Topological Sort Using Graph Theory

Hafidh Rendyanto 13517061
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
hafidhrendyanto@s.itb.ac.id
13517061@std.stei.itb.ac.id

***Abstract*—In this paper we discuss the use of graph theory to implement Topological Sort algorithm, a type of algorithm to create full relation from partial one. In order to do this, it create a graph from knowledge base that we give to it and then apply the said algorithm to that graph to make an ordered list.**

***Keywords*—Topological Sort, Sort Algorithm, Algorithm, Graph Theory.**

I. Introduction

In Computer Science, sorting algorithm is used in many different (and most of the times, diverse) application. It is used to find a solution to a problem, but most of the times, it is used to accelerate another algorithm like search algorithm (ex: binary search). But in this paper, we will focus our attention to a type of sort algorithm that usually used to deduce a solution to a problem (partial relation problem). But before we continue, I want to mention some other uses of sort algorithm.

1) Searching
   Like I have said in the preceding paragraph, sort algorithm is mostly used to accelerate sort algorithm in what known as search pre-processing in which, before we search a data for specific element, we sort the data, so we can use binary search (which have a complexity of O(log n), the lowest of its kind) when the need to search an element of this data arises. On a note, this use of sort algorithm is perhaps the single, most important application of sorting.
2) Closest Pair
   This problem, ask the question of, given a set of n number, how do you find the pair of numbers that have the smallest difference between them. The simplest of algorithm is to traverse that set, and for *every* element *x* in that set, you must traverse every element y of this list except x, compute the difference between x and y, and then find the minimum difference. Of course, this problem goes quadratically large the larger the size of that set ($O(n^2)$). One of the uses of sort algorithm to accelerate this problem is to pre-process the data using sort algorithm, so we need only traverse the data once, and then find the difference between the next and preceding element, this approach is going to reduce the closest pair problem to a complexity of (O(n)).
3) Element Uniqueness
   Another problem that can be optimized using search algorithm is Element Uniqueness, this problem asks the question of is there any duplicates in a given set of n items. Like before the simplest algorithm is to traverse that set, and for *every* element *x* in that set, you must traverse every element y of this list except x, compute the difference between x and y, and if there exist a pair of x and y which have a difference of 0, then that pair of x and y is not unique in that set. Just like before, this algorithm has complexity of $O(n^2)$ and one way to accelerate the problem is to pre-process the data so we just need to linearly search the data for element that is not unique after the pre-process.
4) Frequency Distribution
   This problem asks the question of, given a set of n items, which element occurs the largest number of times in that set. The optimization problem is very similar to the problem that I have mentioned before, If the items are sorted, we can seep from left to right linearly, counting them one by one, since all identical items will be lumped together.
5) Selection
   This problem asks the question of, what is the kth largest item in the set. If the keys are placed in sorted order in an array, the kth largest can be found in constant time by simply looking at the kth position of the set.

Over the course of history, humanity has made many kinds of sort algorithm, like bucket sort, bubble sort, selection sort, insertion sort, merge sort, and many other (which I will go through one by one in the next section). They are different in implementation, but they have one common characteristic, all of them can only sort object with a defining characteristic of conformity like, real number set $\mathbb{R}$ or alphabet. None of them can sort random item. For example, we cannot sort a set like {apple, banana, melon} because we don't know whether apple precede banana or the other way around.

But all hope is not lost, there *is* one sort algorithm (at least that I know) that can sort a set of random items, the sort algorithm that refer is topological sort. This sort algorithm take a set of partial relation of object as input, and then using that set as information, this algorithm, create a new set of ordered object. Our implementation of topological sort use graph to process the information using simple algorithm that I will explain shortly.

II. BRIEF EXPLANATION OF SORT ALGORITHM

A. Definition

The word algorithm is used widely in computer science, mathematics, and science in general. The formal definition of this word is that algorithm is an unambiguous specification of how to solve a class of problem. Algorithm can perform calculation, data processing, and automated reasoning.

Algorithm is used in every part of our daily life, from solving complex problem, proofing mathematical theorem, and even food recipes can be categorized as algorithm. But most importantly, algorithm gives the means to convey our ideas to other, and to make our life easier.

In this paper we are going to talk about the type of algorithm used in data processing, vis, sort algorithm. Sort algorithm itself is a type of algorithm that is used to make an ordered set of an unordered one. This type of algorithm has many applications like what I have described in the preceding chapter.

B. Brief History

Over the course of human history, the word has developed from precise definition of process to something more abstract in this day and age, the reason will be explained shortly. I will describe briefly about the history of the word below,

Ancient Greece

There are two examples of the use of this word in ancient Greece, vis, Sieve of Eratosthenes, which was described in Introduction to Arithmetic by Nicomachus, and the Euclidean algorithm, which was first described in Euclid's Elements.

The Advent of Algebra

The advent of algebra gives birth to the use of variable to solve mathematical problem, which also gives birth to algorithm to solve such problem.

Advancement of Mathematics

Algorithm is expanded in the field of mathematics to solve even more problem and even given its own place of study in Discrete Mathematic, a subset of math.

The Advent of Computer

In this event of human history, the use of the word is not expanded, it is rather, shrink to be used mainly in the field of computer science to describe a set of process that computer go through in order to solve problem.

Current Age

In this day and age, the word expanded again to not only describe precise description of some process but also to model how intelligence work in the field of artificial intelligence. Even if this definition of "algorithm" does not match precisely with artificial intelligence, but I still believe that even the most sophisticated machine intelligence (using neural net or some other kind of advanced implementation) could still be described using algorithm.

C. Classification

There are many ways to classify algorithm, each with their own odds and ends.

By design paradigm

Paradigm can be described as point of view, or way of thinking. With different point of view to a problem, one can create a completely different algorithm to a problem.

a) Brute-force
   This is the naïve method of trying every possible solution to see which is best.
b) Divide and Conquer
   A divide and conquer algorithm repeatedly reduces an instance of a problem to one or more smaller instances of the same problem (usually recursively) until the instances are small enough to solve easily. One such example of divide and conquer is merge sorting. Sorting can be done on each segment of data after dividing data into segments and sorting of entire data can be obtained in the conquer phase by merging the segments.
c) Search and Enumeration
   Many problems (such as playing chess) can be modeled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration and backtracking.
d) Randomized Algorithm
   Such algorithms make some choices randomly (or pseudo-randomly). They can be very useful in finding approximate solutions for problems where finding exact solutions can be impractical (see heuristic method below).
e) Reduction of Complexity
   This technique involves solving a difficult problem by transforming it into a better-known problem for which we have (hopefully) asymptotically optimal algorithms.
f) Backtracking
   In this approach, multiple solutions are built incrementally and abandoned when it is determined that they cannot lead to a valid full solution. One of its popular use is its use in logic programming

Optimization Problem

For optimization problems there is a more specific classification of algorithms

a) Linear Programing
   If the problem that is going to be solved have linear equity and inequality constraints, the constraint of this particular problem can be used directly in producing the optimal solution, such is the definition of this classification
b) Dynamic Programing
   Dynamic Programing use the "infinite" memory of computer to its advantage, it save the result of some computation in memory to avoid recomputing it in the future.
c) The Greedy Method
   A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining

substructures, in this case not of the problem but of a given solution.

d) The Heuristic Method
I think heuristic algorithm use similar paradigm in what was used in calculus. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Just like calculus.

III. Analysis and Discussion

Like what I have mentioned many times in this paper, we are going to focus our attention to one implementation of algorithm to process data vis, sort algorithm. In particular we are going to discuss one sort algorithm that does not require conformity in the object that it sort vis, topological sort.

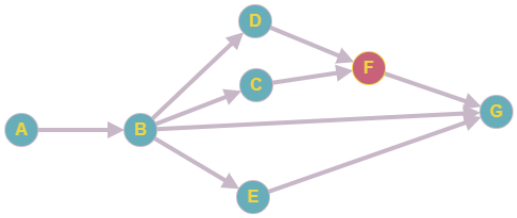
Topological sort can sort a set of object without conformity in them, using some knowledge base which we will refer as partial relation and then transform them to full relation.

For example, topological sort can sort a set of object like {sleep, eat, doing_work} if we give it knowledge base like sleep < eat, sleep < doing_work, and eat < doing_work.

In this paper, we will discuss a particular implementation of topological sort which uses graph theory for its implementation. Not only we will *discus* topological sort, but I have made my own implementation of this algorithm using what I have learn from my course at IF2120 Discrete Math *myself*.

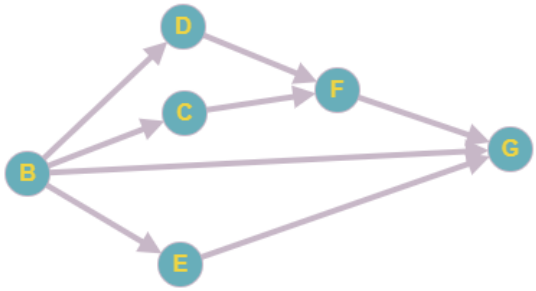
The algorithm that I have used is actually very simple, which I will write bellow,

1. Make a directed graph from the knowledge base that we have. For example if we have element (a < b) in our knowledge base we can make an arc from that with a as our initial vertex, and b as our final vertex.
2. Delete a vertex X that have an incoming degree of 0 (din(q) = 0) then insert that vertex to the list of ordered objects.
3. Delete all arc with X as it incoming degree, then there will be another vertex with incoming degree of 0 because of this step.
4. Repeat step 3 and 4.
   For example if I have a knowledge base of
   a < b, b < c,
   b < d, b < e,
   c < f, d < f,
   f < g, e <g, and
   b < g.
   Then the process of this algorithm is shown as follows,
   Firs, we make a graph out of our knowledge base as follows.

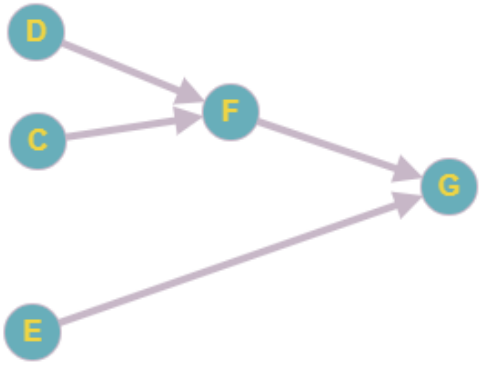


Ordered List = { }.

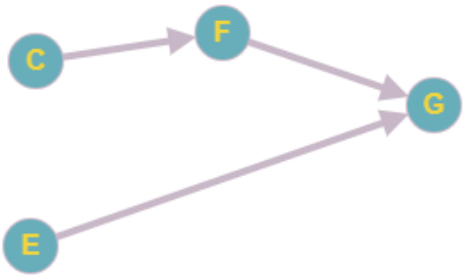
Then we delete vertex A and then insert it into our ordered list,



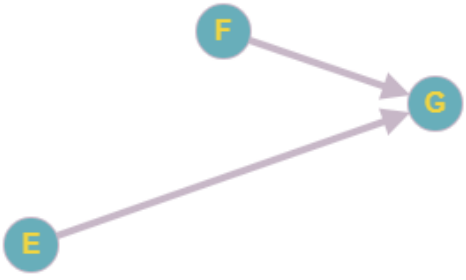
Ordered List = {A}.
Delete vertex B,



Ordered List = { A, B }.
Delete vertex D,



Ordered List = { A, B, D }.
Delete vertex C,



Ordered List = { A, B, D, C }.

Delete vertex E,

Ordered List = { A, B, D, C, E }.
Delete vertex E,

Ordered List = { A, B, D, C, E, F }.
And then we delete vertex G to get an ordered list of { A, B, D, C, E, F, G }.

Source Code

I have made my implementation of topological sort in C language and would like to mention it below,

```
int partialRelation2Graf() {
    /* Local Var */
    address Temp;
    address currAdd;
    address adjacentcyTarget;

    /* Algoritm */
    Temp = Alokasi(Elmt(partialRelation, 1, 1));
    First(InformationArchive) = Temp;
    for(int i = 1; i <= Neff(partialRelation); i++) {
        if(!isThereOnGraf(Elmt(partialRelation, i, 1))) {
            Next(Temp) = Alokasi(Elmt(partialRelation, i, 1));
            Temp = Next(Temp);
        }
        if(!isThereOnGraf(Elmt(partialRelation, i, 2))) {
            Next(Temp) = Alokasi(Elmt(partialRelation, i, 2));
            Temp = Next(Temp);
        }
    }

    for(int i = 1; i <= Neff(partialRelation); i++) {
        currAdd = First(InformationArchive);

        while ((strcmp(Head(currAdd), Elmt(partialRelation, i, 1)) != 0) && (currAdd != NULL)) {
            currAdd = Next(currAdd);
        }
        if (currAdd == NULL) {
            printf("Error at assigning adjacentcy, cannot found target vertex.\n");
            return -1;
        }
        if (adjacentcy(currAdd) == NULL) {
            adjacentcy(currAdd) = Alokasi(Elmt(partialRelation, i, 2));
        } else {
            adjacentcyTarget = adjacentcy(currAdd);
            while(Next(adjacentcyTarget) != NULL) {
                adjacentcyTarget = Next(adjacentcyTarget);
            }
            Next(adjacentcyTarget) = Alokasi(Elmt(partialRelation, i, 2));
        }
    }

    for(int i = 1; i <= Neff(partialRelation); i++) {
        currAdd = First(InformationArchive);

        while ((strcmp(Head(currAdd), Elmt(partialRelation, i, 2)) != 0) && (currAdd != NULL)) {
            currAdd = Next(currAdd);
        }
        if (currAdd == NULL) {
            printf("Error at assigning adjacentcy, cannot found target vertex.\n");
            return -1;
        }
        In(currAdd)++;
    }
    return 0;
}

int sortFunc() {
    address currAdd;
    address adjacentcyTarget;
    address Temp;

    First(SortedList) = NULL;
    while (!isInfoArcEmpty()) {
        currAdd = First(InformationArchive);

        while (In(currAdd) != 0) {
            currAdd = Next(currAdd);
        }
        if (First(SortedList) == NULL) {
            Temp = Alokasi(Head(currAdd));
            First(SortedList) = Temp;
        } else {
            Next(Temp) = Alokasi(Head(currAdd));
            Temp = Next(Temp);
        }
        In(currAdd)--;

        adjacentcyTarget = adjacentcy(currAdd);
        while (adjacentcyTarget != NULL) {
            currAdd = First(InformationArchive);
            while ((strcmp(Head(currAdd), Head(adjacentcyTarget)) != 0) && (currAdd != NULL)) {
                currAdd = Next(currAdd);
```

```
            }
            if (currAdd == NULL) {
                printf("Unexpected NULL at sortFunc\n");
            }
            In(currAdd)--;
            adjacentcyTarget = Next(adjacentcyTarget);
        }
    }
    return 0;
}

int main() {
    printf("Selamat datang di program topological sort!\n");
    printf("Program ini akan merubah keterurutan parsial yang\n");
    printf("kedalam keterurutan lengkap!\n");
    readRelation();
    partialRelation2Graf();
    sortFunc();
    printSolution();
    return 0;
    }
```

I only included the important function in the code snippet and exclude many unimportant parts so not to make the snippet too long.

Test Case

I have also make some test case to test my algorithm, the first example is the same as example I have depicted using graph above,

```
HafidhRendyanto@localhost:~/Documents/Programing/C/TopologicalSort
File  Edit  View  Search  Terminal  Help
[HafidhRendyanto@localhost TopologicalSort]$ ./MAIN
Selamat datang di program topological sort!
Program ini akan merubah keterurutan parsial yang
kedalam keterurutan lengkap!
Tulis informasi parsial relation!
Format penulisan dalam bentuk a b, (yg berarti a < b)
Input 'stop' untuk berhenti.
a b
b c
b d
b e
c f
d f
f g
e g
b g
stop
Pembacaan selesai!
Computing done! Here's your solution :
a < b < c < d < e < f < g
[HafidhRendyanto@localhost TopologicalSort]$
```

The second test case is a recipe to make soup,

```
HafidhRendyanto@localhost:~/Documents/Programing/C/TopologicalSort
File  Edit  View  Search  Terminal  Help
[HafidhRendyanto@localhost TopologicalSort]$ ./MAIN
Selamat datang di program topological sort!
Program ini akan merubah keterurutan parsial yang
kedalam keterurutan lengkap!
Tulis informasi parsial relation!
Format penulisan dalam bentuk a b, (yg berarti a < b)
Input 'stop' untuk berhenti.
potong_dadu_kentang potong_sosis
potong_dadu_kentang iris_bawang_putih
potong_dadu_kentang siapkan_sosis
iris_bawang_putih panaskan_air
panaskan_air sajikan
panaskan_air masukan_bawang
masukan_bawang masukan_kentang
masukan_kentang masukan_wortel
masukan_wortel masukan_sosis
potong_dadu_kentang masukan_kentang
beri_bumbu sajikan
tunggu_mendidih sajikan
aduk_merata sajikan
aduk_merata tunggu_mendidih
beri_bumbu aduk_merata
masukan_kentang aduk_merata
stop
Pembacaan selesai!
Compiting done! Here's your solution :
potong_dadu_kentang < potong_sosis < iris_bawang_putih < siapkan_sosis < panaskan_air < masukan_ba
masukan_kentang < masukan_wortel < masukan_sosis < beri_bumbu < aduk_merata < tunggu_mendidih < sa
[HafidhRendyanto@localhost TopologicalSort]$
```

The last test case, is a procedure to make house,

```
HafidhRendyanto@localhost:~/Documents/Programing/C/TopologicalSort
File  Edit  View  Search  Terminal  Help
Selamat datang di program topological sort!
Program ini akan merubah keterurutan parsial yang
kedalam keterurutan lengkap!
Tulis informasi parsial relation!
Format penulisan dalam bentuk a b, (yg berarti a < b)
Input 'stop' untuk berhenti.
build_wall work_on_your_interior
buy_needed_material lay_foundation
install_electrical_utility find_someone_to_acompany_you
buy_furniture find_someone_to_acompany_you
install_plumbing buy_furniture
work_on_your_interior buy_furniture
install_plumbing work_on_your_interior
build_wall build_roof
build_roof install_electrical_utility
lay_foundation build_roof
lay_foundation build_wall
disign_your_home buy_needed_material
find_location buy_needed_material
find_location disign_your_home
get_the_necesary_permits buy_needed_material
buy_needed_material lay_foundation
disign_your_home get_the_necesary_permits
buy_furniture find_someone_to_acompany_you
work_on_your_interior buy_furniture
find_location install_plumbing
build_wall install_plumbing
build_roof work_on_your_interior
stop
Pembacaan selesai!
Computing done! Here's your solution :
find_location < disign_your_home < get_the_necesary_permits < buy_needed_material <
lay_foundation < build_wall < install_plumbing < build_roof < work_on_your_interior
< install_electrical_utility < buy_furniture < find_someone_to_acompany_you
[HafidhRendyanto@localhost TopologicalSort]$
```

IV. Conclusion

In this paper, we have discussed algorithm in general, discussing it definition, classification all the way to it history, we continue our discussion to more specific type of algorithm that process data, which is sort algorithm, and then we focus once more to a type of sort algorithm that tries to make sense of an object that seemingly cannot be sorted, vis, topological sort.

Topological sort does this, by creating full relation using information derived from knowledge bases. It first make a graph from that knowledge base and then apply the said algorithm to that graph.

V. Acknowledgment

I would like to thank God for giving me power, will, and opportunity to finish this paper in time and I would like to thank my parent for giving me support and the chance to study here in computer science department of STEI.

References

[1] Bell, C. Gordon and Newell, Allen (1971), Computer Structures: Readings and Examples, McGraw–Hill Book Company, New York. ISBN 0-07-004357-4.

[2] Minsky, Marvin (1967). Computation: Finite and Infinite Machines (First ed.). Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-165449-7.

[3] Bolter, David J. (1984). Turing's Man: Western Culture in the Computer Age (1984 ed.). The University of North Carolina Press, Chapel Hill NC. ISBN 0-8078-1564-0., ISBN 0-8078-4108-0pbk.

[4] Burgin, Mark (2004). Super-Recursive Algorithms. Springer. ISBN 978-0-387-95569-8.

[5] Church, Alonzo (1936a). "An Unsolvable Problem of Elementary Number Theory". The American Journal of Mathematics.

[6] Knuth, Donald (1997). Fundamental Algorithms, Third Edition. Reading, Massachusetts: Addison–Wesley. ISBN 0-201-89683-4.

[7] Kosovsky, N. K. Elements of Mathematical Logic and its Application to the theory of Subrecursive Algorithms, LSU Publ., Leningrad, 1981

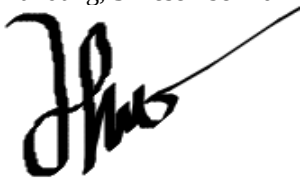
[8] A. A. Markov (1954) Theory of algorithms. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954

[9] https://www.quora.com/What-are-the-uses-of-different-sorting-algorithms-like-bubble-selection-insertion-shell-merge-heap-quick-tree-radix-counting-and-bucket-sort-in-real-life-scenarios, accessed 9 December, 8:13 GMT +7.

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, 3 Desember 2017



Hafidh Rendyanto 13517061