Application of Erdős–Gallai Theorem to Validate Degree Sequence of A Simple Graph

Kelvin Valensius (13511009)
Informatics Engineering
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511009@std.stei.itb.ac.id

Abstract*—In Computer Science, we often meet problems that contain graph theory. This paper will give a sample problem of graph theory and how to solve it. This paper is divided into several sections. Section I will introduce the paper. Section II will present the problem. Section III is divided into 5 section. Section A will talk about some false approaches to solve the problem. Section B will talk about Erdős–Gallai Theorem statement. Section C will discuss about the proof of the theorem. Section D will discuss about how to implement the theorem. Section E will discuss about the complexity of the algorithm. Section IV will show the implementation of the algorithm in the form of code in C++. Section V will conclude the paper.

***Index Terms*— Erdős–Gallai Theorem, degree sequence, simple graph**

I. Introduction

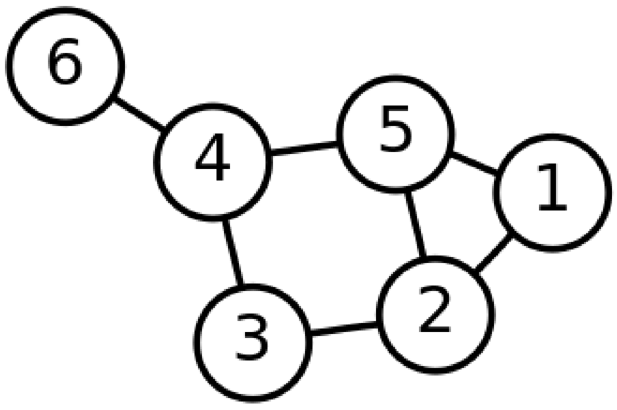


Fig. 1.1 Graph

Study of graph, also known as graph theory, is one of the majors in Discrete Mathematics. It also has many applications in Computer Science. Graph is a collection of vertices and edges connecting some subset of them [1]. As we can see in Fig. 1.1, the circles with number inside them are the vertices and the lines that connected the circles are the edges.

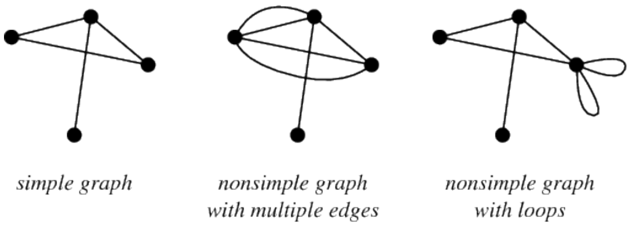

simple graph
nonsimple graph with multiple edges
nonsimple graph with loops

Fig. 1.2 (a) simple graph; (b) non-simple graph with multiple edges; (c) non-simple graph with loops

There are many kinds of graph and the one that will be discussed is simple graph. Simple graph is an undirected and unweighted graph that has no loop (an edge that connects a vertex to itself) and no multiple edge (two vertices that are connected by more than one edge) [2]. As we can see in Fig. 1.2, part (a) shows sample of a simple graph and part (b) and (c) show sample of non-simple graph.

To represent graph, There are many kinds of way. The one that will be discussed is by using degree sequence. Degree sequence is a non-negative and non-increasing sequence of the vertex degrees (number of edge that connected to the vertex) of its graphic vertices [3].

In this paper, the author will discuss how to validate degree sequence of a simple graph. There are some theorems that can be used to do this, but the author will only discuss about Erdős–Gallai Theorem.

II. Problem

As mentioned before, the author will discuss how to validate degree sequence of a simple graph. The source of this problem is from [4]. The problem states that given **n** numbers of integers, find out whether this **n** number of integers can represent the degrees of **n** vertices of a simple graph or not. Given that the constraints of the problem are :

- $1 \leq \mathbf{n} \leq 10000$.
- Time limit : 2 seconds (in this paper, we will assume that a computer can run 100000000 processes in 1 second).

For example, given **4** integers, 3, 3, 3, and 3. Then it is possible to represent the degree of **4** vertices of a simple graph (it is a complete graph of 4 vertices). Another example, given **6** integers, 2, 4, 5, 5, 2, and 1 (it does not have to be non-increasing sequence because it is not a degree sequence after all). Then it is impossible to represent the degree of **6** vertices of a simple graph with that configuration.

III. Solution

A. Some False Approaches

First, the author will discuss some approaches that fail to solve the problem and why they fail. First approach that comes to mind is using Greedy approach. To do this, we have to sort all **n** integers into non-increasing order. Define the value of each number is a_i. For each a_i that is positive, we decrease a_i smallest number that is positive and not a_i itself by 1, then a_i become 0. If you cannot find a_i smallest number for each iteration, then it must be impossible to represent the degree of **n** vertices of a simple graph with that configuration. If after the iteration all **n** integers become zero, then it is possible, and vice versa.

This approach seems promising, since it could solve all examples that is given in section II correctly and also it runs quite fast because the complexity is $O(N^2)$ (it meets the constraint requirements). Unfortunately, there are some corner cases that can make this approach fails. For example, given **6** integers, 4, 4, 4, 3, 2, and 1. This approach will state that it is impossible instead of possible (the correct result). This is why this approach fails.

There is another approach that comes to mind, that is using Complete Search. To do this, we try all the combinations of pairing the connected vertices. If there is a valid combination then it is possible, and vice versa. This approach will always give a valid answer. But the complexity is $O((N!)^N)$ and it is way too big. This is why this approach fails too.

B. Erdős–Gallai Theorem

In this section, the author will discussed about Erdős–Gallai Theorem. This theorem state that a non-negative and non-increasing sequence $\{d_1, ..., d_n\}$ can be represented as degree sequence of a simple graph with n vertices if and only if the sum of vertex degrees is even and

$$\sum_{i=1}^{r} d_i \le r(r-1) + \sum_{i=r+1}^{n} \min(r, d_i) \ (eq. 3.1)$$

holds for $1 \le r \le n-1$ [5]. The proof of this theorem will be discussed in the next section.

C. Proof

In this section, the author will explain why this theorem could solve the problem above. From the Erdős–Gallai Theorem statement above, we get 3 requirements for a sequence to be represented as degree sequence of a simple graph, they are :

- The sequence must be non-negative and non-increasing.
- The sum of all numbers in the sequence must be even.
- The sequence must meet eq. 3.1 that holds for $1 \le r \le n-1$

The first requirement is trivial, it has to be done because degree sequence is a non-negative and non-increasing sequence of the vertex degrees (number of edge that connected to the vertex) of its graphic vertices [3]. The second requirement is the handshaking lemma. Handshaking lemma is the statement that every finite undirected graph has an even number of vertices with odd degree. This statement implies that the sum of all degrees is even. This is why the second requirement is needed.

From eq. 3.1, we can assume that S is the set of vertices from a simple graph and A is the subset of S that contains only r vertices that have highest degree in S. As we can see, for all valid r, there are at most r(r-1) edges that formed within A, and for each vertex $v \notin A$, there are at most $\min(r, d_v)$ edges from v into A. And these statements meet the necessity of the last requirement.

D. Implementing The Theorem

To fulfill the first requirement, we have to check whether the sequence contain negative integer or not. The requirement meets if there is no negative integer. Then we have to sort the sequence into non-increasing order because the problem above does not state that the input sequence will be in non-increasing order. To fulfill the second requirement, we have to check the sum of all numbers in the sequence. The requirement meets if the sum is even.

The last requirement can be fulfilled using iteration. For each r that is valid, we check whether eq. 3.1 is true or not. If it is true for all possibilities of r, then it fulfills the requirement, and vice versa.

E. Complexity

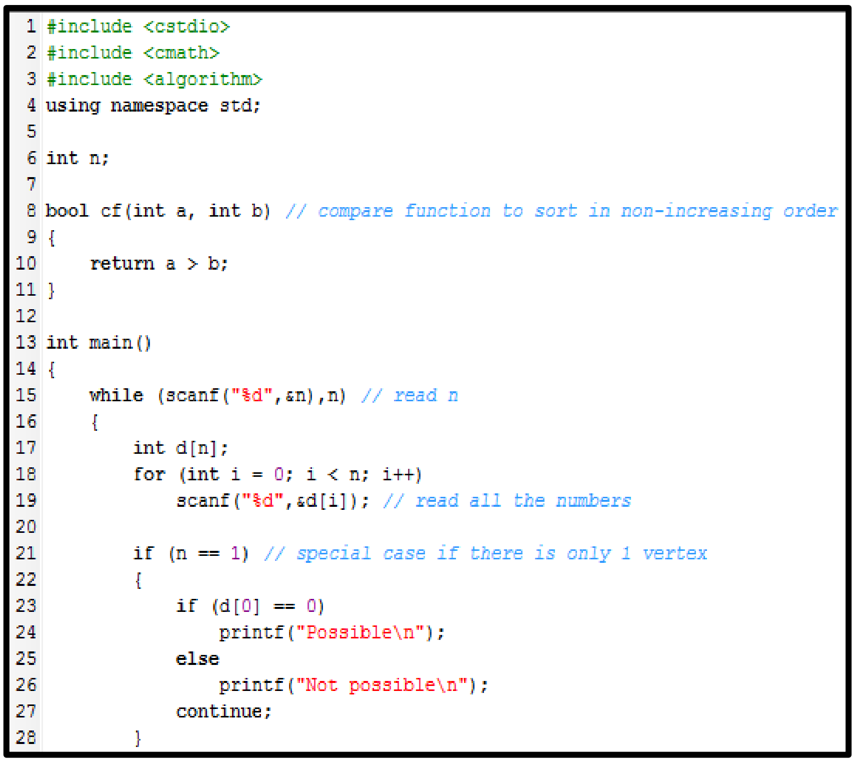
In the previous section, the author has shown the Erdős–Gallai Theorem, why it can solve the problem given above, and how to implement it. But, we haven't checked the complexity whether it will meet the requirement of the constraint given or not. So, in this section, the author will show how is the complexity of the implementation of this theorem.

To make it simpler, the author will divide the steps of the implementation of this theorem in 3 main steps, *precheck*, *sort*, and *validate the inequality*. In *precheck* step, we need to check whether the sequence is non-negative or not and whether the sum of all numbers in the sequence is even or not. To do this we can do linear search and the complexity is O(N). In *sort* step, we have to sort the sequence into non-increasing order. To do this we can use Standard Template Library that is available in C++ to sort the sequence and the complexity is $O(N \log N)$. For the last step, we can use prefix sum to improve the performance of the algorithm and it can be done before this step and the complexity is O(N). The prefix sum is used to calculate the left side of eq. 3.1. Then we iterate r from 1 to n-1, for each r, we calculate the right side using linear search and it costs O(N). So the last step complexity is $O(N^2)$. Total complexity of this algorithm is

$$O(f(N)) = O(N) + O(N \log N) + O(N) + O(N^2)$$
$$= O(N^2) \ (eq. 3.2)$$

From the equation above, we can see that this theorem meet the requirement of the constraint given. So this solution is correct.

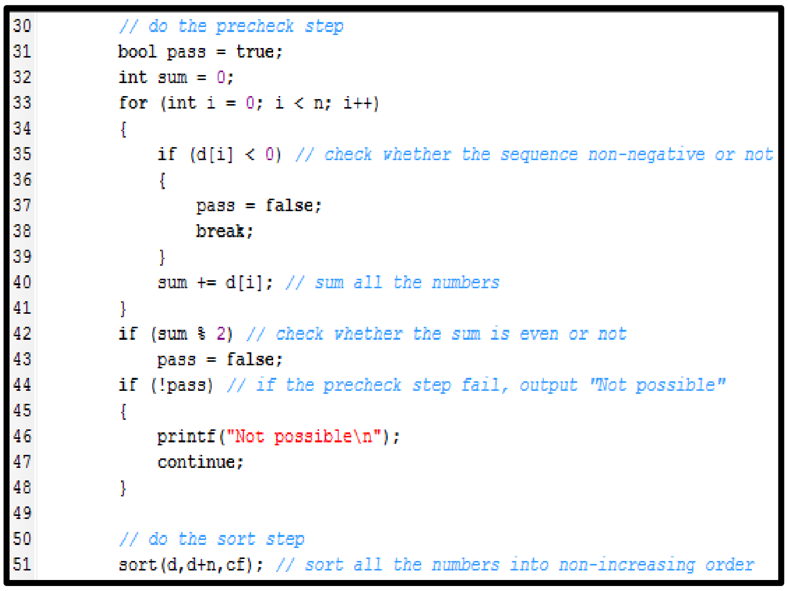
IV. CODE IMPLEMENTATION USING C++



```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

int n;

bool cf(int a, int b) // compare function to sort in non-increasing order
{
    return a > b;
}

int main()
{
    while (scanf("%d",&n),n) // read n
    {
        int d[n];
        for (int i = 0; i < n; i++)
            scanf("%d",&d[i]); // read all the numbers

        if (n == 1) // special case if there is only 1 vertex
        {
            if (d[0] == 0)
                printf("Possible\n");
            else
                printf("Not possible\n");
            continue;
        }
```

Fig. 3.1 Code Implementation Using C++ part 1

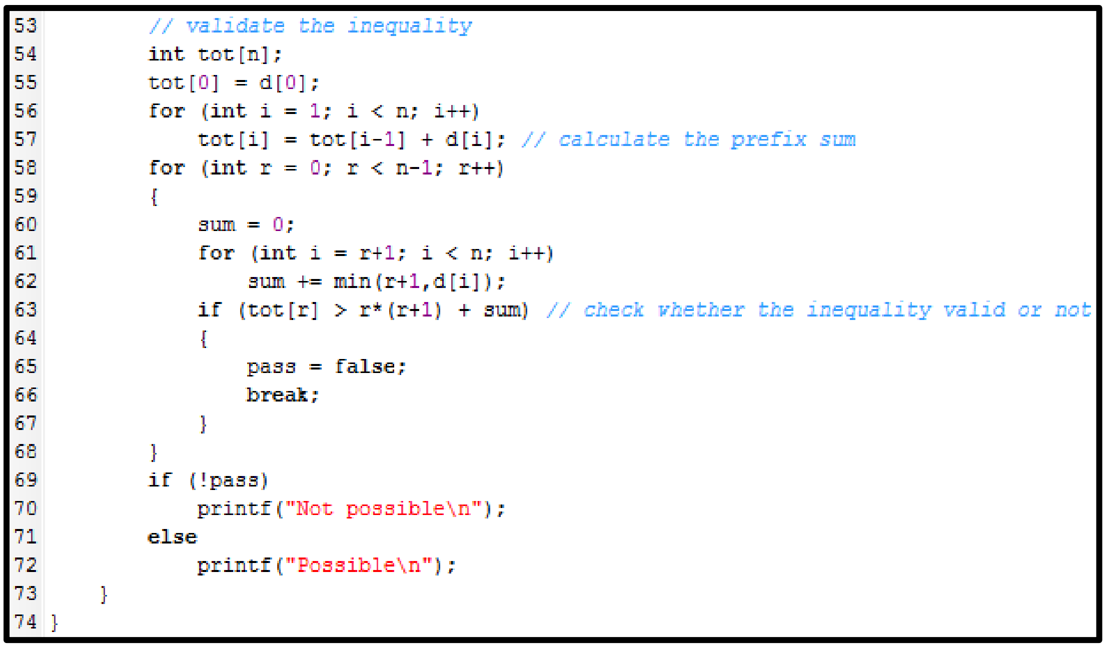
In Fig. 3.1, there are 3 main parts. The first part is the compare function that is used in sort procedure to sort in non-increasing order. The second part is reading input from standard input (stdin). The last part is where special case is handled. The special case happen if there is only 1 vertex, because Erdős–Gallai Theorem can't handle sequence with only 1 vertex.



```cpp
        // do the precheck step
        bool pass = true;
        int sum = 0;
        for (int i = 0; i < n; i++)
        {
            if (d[i] < 0) // check whether the sequence non-negative or not
            {
                pass = false;
                break;
            }
            sum += d[i]; // sum all the numbers
        }
        if (sum % 2) // check whether the sum is even or not
            pass = false;
        if (!pass) // if the precheck step fail, output "Not possible"
        {
            printf("Not possible\n");
            continue;
        }

        // do the sort step
        sort(d,d+n,cf); // sort all the numbers into non-increasing order
```

Fig. 3.2 Code Implementation Using C++ part 2

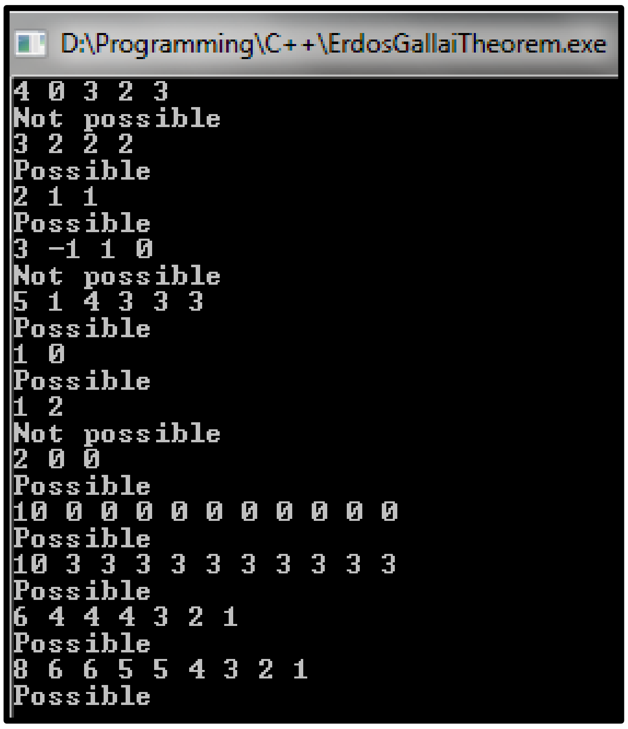
In Fig. 3.2, there are 2 main parts. The first part is the *precheck* step that has been explained in previous section. The last part is the *sort* step that has also been explained in previous section



```
        // validate the inequality
        int tot[n];
        tot[0] = d[0];
        for (int i = 1; i < n; i++)
            tot[i] = tot[i-1] + d[i]; // calculate the prefix sum
        for (int r = 0; r < n-1; r++)
        {
            sum = 0;
            for (int i = r+1; i < n; i++)
                sum += min(r+1,d[i]);
            if (tot[r] > r*(r+1) + sum) // check whether the inequality valid or not
            {
                pass = false;
                break;
            }
        }
        if (!pass)
            printf("Not possible\n");
        else
            printf("Possible\n");
    }
}
```

Fig. 3.3 Code Implementation Using C++ part 3

In Fig. 3.3, there is only 1 part, the *validate the inequality* step that has been mentioned before in previous section.



```
D:\Programming\C++\ErdosGallaiTheorem.exe
4 0 3 2 3
Not possible
3 2 2 2
Possible
2 1 1
Possible
3 -1 1 0
Not possible
5 1 4 3 3 3
Possible
1 0
Possible
1 2
Not possible
2 0 0
Possible
10 0 0 0 0 0 0 0 0 0 0
Possible
10 3 3 3 3 3 3 3 3 3 3
Possible
6 4 4 4 3 2 1
Possible
8 6 6 5 5 4 3 2 1
Possible
```

Fig. 3.4 Result test using author's tests

Fig. 3.4 shows the result test using author's tests. From the result we can see that the code implementation of Erdős–Gallai Theorem has given the right result.

This code has been successfully compiled and has been passed through tests that is conducted by UVa Online Judge [7].

V. CONCLUSION

The author has shown the algorithm that can be used to validate degree sequence of a simple graph, that is using Erdős–Gallai Theorem. The theorem has been proven and it meets all the requirement of the constraint given. The complexity of the theorem is $O(N^2)$.

REFERENCES

[1] http://mathworld.wolfram.com/Graph.html; Access date : December 15 2012.

[2] http://mathworld.wolfram.com/SimpleGraph.html; Access date : December 15 2012.

[3] http://mathworld.wolfram.com/DegreeSequence.html; Access date : December 17 2012.

[4] http://uva.onlinejudge.org/external/107/10720.html; Access date : December 17 2012.

[5] http://mathworld.wolfram.com/GraphicSequence.html; Access date : December 17 2012.

[6] Blitztein, J. and Diaconis, P. (2006). *A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees*. Standford University. 3-4.

[7] http://uva.onlinejudge.org/index.php; Access date : December 17 2012.

ACKNOWLEDGMENT

Kelvin Valensius, as the author of this paper, want to express his deepest gratitude to Dra Harlili, M.Sc and Dr. Ir. Rinaldi Munir, M.T. as the lecturers of IF2091 – "Struktur Diskrit". Special thanks to God, my family, and all my friends that supporting the making of this paper.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Desember 2012

[illegible]

Kelvin Valensius (13511009)