Innovative Project Report

On

Voting System using 2-SAT Problem

Submitted towards the partial fulfillment of

the requirement for the award of the degree of

Bachelor of Technology

In

Information Technology

Submitted by

Pranjal Singla (2K20/IT/102)

and

Utkarsh Pandey (2K20/IT/155)

Under the Supervision

of

Dr. Jasraj Meena

Department of Information Technology

Delhi Technological University

Bawana Road, New Delhi- 110042

INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(FORMERLY Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE’S DECLARATION

We, Pranjal Singla & Utkarsh Pandey, Roll No(s). 2K20/IT/102 & 2K20/IT/155, students of B. Tech. in Information Technology, hereby declare that the project Dissertation titled **Voting System using 2-SAT Problem** which is submitted by us to the Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the mid-semester component evaluation, semester-3 of Bachelor of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any degree, Diploma Associateship, Fellowship or any similar title or recognition.

Place: Delhi　　　　　　　　　　　　　　　　Pranjal Singla

Date: November 2021　　　　　　　　　　　　Utkarsh Pandey

CERTIFICATE

We hereby declare that the project Dissertation titled “Voting System using the 2-SAT Problem” which is submitted by Pranjal Singla & Utkarsh Pandey, Roll No(s). 2K20/IT/102 & 2K20/IT/155, Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the mid-semester component evaluation, semester-3 of Bachelor of Technology, is the record of the project work carried out by the students under my supervision.

ACKNOWLEDGEMENT

We would like to sincerely thank our Data Structures faculty Dr. Jasraj Meena for his valuable guidance regarding the structure, format and other aspects of this report. We would also like to thank him for allowing us to write a report on this topic viz. 2-SAT Problem. We would also like to extend sincere gratitude towards our Vice Chancellor Mr. Yogesh Singh for allowing the students to improve their practical skills and innovative thinking with practical and crucial subjects like “Data Structures”.

We also express our gratitude towards our parents who made it possible for us to successfully complete this report. It would not be possible for us to engage in this endeavour productively and efficiently without their constant support and guidance.

ABSTRACT

We used the solving approach to the 2-SAT problem, using Graphs and Satisfiability, and tried to modify it to make a potentially better and fair voting system. In the current paradigm, voting mainly follows a majority vote approach, leaving the wishes of those in a minority unaddressed. The model we have come up with, takes the input number of choices for the people, the number of entities participating in the voting, and their respective preferences with regard to the choices provided. After taking this input, the program maximises the number of priorities met, while maintaining that **at least one** of the preferences of all are accepted and worked upon if possible. In a case where such an arrangement is not possible, we keep note of the entities whose preferences were not met, and the governing body can then go ahead and release compensation for them. Considering this, we can view the votes as a Conjunctive Normal Form, or a Product of Sum and thus create a mathematical formula for the votes and the voters.

The solution is built based on the concept of implications in Boolean Satisfiability, and how a Conjunctive Normal Form can be converted to a series of implications. Moreover, we can model the implications thus formed into a Graph data structure, thus essentially creating a model of the voters and their priorities.

CONTENTS

CHAPTER 1: Introduction and Overview

1.1 Satisfiability:

Satisfiability is one of the most elementary parts of Formal Logic. A formula or statement is said to be Satisfiable if there exists **at least one** such model wherein the given statement holds true. This is in sharp contrast to Validity where a statement is only valid if the statement holds true for all the models or interpretations.

1.2 The Boolean Satisfiability Problem:

Also known as the K-Sat problem, The Boolean Satisfiability Problem aims to find a model or interpretation for a set of given statements such that the resultant is satisfiable. In other words, the system aims to find a set of values for the variables such that the given statement becomes true.

While the solution for this problem seems very intuitive to the human mind for smaller values, as the number of variables increases, the complexity shoots up exponentially, making the resolution highly complex.

For a clear understanding of what is happening, let's take a real-life problem that we will solve using the K-Sat problem:

Let's say we have a small city with around a million families, and we want to release a new development plan for the people. Now, this plan may include variables like, making a spare patch of land a concert ground. Now, there may be some people who would love a new concert ground while others who are strictly against the same. Similarly, let's say there are m options for the people.

Now, we ask each family to fill in their priorities of whether they want a proposal 'x' to be included in the plan or not. Obviously satisfying each and every demand of everyone is generally impossible, so we strive to find such an arrangement where **at least one** condition put forward by each family is satisfied.

Such a situation can be modelled in terms in terms of a **Conjunctive Normal Form** (CNF):

x1 x2 x3 ······∨xm (x1x2x3······xm)   -...n times

And is called the K-Sat problem where K is the number of Disjunctions in a single Conjunction term.

Such versions include the 1-SAT Problem which is fairly easy, 2-SAT which we hope to use for completing our aim of building a model to solve the problem and so on.

For the sake of simplicity and realistically implementing this problem, we have reduced the constraints and made every family enter exactly 2 choices for the options.

Thus, reducing to problem to:

(x1x4)(x2x5)(x3x6) -...n times

Chapter 2: Solution using Graphs:

The CNF obtained above can be converted into an Implication Graph which then translates into a Directed Graph using the following way:

$$A \vee B \equiv (A \Rightarrow B)\ \ (B \Rightarrow A)$$

Here, we can see that there is a directed edge from B to A and from A to B which in turn forms our directed graph.

We say that if any option xi and it’s conjugate xi lie in the same Strongly Connected Components (SCC) then the problem is Unsolvable meaning that we cannot find such an arrangement of the assignment of variables such that the following expression comes out to be true while also satisfying the constraints.

Now in real life ‘Unsolvable’ doesn’t make any sense but what it translates into is that there will be certain families who do not get their required choice in the final distribution, but for the sake of simplicity we have declared it as ‘Unsolvable’.

We are finding SCC’s using the classical Kosaraju’s Algorithm and then assigning each node it’s SCC number. This is done by constructing a normal and transpose graph simultaneously. We run one DFS on the normal graph and add elements into the stack in the reverse order. We then traverse each element in the stack and run a DFS on the transpose graph which numbering vertices according to their SCC number. Finally, based on the SCC numbers of a variable and its conjugate, we assign the values in our final distribution.

Thus, we reduce the time complexity from O(2mn) to O (n+m) which for at least for 1e6 can run in under 1 second for state-of-the-art applications.

Chapter 3: Motivation

The K-Sat problem has a vast range of possible applications, therefore, there is a lot of scope as far as applications are concerned. Whether it be the city plan as the case taken up by us, or a survey by a company all the way to making decisions in a class of 75 students, the K-Sat problem fits right in.

Moreover, such a problem with brute force would require O(2mn) which is highly impractical. For making an application that could possibly scale up to a million families, such a system which clearly fails. Thus, we tried to come with an algorithm to address such an issue using graph theory and satisfiability.

Chapter 4: BASIC THEORY

4.1 Graphs

A Graph is a non-linear data structure, and consists of a finite set of nodes or vertices. Graphs are further divided into Directed, Non-Directed, Weighted, Non-Weighted Graphs depending upon the type of edges incorporated in the structure.

4.2 Directed Graph

When a graph has an ordered pair of vertices, it is called a directed graph. The edges of the graph represent a specific direction from one vertex to another. When there is an edge representation as $u \to v$, the direction is from u to v. The first element u is the initial node or the start vertex. The second element v is the terminal node or the end vertex.

4.3 Transpose of a Graph

Transpose of a directed graph G is another directed graph on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G. That is:

If G contained an edge $u \to v$,

Then the Transpose G' will contain an edge $v \to u$, and vice versa.

4.4 Storing edges

Here, we have used an adjacency list to store the edges between the nodes. The list is just a space optimised version of the adjacency matrix where instead of assuming that the given graph is dense and there is an edge between every pair of nodes, we use a dynamically allocated array to store the neighbouring edges for a certain node which can be achieved using a vector in C++.

Chapter 5: Methodology

5.1 Handling negative Nodes

For the problem at hand, since any family can possibly enter information like:

+7 -2, we would need to somehow store -2 as a node in our adjacency list.

This is not possible as the indexing in our array by convention starts from 0, hence we came up with an alternative approach to address this issue, we are storing 2 nodes for every absolute value of number entered. Formally:

For every node m we are inserting:

2m for m,

2m+1 for -m.

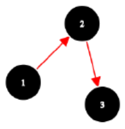
Doing this, we essentially store all the positive nodes in even numbers, i.e. as $2m$, while the negative nodes are stored in odd numbers, i.e. $2m+1$, thus solving the issue of negative nodes.

5.2 Depth First Search (DFS)

As the name suggests, DFS is a form of Graph traversal wherein we explore the depths of the graph from a given node first.

Depth first traversal or Depth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure. We start from a node and visit one of its neighbours and so on recursively till we visit all the nodes of the source. The time complexity of such a traversal is $O(n+m)$, where n is the number of nodes and m is the number of edges.

5.3 Departure Time of a Node



Let’s say we have this graph, and we have to find the Finishing Time/Departure Time for each vertex.

The Departure Time for each vertex is defined as the time when the DFS last backtracks to that vertex during a traversal.

For this particular graph, the departure times are, assuming the time started from 0:

Node No.	Departure Time
1	6
2	5
3	4

5.4 Topological Order

A topological sort or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge u → v from vertex u to vertex v, u comes before v in the ordering. For instance, the vertices of the graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another; in this application, a topological ordering is just a valid sequence for the tasks. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG).

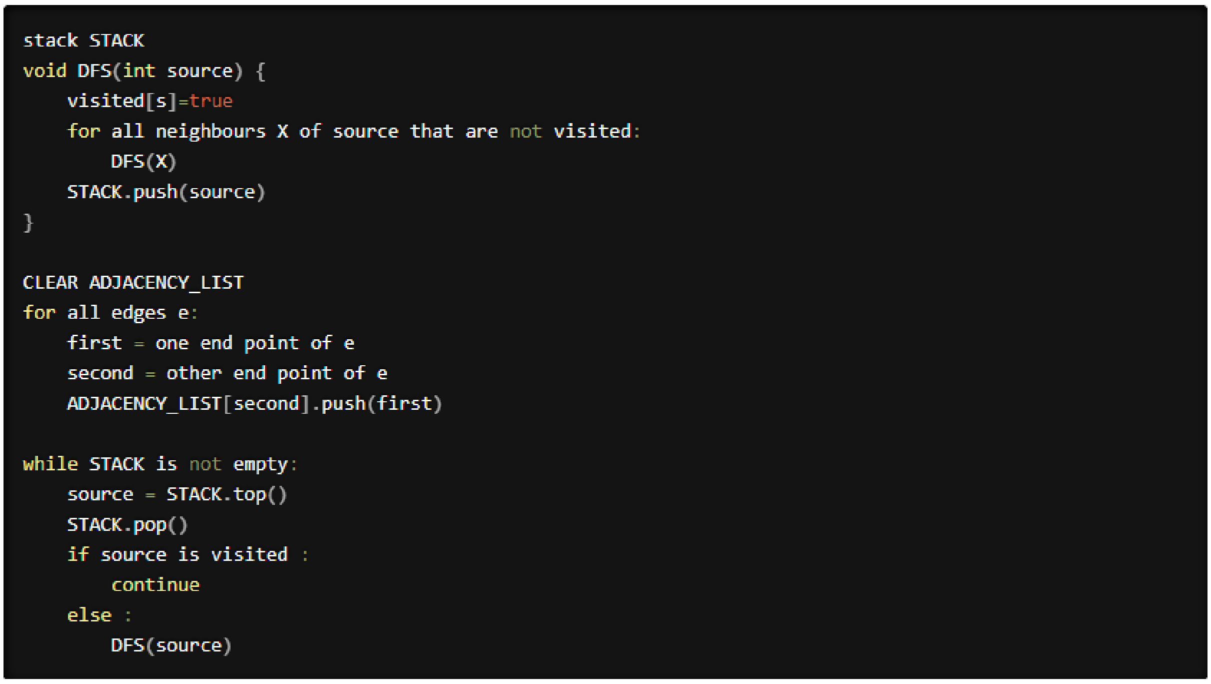
The Topological Ordering of the graph prints the vertices in order of their decreasing departure time calculated with the help of a DFS.

5.5 Strongly Connected Components (SCC)

A graph is said to be strongly connected if every vertex is reachable from every other vertex. The strongly connected components of an arbitrary directed graph form a partition into subgraphs that are themselves strongly connected.

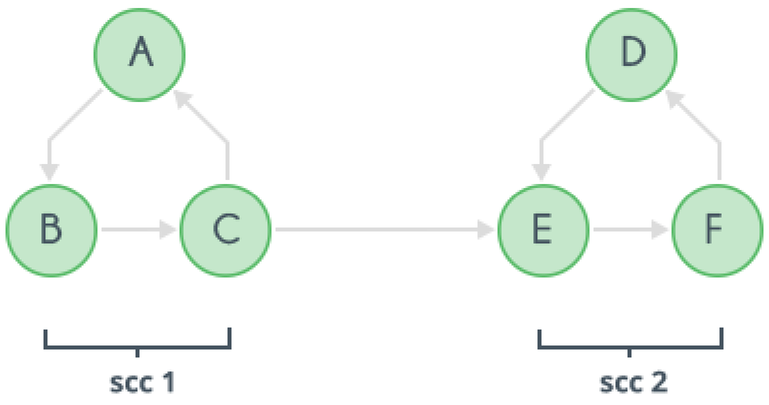
5.6 Kosaraju’s Algorithm

The Kosaraju algorithm is a DFS based algorithm used to find Strongly Connected Components (SCC) in a graph. It is based on the idea that if one is able to reach a vertex v starting from vertex u, then one should be able to reach vertex u starting from vertex v and if such is the case, one can say that vertices u and v are strongly connected - they are in a strongly connected subgraph.

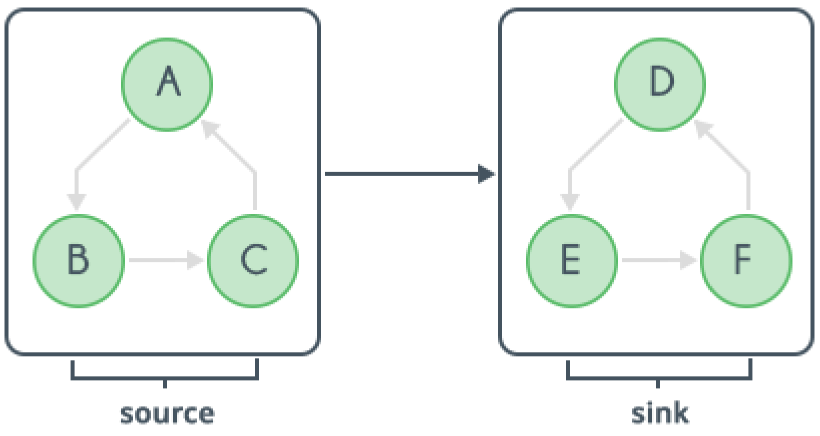


```
stack STACK
void DFS(int source) {
    visited[s]=true
    for all neighbours X of source that are not visited:
        DFS(X)
    STACK.push(source)
}

CLEAR ADJACENCY_LIST
for all edges e:
    first = one end point of e
    second = other end point of e
    ADJACENCY_LIST[second].push(first)

while STACK is not empty:
    source = STACK.top()
    STACK.pop()
    if source is visited :
        continue
    else :
        DFS(source)
```

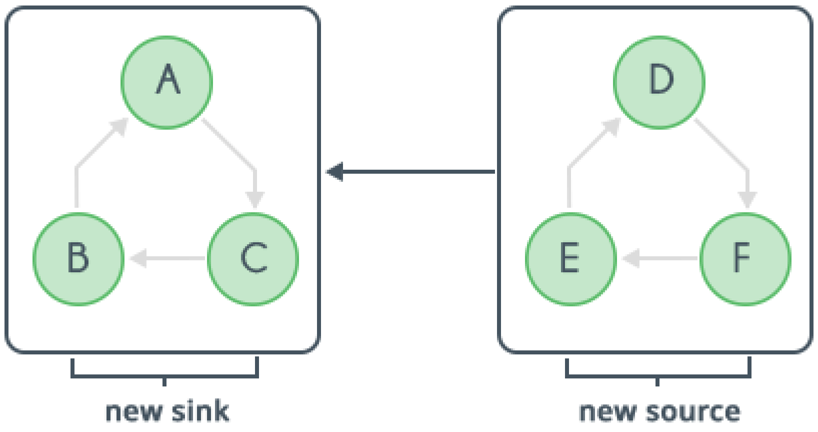
Consider the following graph:



It has two strongly connected components: scc1 and scc2.
Upon performing the first DFS with scc1 as the source, we get the following scenario:



- Upon reversing the graph and performing DFS again with scc2 as the source, we get the following scenario:



We infer that after both the DFS passes, the strongly connected components are clustered together.

Chapter 6: Procedure

We employ all the above-mentioned techniques to obtain an answer which would be best explained with the help of an example.

Let n, the number of families be 3,

and let m be the number choices which are 5 for this example

let n choices be of the form:

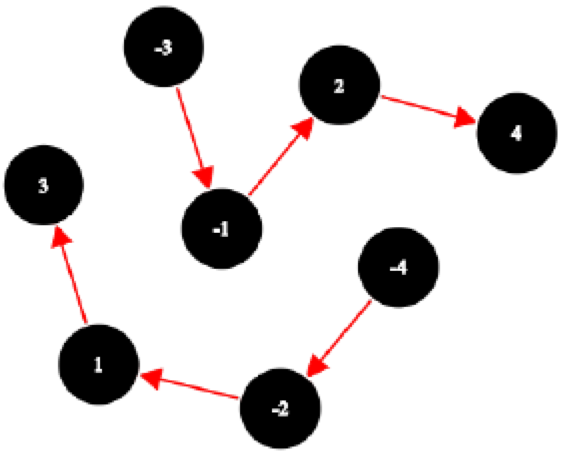
+ 1 + 2

- 1 + 3

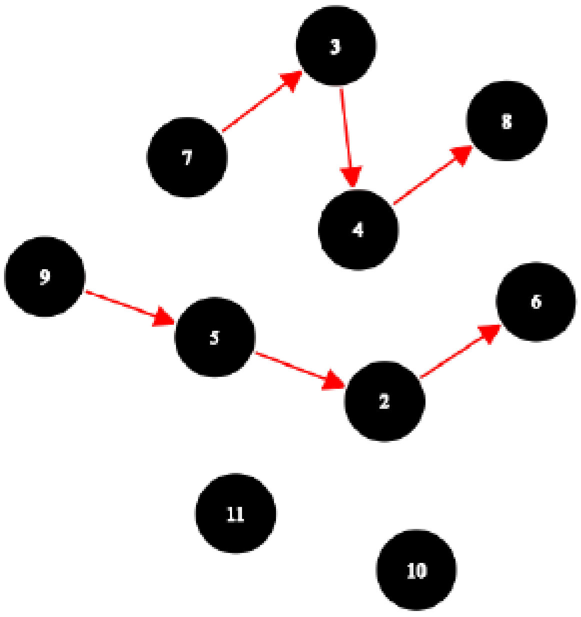
+ 4 – 2

Where + means to include a certain choice and – means to exclude a certain choice.

This is essentially a Conjunctive Normal Form which can be converted into a Directed Graph:



By scaling the nodes to avoid Negative values, we get:



Nodes 10, 11 are not picked by any family and hence are unused.

Now marking All the SCC Numbers of the nodes:

NODE NO.	POSITIVE NODE (+)	NEGATIVE NODE (-)
1	9	7
2	8	6
3	10	5
4	4	3
5	2	1

Now for the most important part. If we take the above graph or any Directed Graph and represent each Strongly Connected Component by a single node, we get a Directed Acyclic Graph (DAG). Due to the manner Kosaraju's Algorithm is applied the resulting condensation graph with each SCC number is obtained in topological order.

Since topological order is actually like a dependency order, hence we experimented with the SCC number of the positive and negative values of a certain node and we found out that:

If SCC[a] > SCC[-a], it is better to include the specific node in our final distribution, else we do not include this node in our final distribution.

We just need to set a value to an element there (the first component in topological order) and propagate it (according to the logical implication operation and to the fact that if a = True you must set -a = False). If we try to propagate to an element that already has a true or false value (and it doesn't clash with the one we're trying to set) we can simply skip it, because it means we've already propagated from it.

6.1 Time Complexity

Through this project, we have managed to reduce the time complexity of such tasks from O (2mn) to O(n+m), which is a huge improvement.

Original time complexity, derived from the brute force approach, becomes exponential. Consider the 2 given options for all m choices, which makes the total number of possible priority arrangements 2^m. Now, since we have to make a total of n iterations for all n entities, checking to see if their priorities are met, we come to the expression (2mn).

On the other hand, in our case, since we are doing a DFS call on a graph built by modelling these choices, entities and priorities, we can solve the same problem in O(n+m).

Chapter 7: Conclusion

We set out with the objective to find an optimum distribution for a given set of constraints using graphs and satisfiability. Having combined the knowledge from the subjects of Data Structures and Discrete Structures, we managed to achieve this goal. The project can successfully find such distributions in Linear Time, while the general methods to find such a solution was exponential.

The Project can be used for any form of a voting mechanism to satisfy constraints of a majority of people, thus assisting and potentially improving all administrating bodies.

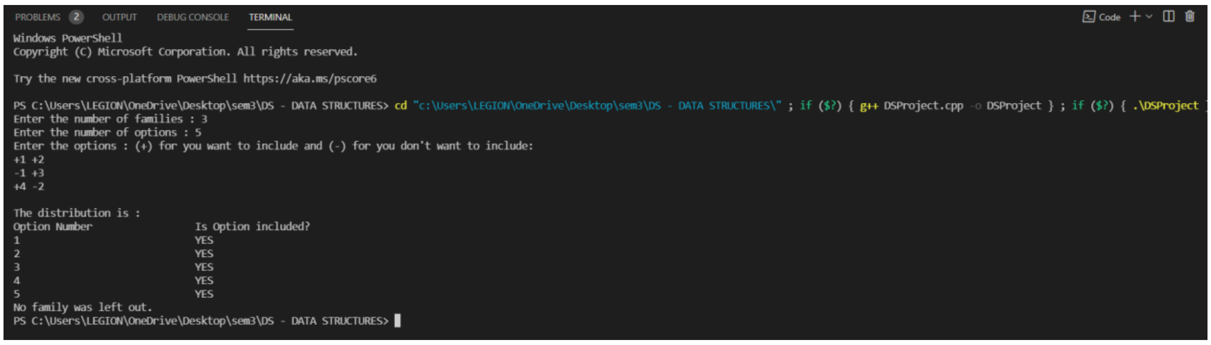
Code:

GITHUB LINK: -

https://github.com/utkarsh06P/Data_Structure_Final_Code/tree/main

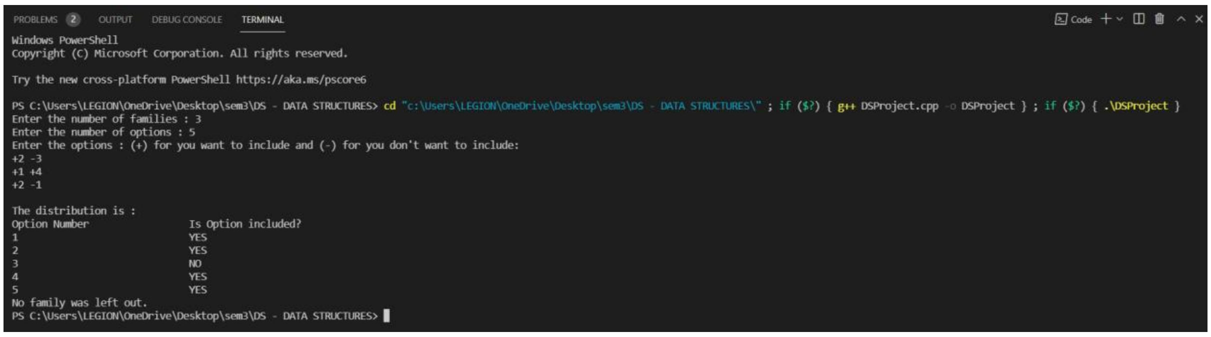
Output Screen:

1) When all the options are considered, no family is left out



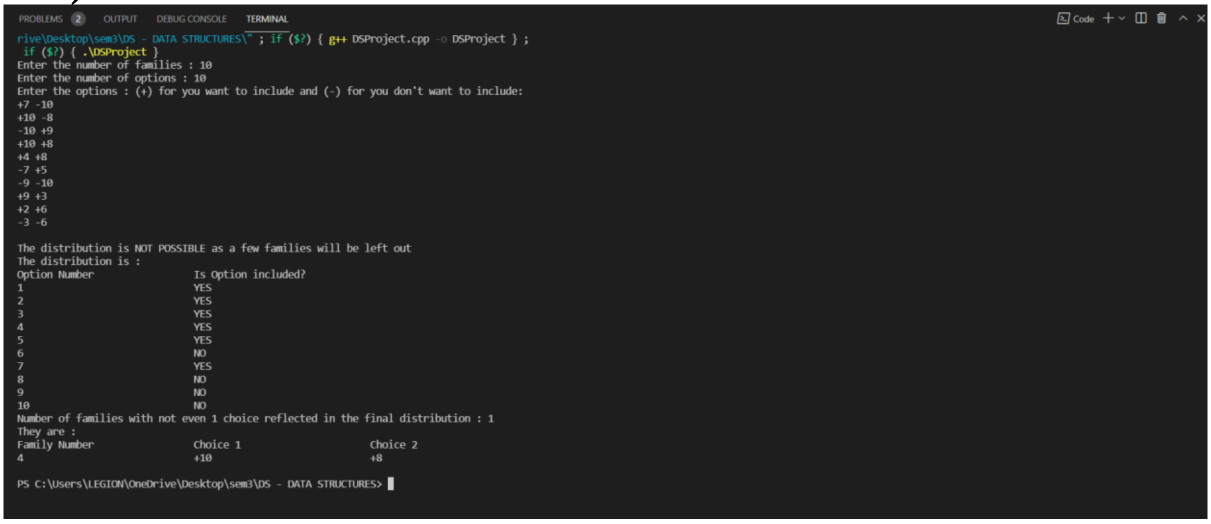
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES> cd "c:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES\" ; if ($?) { g++ DSProject.cpp -o DSProject } ; if ($?) { .\DSProject
Enter the number of families : 3
Enter the number of options : 5
Enter the options : (+) for you want to include and (-) for you don't want to include:
+1 +2
-1 +3
+4 -2

The distribution is :
Option Number                   Is Option included?
1                               YES
2                               YES
3                               YES
4                               YES
5                               YES
No family was left out.
PS C:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES>
```

2) When all the options are not considered, no family is left out



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES> cd "c:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES\" ; if ($?) { g++ DSProject.cpp -o DSProject } ; if ($?) { .\DSProject }
Enter the number of families : 3
Enter the number of options : 5
Enter the options : (+) for you want to include and (-) for you don't want to include:
+2 -3
+1 +4
+2 -1

The distribution is :
Option Number                   Is Option included?
1                               YES
2                               YES
3                               NO
4                               YES
5                               YES
No family was left out.
PS C:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES>
```

3) When few families are left out



```
rive\Desktop\sem3\DS - DATA STRUCTURES\" ; if ($?) { g++ DSProject.cpp -o DSProject } ;
 if ($?) { .\DSProject }
Enter the number of families : 10
Enter the number of options : 10
Enter the options : (+) for you want to include and (-) for you don't want to include:
+7 -10
+10 -8
-10 +9
+10 +8
+4 +8
-7 +5
-9 -10
+9 +3
+2 +6
-3 -6

The distribution is NOT POSSIBLE as a few families will be left out
The distribution is :
Option Number                   Is Option included?
1                               YES
2                               YES
3                               YES
4                               YES
5                               YES
6                               NO
7                               YES
8                               NO
9                               NO
10                              NO
Number of families with not even 1 choice reflected in the final distribution : 1
They are :
Family Number                   Choice 1                        Choice 2
4                               +10                             +8

PS C:\Users\LEGION\OneDrive\Desktop\sem3\DS - DATA STRUCTURES>
```

Chapter 8: Bibliography:

1. https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#3-satisfiability
2. https://en.wikipedia.org/wiki/2-satisfiability
3. https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm
4. https://en.wikipedia.org/wiki/Strongly_connected_component
5. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
6. https://cp-algorithms.com/graph/2SAT.html
7. https://www.geeksforgeeks.org/iterators-c-stl/
8. https://www.geeksforgeeks.org/2-satisfiability-2-sat-problem/
9. https://cses.fi/problemset/task/1684
10. https://codeforces.com/problemset/problem/1215/F