Bidirectional Search Using BFS to Calculate Degrees of Separation

Annisa Ayu Pramesti 13518085
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518085@std.stei.itb.ac.id

***Abstract*—People are all interconnected with each other. Nowadays, social media is very often used and has so many users from various parts of the world. The concept of Six Degrees of Separation stated that for every two people in the world, they are separated by maximum of 5 people. To calculate the degrees of separation, we can find the shortest path possible from two distinct nodes of a graph of world connection with people as the nodes. This study's aim is to calculate the degrees of separation of two Facebook account using a dummy database. A graph will be generated based on the database and then we will perform BDS algorithm with two account as parameters. For every query, there will be calculated the time requited and the average time.**

Keywords—Bidirectional Search, Six Degrees of Separation, shortest path

I. Introduction

Shortest path problem is quite famous problem that mostly be solved using unidirectional search algorithms. The goal is to find the smallest path from a given start state to a goal state. Some famous algorithms are Dijkstra, BFS for undirected and unweighted graph, or Floyd-Warshall algorithm if the goal is to find the shortest path between all pairs of vertexes. Bidirectional search (BDS) is an algorithm to find the path where the search is performed simultaneously from the start and from the goal state until the two search frontiers meet[1]. Bidirectional search replaces single search graph which is likely to grow exponentially with two smaller subgraphs, one starting from initial vertex and other starting from goal vertex. Bidirectional search is complete if BFS is used in both searches and is optimal if the graph has uniform cost.

There is a concept of "Six Degrees of Separation" that any two given entities are no more than six steps away from each other in some sense. Michael Gurevich conducted seminal work in his empirical study of the structure of social networks in his 1961 Massachusetts Institute of Technology PhD dissertation under Ithiel de Sola Pool. Mathematician Manfred Kochen, an Austrian who had been involved in urban design, extrapolated these empirical results in a mathematical manuscript, Contacts and Influences[2], concluding that in a U.S.-sized population without social structure, "it is practically certain that any two individuals can contact one another by means of at most two intermediaries. In a [socially] structured population it is less likely but still seems probable. And perhaps for the whole world's population, probably only one more bridging individual should be needed." There are already so many researches on this topic with different approaches. The concept of six degrees of separation is often represented by a graph of relationships. Real-world applications of the theory include power grid mapping and analysis, disease transmission mapping and analysis, computer circuitry design and search engine ranking. Facebook is a social networking site that connects so many people on the internet. Facebook accounts are interconnected by their friend list and can be represented by undirected graph.

This study's goal is to find the degrees of separation between two Facebook account that is defined by smallest path between them. To find the smallest path from two accounts, we can traverse the list of friends on every account to find the least number of friends it takes to navigate between any accounts. The implementation in this study is limited to dummy databases and can be further developed using actual Facebook databases.

II. Six Degrees of Separation

Six degrees of separation is a well-known idea that any two people on this planet can be connected via an average number of six people[3]. The theory of six degrees of separation states that any two random-selected people on this world can get to know each other by no more than six steps of intermediate friend chains. There are many experiments conducted on social media platform such as Facebook by collecting the profile information of volunteer members provided, they are willing to download and install an application. The result also supported the theory with an average of 5.73 degrees of separation. But the homogeneous characteristics of all participants weaken the result and cannot guarantee it is still true for a diverse range of users with different interests and backgrounds. Furthermore, except the few experiment studies, there has never been a mathematical proof or analysis on the correctness of the six degrees of separation theory in the virtual online society world[3].

III. Blind Search Algorithms, Depth First Search, Breadth First search, and Bidirectional Search

A. Blind Search Algorithms

A blind search (also called an uninformed search) is a search that has no information about its state or search space. The only thing that a blind search can do is distinguish a non-goal state from a goal state.

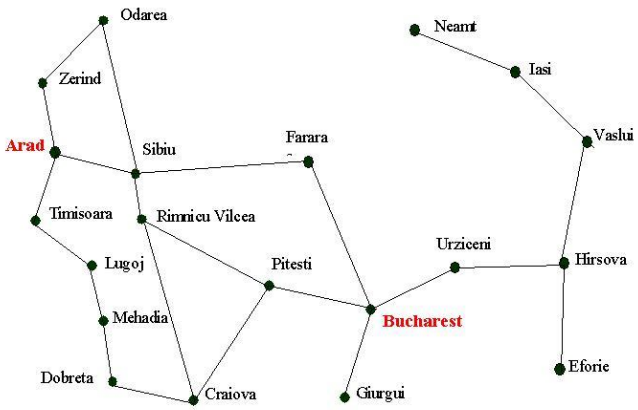

Figure 1. Simplified map of Romania
(source = http://www.cs.nott.ac.uk/~pszgxk/courses/g5aiai/003blindsearches/blind_searches.htm)

Consider Fig 1. Assume that we are currently in Arad and we want to get to Bucharest. If we produce a search tree, level 1 will have three states: Zerind, Sibiu and Timisoara. A blind search will have no preference as to which node it should explore first because it has no idea about the search space of every node.

There are reasons to use a blind search rather than a search with some built in intelligence. One of the reasons is that there may not be any useful information from the graph to be considered through the search. Some problem might just be looking for an answer and won't know where the answer is until it is found.

It is also useful to know the flow and process of uninformed searches as they form the basis for some of the intelligent searches that is going to be useful in solving more complex problems.

There are so many type blind searches. In this paper, we are about to consider blind searches that only differ in the order in which we expand the nodes but, as we shall see, this can have a dramatic effect as to how well the search performs.

B. Depth First Search

Depth First Search (DFS) is an algorithm for searching or traversing tree or graph. This is recursive algorithm involving exhaustive search that uses idea of backtracking. A basic DFS generates and examines every global state that is reachable from a given initial state[4]. The algorithm first check for the validity of the state and its properties using bounded function. If the state is valid, then we generate the connected states. A recursive call to the search procedure is then made for each state that is reachable from this state in one atomic execution step[4]. If the state is not valid then the node is bounded. After that we backtrack to previous node.

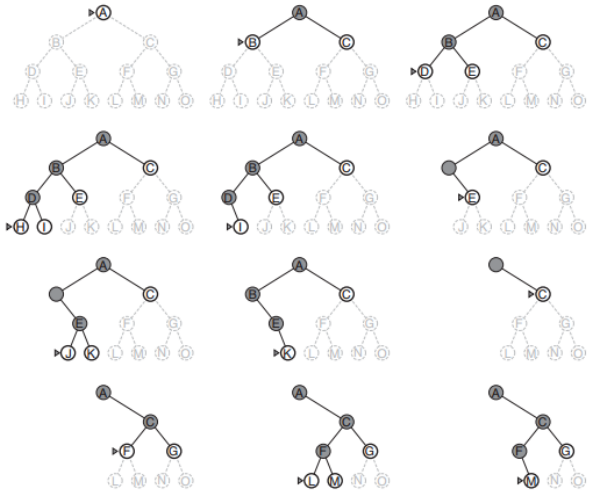

Figure 2. Depth First Search step by step
(source = https://medium.com/kredo-ai-engineering/search-algorithms-part-2-uninformed-search-algorithms-1-be5583a2f1e1)

DFS algorithm is not complete if we do not remember all the nodes present in the path from the root to the current node. If the nodes are stored, it might also save redundant paths. Since DFS idea is to traverse every generated state so the process is unsuitable to perform bidirectional search algorithm.

C. Breadth First Search

Breadth First Search (BFS) is an important building block of many graph algorithms, and commonly used to elaborate with other algorithm to solve many problems. It is usually used to test for connectivity or compute the single source shortest paths of unweighted graphs, directed or undirected[5]. Time complexity of BFS is $O(V+E)$ where V is number of vertices in the graph and E is number of edges in the graph. BFS is complete. This means that for a given graph, BFS will find the solution if it exists.

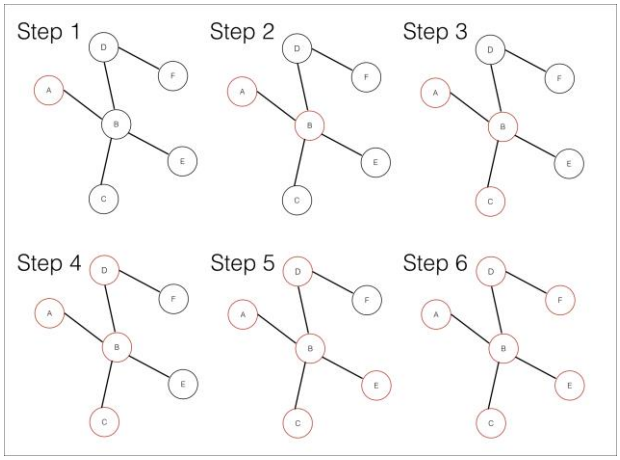

Figure 3. Breadth First Search step by step

(source = https://static.packt-cdn.com/products/9781785884504/graphics/image_07_009.jpg)

Given a distinguished graph *s*, Breadth-First Search (BFS) systematically explores the graph *G* to discover every vertex that is reachable from *s*. Let *V* and *E* refer to the vertex and edge sets of *G*, whose cardinalities are $n = |V|$ and $m = |E|$. We assume that the graph is unweighted; equivalently, each edge $e \in E$ is assigned a weight of unity. A path from vertex *s* to *t* is defined as a sequence of edges $<u_i, u_{i+1}>$ (edge directivity assumed to be $u_i \rightarrow u_{i+1}$ in case of directed graphs), $0 \leq i < l$, where $u_0 = s$ and $u_l = t$. The length of a path is the sum of the weights of edges. We use *d(s, t)* to denote the distance between vertices *s* and *t*, or the length of the shortest path connecting *s* and *t*. BFS implies that all vertices at a distance *k* (or "level" *k*) from vertex s should be first "visited" before vertices at distance *k + 1*. The distance from s to each reachable vertex is typically the final output. In applications based on a breadth-first graph traversal, one might optionally perform auxiliary computations when visiting a vertex for the first time. Additionally, a "breadth-first spanning tree" rooted at *s* containing all the reachable vertices can also be maintained[6].

Algorithm 1 BFS algorithm.

Input: *G(V, E)*, source vertex s.
Output: *d[1..n]*, where *d[v]* gives the length of the shortest path from *s* to $v \in V$.

```
1: for all v ∈ V do
2:    d[v] ← ∞
3: d[s] ← 0, level ← 1, FS ← φ, NS ← φ
4: push s → FS
5: while FS ≠ φ do
6:    for each u in FS do
7:       for each neighbor v of u do
8:          if d[v] = ∞ then
9:             push v → NS
10:            d[v] ← level
11:   FS ← NS, NS ← φ, level ← level + 1
```

There are so many real-life problems that can be solved using BFS. One of the examples is graph based fingerprint and matching algorithm using coupled BFS[7].

D. Bidirectional Search

Bidirectional search (BDS) is a graph search algorithm to find smallest path from source from goal node. The name represents the algorithm very well because there are two simultaneous searches[8]. The first search is forward search from source/initial node and the second is backward search from goal/target node. Bidirectional search can be run and guided by heuristic estimate of remaining distance from source to goal and vice versa. This algorithm is just modification of many search algorithm but performed twice from initial and target, so they are expected to meet in the middle.

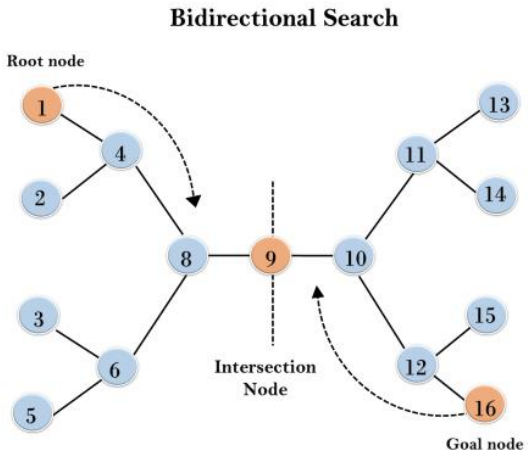

Figure 4. Bidirectional search illustration
(source = https://www.javatpoint.com/ai-uninformed-search-algorithms)

To get the optimal result of BDS, the path generated by the graph should all have the same cost. The algorithm can be very effective approach when the branching factor is the same in both directions. Bidirectional search is complete if for both searches, BFS is used.

So many cases are found to be solved faster using BDS than regular BFS because it can reduce the amount of required exploration/expansion of nodes. Suppose if branching factor of a graph is *b* and the distance of goal node from initial node is *d*, then by using BFS/DFS the searching complexity would be $O(b^d)$. If we divide this search by two, we get searching complexity $O(b^{d/2})$ for each divided search then the total complexity would be $O(b^{d/2}+b^{d/2})$ whish is far less than the BFS/DFS search complexity. This also means that if the branching factor is not the same for every node, then there is no guarantee that BDS will be better than BFS/DFS.

We consider a generic bidirectional search in Algorithm 2. First, it maintains open and closed lists for each direction. The directions are *f* (forward) or *b* (backward). States originating from the start are in the forward direction, while those originating from the goal are in the backwards direction. Second, it must check for solutions by finding the same state on the opposite open list, instead of performing a goal test against a goal state. Third, it does not always terminate immediately upon finding a solution. The search must continue until the best solution found thus far has been proven to be optimal[9].

Algorithm 2 Bidirectional search algorithm

```
1: procedure bidirectional_search(start, goal)
2:    push(start, OPEN_f )
3:    push(goal, OPEN_b)
4:    while OPEN_f and OPEN_b not empty do
5:       remove best state s from OPEN_f / OPEN_b
6:       if can terminate search then return success
7:       end if
8:       if s was on OPEN_f then
9:          move s to CLOSED_f
10:         for each successor s_i of s do
```

```
11:          if s_i on OPEN_f then
12:             update cost of s_i on OPEN_f if shorter
13:          else if s_i not on CLOSED_f then
14:             add s_i to OPEN_f
15:          end if
16:          if s_i on OPEN_b then // ISD
17:             update best solution
18:          end if
19:       end for
20:    else
21:       // Analogous code in backwards direction
22:    end if
23: end while
24: return failure
25: end procedure
```

A real life example of bidirectional search algorithm is in keyword search on graph databases[10]. Relational, XML and HTML data can be represented as graphs with entities as nodes and relationships as edges. A central problem in this scenario is to efficiently extract from the data graph a small number of the "best" answer trees. A Backward Expanding search, starting at nodes matching keywords and working up toward confluent roots, is commonly used for predominantly text-driven queries. But it can perform poorly if some keywords match many nodes, or some node has very large degree. Bidirectional Search improves on Backward Expanding search by allowing forward search from potential roots towards leaves. To exploit this flexibility, a novel search frontier prioritization technique is devised based on spreading activation. In this case, Bidirectional Search significantly outperforms Backward Expanding search. In addition to edge weights, the ranking of an answer may also depend on a notion of node prestige. As with Pagerank with decay, not all nodes are equal in status[11]. A method of computing node prestige based on indegree is defined in [12], while [13] defines global and per-keyword node prestige scores for each node. Node prestige scores can be assumed to be precomputed for our purpose, although they could potentially be computed on-the-fly.

There is also BDS that is guaranteed to meet in the middle as explained from [14]. They introduce a framework that divides the state-space into disjoint regions and allows a careful analysis of the behavior of the different algorithms in each of the regions. The algorithm that they are using as search algorithm for every starting node is A*[15].

IV. Data Structure, Query, and Response Model

A. Data Structure

Facebook users are connected via their friend list. To be able to connect, Facebook users must add friend to another user and then wait for the request to be accepted. This implies once the request is accepted, the other user do not have to request back to get connected. This relationship can be represented to undirected graph because for all user, if one user is connected to another user, the opposite applies.

We model the connections as an undirected graph in which nodes are users and edges are relationships. For each user p in the database that we need to represent, the data graph has a corresponding node u_p. For each pair of users p_1 and p_2 such that there is a connection from p_1 to p_2, the graph contains an edge from u_{p1} to u_{p2}[10].

B. Query

A query is a set of keywords consists of two terms t_1 and t_2. t_1 is Facebook id of the first account and t_2 is the Facebook is of the second account. We are going to find the degrees of separation between t_1 and t_2. Let u_1 be the node matches with t_1 and u_2 be the node matches with t_2. The query is assumed to be in the database graph. The next step is to apply the BDS algorithm to the graph.

C. Response Model

A response or answer to a keyword query is one or more shortest path between the correspondent nodes of the query. If there are more than one path, then all the answer should be in the same degrees. The answer will be represented in an undirected graph that connects the two nodes from the query. The answer also includes how many degrees/how long the path calculated from the initial to target node.

V. Facebook Degrees of Separation

A. Building Facebook Connection Graph

Using the data structure that we have been mentioned earlier, we generate the graph with Facebook connections as edges and account ids as nodes. Here we are using dummy database that consists of 100 fake accounts that are interconnected. These accounts might be not representing the actual data since Facebook has huge databases but the more real accounts/nodes the better because we then can prove the concept of "Six Degrees of Separation".

B. Calculate the Degrees of Separation

Before we compute the degrees of separation, we first determine the query. It is assumed that the query should be contained in the graph. Below is the implementation of bidirectional search between 2 ids from given query. After we find the shortest path possible, we can calculate the degrees of separation. For all result obtained from this procedure, we show them if the results are in the same degrees.

```
int biDirSearch(int s, int t)
{
    bool s_visited[V], t_visited[V];
    int s_parent[V], t_parent[V];
    list<int> s_queue, t_queue;
    int intersectNode = -1;

    for(int i=0; i<V; i++)
    {
        s_visited[i] = false;
        t_visited[i] = false;
    }

    s_queue.push_back(s);
    s_visited[s] = true;
    s_parent[s]=-1;
    t_queue.push_back(t);
    t_visited[t] = true;
    t_parent[t] = -1;

    while (!s_queue.empty() && !t_queue.empty())
    {
        BFS(&s_queue, s_visited, s_parent);
        BFS(&t_queue, t_visited, t_parent);

        intersectNode = isIntersecting(s_visited, t_visited);

        if(intersectNode != -1)
        {
            cout << "Path exist between " << s << " and "
                 << t << "\n";
            cout << "Intersection at: " << intersectNode << "\n";

            printPath(s_parent, t_parent, s, t, intersectNode);
            exit(0);
        }
    }
    return -1;
}
```

Figure 5. Implementation of bidirectional search between 2 ids *(source = https://www.geeksforgeeks.org/bidirectional-search/)*

C. *Testing*

To test the degrees of separation, the queries are obtained from choosing random account ids from the database We choose 20 different queries and for every query, we perform bidirectional search from point B and basic BFS in comparison to get the degrees of separation. From the test conducted, 100% results from bidirectional search are the same with results from BFS. The difference between the two algorithms are the time complexity as we have discussed. The bidirectional search is faster than BFS by 1.32 average ratio. This time complexity indicate that the database's branching factor is quite the same for every node to be better than BFS but we did not calculate the prediction of time required based on the comparison of time complexity between BDS and BFS.

VI. Conclusion

A program has been created using Bidirectional Search using BFS as search algorithm for each starting node. This program is to calculate degrees of separation between two different nodes from a given graph. The accuracy of this program is 100% for a given database and the overall time complexity is better than unidirectional search BFS by 1.32 average ratio.

VII. Future Research

This research can be done better with bigger database or real-life database. There are many researches that can be developed based on this idea proposition such as developing the search algorithm so it can be much faster and efficient or implementing this algorithm to another interesting problems. This algorithm can also be used in various researches in many fields such as medical field, natural language processing, etc., if the data can be represented by undirected and unweighted graph.

Video Link at Youtube

The video of this paper can be accessed in the following link:https://www.youtube.com/watch?v=sKWmOgJwzgk&t=48s

Acknowledgment

The author would like to express her gratitude to God Almighty for his guidance, to her parents for their eternal support, love and education, and to the Mrs. Masayu Leyla Khodra for her teachings of Algorithm Strategies in Class K1 to complete this paper.

References

[1] C. Moldenhauer, A. Felner, N. Sturtevant, and J. Schaeffer, "Single-frontier bidirectional search," *Proc. 3rd Annu. Symp. Comb. Search, SoCS 2010*, pp. 151–152, 2010.

[2] I. De Sola Pool and M. Kochen, "Contacts and influence," in *The Structure and Dynamics of Networks*, 2011.

[3] L. Zhang and W. Tu, "Six Degrees of Separation in Online Society," *Journal.Webscience.Org*, pp. 1–5, 2009.

[4] G. Holzmann, D. Peled, and M. Yannakakis, "On nested depth first search," vol. 32, pp. 23–31, 1997.

[5] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," *Sci. Program.*, vol. 21, no. 3–4, pp. 137–148, 2013.

[6] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," *Proc. 2011 SC - Int. Conf. High Perform. Comput. Networking, Storage Anal.*, 2011.

[7] S. Chikkerur, A. N. Cartwright, and V. Govindaraju, "K-plet and coupled BFS: A graph based fingerprint representation and matching algorithm," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3832 LNCS, pp. 309–315, 2006.

[8] A. Bundy and L. Wallen, "Bidirectional Search," in *Catalogue of Artificial Intelligence Tools*, 1984.

[9] N. R. Sturtevant and J. Chen, "External memory bidirectional search," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2016-Janua, pp. 676–682, 2016.

[10] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," *VLDB 2005 - Proc. 31st Int. Conf. Very Large Data Bases*, vol. 2, pp. 505–516, 2005.

[11] P. Lofgren, S. Banerjee, and A. Goel, “Personalized PageRank estimation and search: A bidirectional approach,” in *WSDM 2016 - Proceedings of the 9th ACM International Conference on Web Search and Data Mining*, 2016.

[12] A. Balmin, S. Diego, and L. Jolla, “ObjectRank : Authority-Based Keyword Search in Databases ∗ Computer Science,” *Science (80-.).*, pp. 564–575, 2004.

[13] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, “Keyword searching and browsing in databases using BANKS,” *Proc. - Int. Conf. Data Eng.*, 2002.

[14] R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant, and J. Chen, “MM: A bidirectional search algorithm that is guaranteed to meet in the middle,” *Artif. Intell.*, vol. 252, pp. 232–266, 2017.

[15] Z. Zhang, N. R. Sturtevant, R. Holte, J. Schaeffer, and A. Felner, “A* search with inconsistent heuristics,” in *IJCAI International Joint Conference on Artificial Intelligence*, 2009.

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, 1 Mei 2020

Annisa Ayu Pramesti
13518085