Design and Analysis of Algorithms course Project

Brahma Kulkarni (IMT2017011) and Ananya Appan (IMT2017004)

1 Problem Statement

The problem we've selected is **Cycle Edges**. In this problem we are given a graph G(V,E), we have to select a set of edges, say 'S', such that for any cycle C in G, there is at least one edge of the cycle which belongs to S. Furthermore, we have to optimize the set of edges chosen such that the sum of weights of edges in S is minimum. In other words, find S such that

$$\forall C \in G, \exists e \in C \; s.t \; e \in S \; and \; \sum_i w(e_i) \; is \; minimized \; .$$

2 Sources

We only referred to our Data Structures and Algorithms course notes and previously written codes.

3 Data Structures Used

Our problem required no specific data structures. However, we implemented a graph and represented it using an edge list. The edge list itself was a vector of tuples, each tuple depicting an edge.

4 Introduction

4.1 Algorithm Used

Kruskal's algorithm to find the minimum spanning tree of a graph considers edges in the ascending order of their weights. In order to find the maximum spanning tree of a graph, we consider the edges in the decreasing order of their weights. Hence, we call our algorithm ***reverse-kruskal.***

4.2 Terminologies Used

1. **G = (V,E)**: the input Graph with vertex set V and edge set E.
2. **MST**: Maximum Spanninng Tree
3. **S**: The set of edges which belong to our solution set.
4. Let G have k connected components. Let them be $G_1, G_2, ..., G_k$. Let MST_i be the corresponding MST of G_i. We define $\boldsymbol{X}$ to be the union of all such MSTs i.e. $\boldsymbol{X} = \cup_i MST_i$

4.3 Solution Idea

If all the edges in S are removed from G, what we will be left with is either a tree or a forest, depending on the number of connected components in G. This can be proved as follows:

Let S^C be the remaining set of edges in G after removing S from it. Assume, for contradiction that at least one of the remaining components is not a tree. Since our graph is undirected, S^C must contain a cycle, say C. Thus,

$$\forall edges\ e \in C, e \in S^C \Rightarrow !(\exists e \in C\ st\ e \in S)$$

No edge of the cycle will belong to S, thus rendering S invalid.

In order to minimize the total weight of S, we need to maximize the total weight of edges in these residual trees (forest). To do this, we need to generate maximum spanning trees. All the edges that aren't part of any MST will be put in S, giving us the desired output.

5 Pseudo-Code

As we only use one function called reverse-kruskal, the pseudo code of reverse-kruskal is given bellow:

reverse-kruskal:
sort edges in descending order of weights
run the standard kruskal procedure on this sorted edge set
all edges that aren't part of X are added to S, which is our solution

6 Analysis of running time

- Initializing all global variables, vectors and vectors of vectors takes $O(|V|)$ time.
- In reverse-kruskal, sorting the edges in descending order takes $O(|E|log(|E|))$ time.
- After this, we iterate over each edge and in the worst case, for each edge, we have to change the colour of $|V|$ number of edges. Hence, this takes $O(|E||V|)$ time.
- Hence, the total running time is $O(|E|log|E| + |E||V|)$.

7 Proof of Correctness

7.1 *Claim 1: X and S form a partition*

According to our algorithm, if two vertices are coloured differently, the edge joining them can be added to a maximum spanning tree, and hence, X. After this is done, all the vertices in the corresponding MST at that point of time are coloured with the same colour. Hence, at a later point, if the two vertices under consideration are of the same colour, adding the edge between would form a cycle. Such edges are added to S. Thus,

$$X \cup S = E\ and\ X \cap S = \phi \tag{1}$$

7.2 *Claim 2: For every cycle in the graph, there exists at least one edge belonging to C that belongs to S*

$$\forall C,\ \exists e \in C\ s.t.\ e \in S \tag{2}$$

Proof: Let C belong to the i^{th} connected component of G, say G_i and let MST_i be the corresponding MST of G_i.

$$MST_i\ contains\ no\ cycle \Rightarrow \exists e \in C\ s.t\ e \notin MST_i \Rightarrow e \in S \quad (from\ 1)$$

7.3 *Claim3: Number of edges in S is fixed*

Proof: Let G_i have n_i vertices. Let MST_i correspond to G_i hence, MST_i has $n_i - 1$ edges. From 1,

$$X \cup S = E$$

$$X \cap S = \phi$$

$$\sum_i (n_i - 1) + |S| = |E| => |S| = |E| - \sum_i (n_i - 1) = |E| - \sum_i n_i + \sum_i 1 = |E| - |V| + k$$

$$|S| = |E| - |V| + k = constant \tag{3}$$

7.4 *Claim4: Forming Maximum Spanning Trees results in minimizing the total weight of edges in S, i.e., S is the optimal solution*

Proof: Let S, consisting of l edges be represented as $S = e_1, e_2, ..., e_i, ..., e_l$. Consider any edge e_i. Let $X = e_1, e_2, ..., e_{i'}, ..., e_{|V|-k}$. Assume , for contradiction, that our solution S is not optimal.
From Claim 3, $|S|$ is fixed. Thus, some edges from S must be replaced with some other edge from X, say e_i' . e_i' must also be replaced with e_i . Let our new solution be $S' = e_1, ..., e_i', ...e_l$. There are 2 cases:

Case 1: e_i was considered before e_i' while running reverse-kruskal.

Since we consider edges in decreasing order of their weights, $w(e_i') > w(e_i)$. But, e_i was not added into X. Thus, e_i formed a cycle with the edges added into the MST at that point. Thus, now, one of the MSTs, say MST_j has a cycle. Hence, our solution S' is not valid.

Case 2: e_i was considered after e_i' while running reverse-kruskal

$w(e_i) > w(e_i')$. Let s be the sum of weights in S, and s' be the sum of weights in S'.

$$s = w(e_1) + w(e_2) + ... + w(e_i) + ... + w(e_l)$$

$$s' = w(e_1') + w(e_2') + ... + w(e_i') + ... + w(e_l')$$

$$\Rightarrow s - s' = w(e_i) - w(e_i') < 0$$

$$\Rightarrow s < s'$$

Thus, our previous solution was better.

8 Contributions by team members

Both of us thought of the solution idea and proof of correctness together. Individual contributions are:

- **Brahma**: Visualizing the graph after removing set of edges S, identifying X and S are partitions, Claim 2, Claim 3, wrote the code, gave outline on content of README.
- **Ananya**: Identifying remaining graph as MST, Claim 1, Claim 4, guided Brahma in writing the code , generated large testcases, wrote the final README

9 Instructions on how to compile and run the code

- Our code was completely written in $C++$. Hence, the command to compile it is: **g++ code.cpp**
- To run the code and read the input directly from some pre-defined test cases generated by us, use the following command (suppose you want to use the test case **input_2_cycles.txt**): **./a.out < Testcases/input_2_cycles.txt** (this is called piping)
- To run the code, the command is: **./a.out**. The input format is specified in **Testcases/format.pdf**.
- Additional information about the testcases pre-defined by us is given in **Testcases/format.pdf**