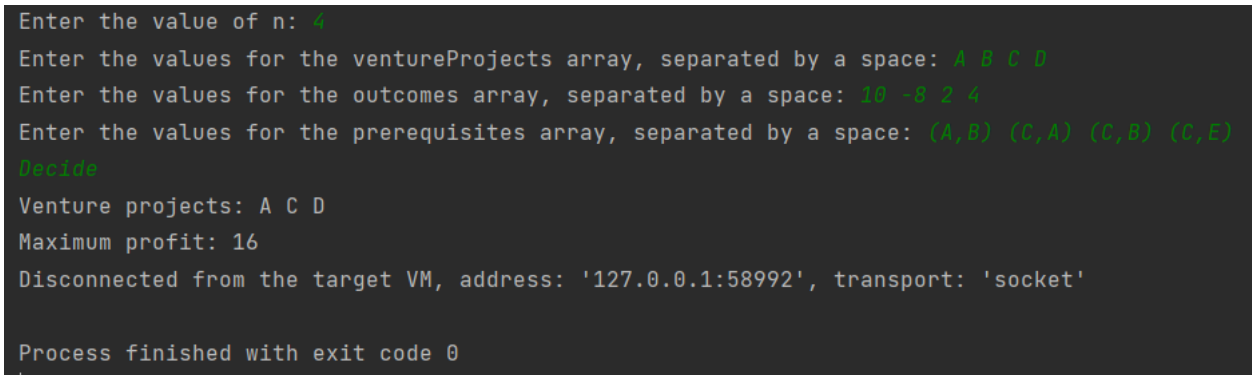
```
Enter the value of n: 9
Enter the values for the ventureProjects array, separated by a space: A B C D E F G T M
Enter the values for the outcomes array, separated by a space: 10 -8 2 4 -5 3 2 10 100
Enter the values for the prerequisites array, separated by a space: (A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T)
Decide
Venture projects: A C D F G T M
Maximum profit: 131
Disconnected from the target VM, address: '127.0.0.1:58973', transport: 'socket'

Process finished with exit code 0
```

****

TEST INPUTS 2 >>

4
A B C D
10 -8 2 4
(A,B) (C,A) (C,B) (C,E)
Decide

****



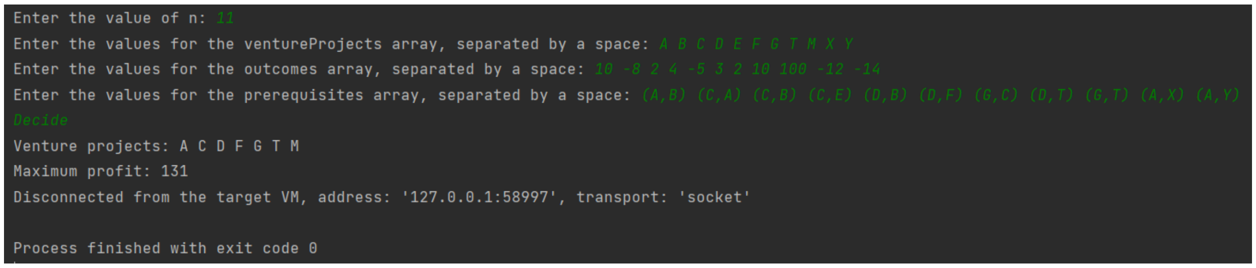
```
Enter the value of n: 4
Enter the values for the ventureProjects array, separated by a space: A B C D
Enter the values for the outcomes array, separated by a space: 10 -8 2 4
Enter the values for the prerequisites array, separated by a space: (A,B) (C,A) (C,B) (C,E)
Decide
Venture projects: A C D
Maximum profit: 16
Disconnected from the target VM, address: '127.0.0.1:58992', transport: 'socket'

Process finished with exit code 0
```

****

TEST INPUTS 3 >>

11
A B C D E F G T M X Y
10 -8 2 4 -5 3 2 10 100 -12 -14
(A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T) (A,X) (A,Y)
Decide

****



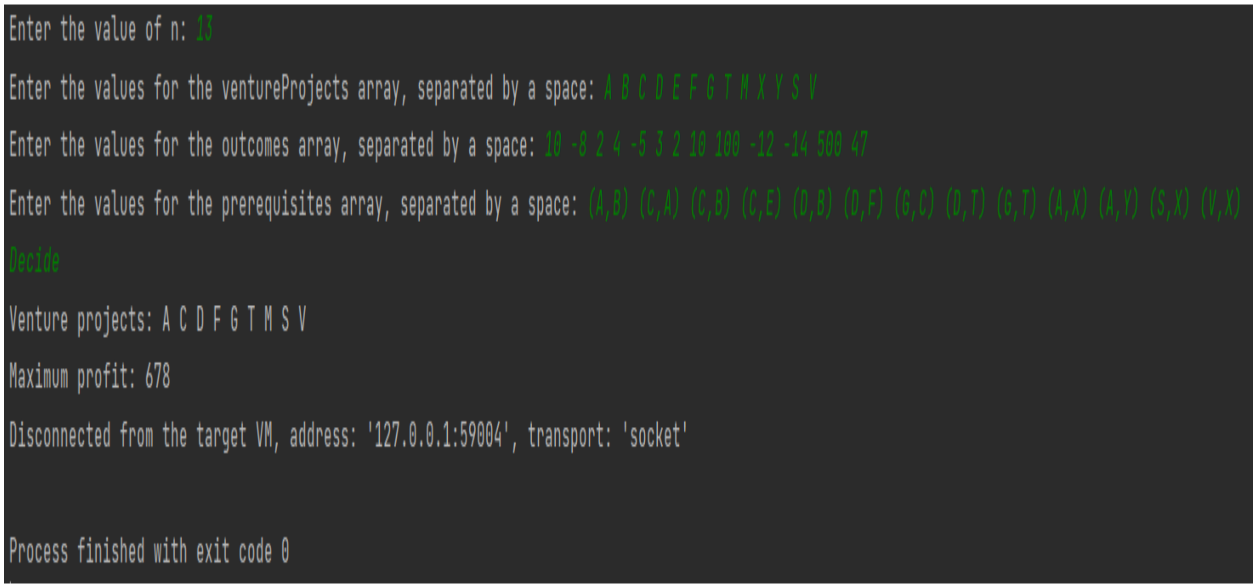
```
Enter the value of n: 11
Enter the values for the ventureProjects array, separated by a space: A B C D E F G T M X Y
Enter the values for the outcomes array, separated by a space: 10 -8 2 4 -5 3 2 10 100 -12 -14
Enter the values for the prerequisites array, separated by a space: (A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T) (A,X) (A,Y)
Decide
Venture projects: A C D F G T M
Maximum profit: 131
Disconnected from the target VM, address: '127.0.0.1:58997', transport: 'socket'

Process finished with exit code 0
```

****

TEST INPUTS 4 >>

13
A B C D E F G T M X Y S V
10 -8 2 4 -5 3 2 10 100 -12 -14 500 47
(A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T) (A,X) (A,Y) (S,X) (V,X)
Decide

****

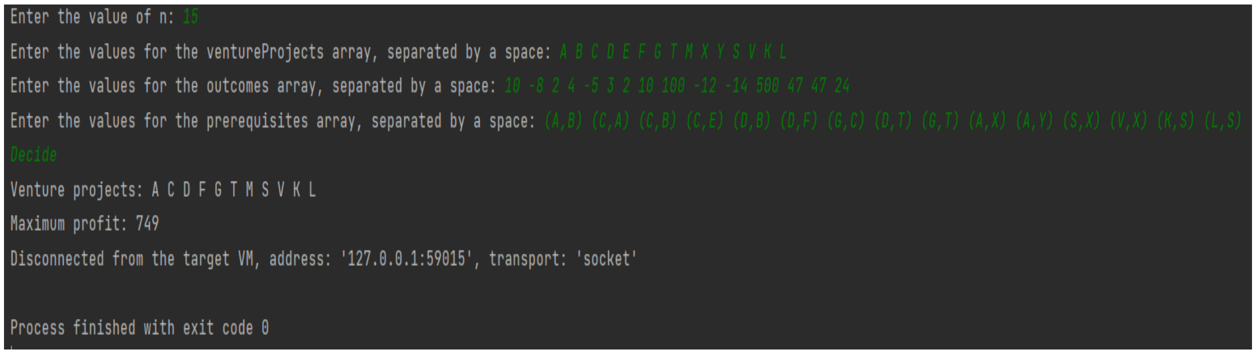


```
Enter the value of n: 13
Enter the values for the ventureProjects array, separated by a space: A B C D E F G T M X Y S V
Enter the values for the outcomes array, separated by a space: 10 -8 2 4 -5 3 2 10 100 -12 -14 500 47
Enter the values for the prerequisites array, separated by a space: (A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T) (A,X) (A,Y) (S,X) (V,X)
Decide
Venture projects: A C D F G T M S V
Maximum profit: 678
Disconnected from the target VM, address: '127.0.0.1:59004', transport: 'socket'


Process finished with exit code 0
```

****

TEST INPUTS 5 >>

15
A B C D E F G T M X Y S V K L
10 -8 2 4 -5 3 2 10 100 -12 -14 500 47 47 24
(A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T) (A,X) (A,Y) (S,X) (V,X) (K,S) (L,S)
Decide
****



```
Enter the value of n: 15
Enter the values for the ventureProjects array, separated by a space: A B C D E F G T M X Y S V K L
Enter the values for the outcomes array, separated by a space: 10 -8 2 4 -5 3 2 10 100 -12 -14 500 47 47 24
Enter the values for the prerequisites array, separated by a space: (A,B) (C,A) (C,B) (C,E) (D,B) (D,F) (G,C) (D,T) (G,T) (A,X) (A,Y) (S,X) (V,X) (K,S) (L,S)
Decide
Venture projects: A C D F G T M S V K L
Maximum profit: 749
Disconnected from the target VM, address: '127.0.0.1:59015', transport: 'socket'

Process finished with exit code 0
```

References:

- chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://stanford.edu/~rezab/discrete/Notes/3.pdf
- https://www.hackerearth.com/practice/algorithms/graphs/minimum-cost-maximum-flow/practice-problems/
- https://www.youtube.com/watch?v=3LG-My_MoWc&ab_channel=TutorialsPoint
- https://www.w3schools.com/java/java_user_input.asp