Determination of Shortest path in a graph(BFS) in a graph and use in Genome assembly

Natte Sai Bharath
Department of Computer Science
and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, Kerala, India
amenu4aie20052@am.students.amrita.edu

Abstract—Determining the shortest route is one of the most discussed issues using the other Algorithm such as Djikstra, Floyd Warshall and this research is the Breadth First Search algorithm used, the Breadth First Search algorithms in this study are used to determine the shortest route and accuracy from Cartesian field, advanced and accurate cartesian route testing sites that use the Breadth First Search algorithm can do very well and find a specific route to choose from the best to the longest route.And then we will discuss the BFS use in Genome assembly

1.INTRODUCTION

The development of computer-based technology is currently undergoing rapid development, and the development has also integrated various aspects of the health sector with the aim of completing toll-intensive work, the most sophisticated platform design today. Yan ZHOU [1] conducted a very short study using the Breadth First Search algorithm by approaching the search from the start to the destination by checking the layer by layer. against all possibilities can be transferred and remove all the impossible routes to pass. Like Yan ZHOU this experiment is trying to find a shorter route that can pass through the Cartesian field. This matter can be explained as follows; it is determined that the object will move from the center (0,0) to the point A (m, n). The object should only bend at the grid points and always align with the x axis or the y axis. The object must not exceed the point of transmission and must not exceed the point of crossing. Thereafter a set of provisions that limit the movement of an object is provided. The question is what clues can I pass using the words described above.

1.1 BFS Theory

The Breadth First Search (BFS) is another fundamental search algorithm used to explore nodes and edges of a graph. It runs with a time complexity of and is often used as a building block in other algorithms.The BFS algorithm is particularly useful for one thing: finding the shortest path on unweighted graphs.A BFS starts at some arbitrary node of a graph and explores the neighbor nodes first, before moving to the next level neighbours.The advantage of using this algorithm is that it will not get caught, and if there is more than one.The solution is then a comprehensive search will find all the solutions, and if there is more than one solution then a small solution will be found.

2. LITERATURE REVIEW

Making computer routes in densely populated areas with multiple nodes and millions of traffic where some vertices may be similar or very close still consumes a lot of what is expected of the calculation and makes the calculation very difficult. The desire to produce complete algorithms, that is, algorithms that find a way when one is present - causes mathematical instability when applied to problems of high magnitude despite their low (polynomial) complexity. It is important to address the practical problems of organizing movement in complex industrial areas as discussed which has led to the development of possible search algorithms and Monte Carlo, commonly known as the random road mapping method [2]. Although algorithms are likely to successfully trade perfection so that they can accurately calculate, they have difficulty finding the right paths in small tunnels in a relaxed environment, where the chances of sample preparation are low.

2.1 Review on BFS

The first deep algorithm that can find the final locations with connected transverse nodes from the first position to the end.The node is a Breadth-first Algorithm. This benefit of this algorithm is very accurate because all nodes are present. The disadvantages of this algorithm are sometimes it is possible to find the end of a node longer than other algorithms for all nodes have been passed.

It starts from the beginning and marks all nodes such as ‘not visited’. After that, the first node is over connected to the queue. Algorithm registers distances by transmitting connected nodes. Then, it repeats the same process in other nodes.

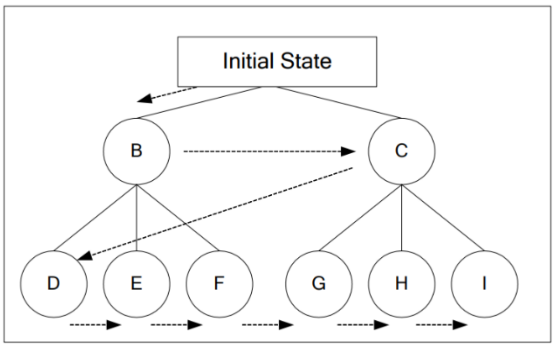
3.METHODS AND DATASETS

3.1 Breadth-first Search(BFS) Methodology

Advanced search first uses the line as a data structure to determine which vertex to visit next. Every time a new vertex is visited, all of its neighbors are added at the end of the line. Determining which vertex should you visit next, vertex from the front of the line is removed and the process is repeated. Therefore, the vertices are visited in the first direction (FIFO). The result is that all tweets away from the first vertex are visited before tweets a +1 distance from the first vertex.

Breadth-first search provides two values per vertex v:

The distance, which gives a small number of edges on any path from the vertex of the source to the vertex v.Previous vertex of v on a short path from the vertex of the source. Source vertex predecessor is a special, null-like value, indicating that it has no predecessor.If there is no path from source vertex to vertex v, the v range is infinite and the preceding one has the same special value as the previous source.



```
# Global/class scope variables
n = number of nodes in the graph
g = adjacency list representing
unweighted graph
# s = start node, e end node, and 0 ≤
e, s < n
function bfs (s, e):
# Do a BFS starting at node s
prev = solve(s)
# Return reconstructed path from s →
e
return reconstructPath(s, e, prev)

function solve(s):      #function for
BFS
q = queue data structure with enqueue
and dequeue
q.enqueue(s)
visited = [false, ..., false] # size n
visited[s] = true
prev = [null, ..., null] # size n
while !q.isEmpty():
    node = q.dequeue()
```

```
     neighbours = g.get(node)
     for(next : neighbours):
         if !visited [next]:
             q.enqueue(next)
             visited [next] = true
      prev [next] = node
      return prev
 End BFS
```

```
function reconstructPath , prev  :
Reconstruct path going backwards
from e
path = []
for (at = e; at != null; at = prev [at]):
path.add(at)
path. reverse()
If  and e are connected return the path
if path [0] = s:
return path
return []
```

3.2 Genome Assembly

Genome integration refers to the process of sequencing nucleotide sequences. It is necessary to combine, because the length of the sequential reading - at least for now - is much shorter than most genes or many genes. Genome integration was facilitated by the availability of a public website, which is free of charge on the National Center for Biotechnology Information (http://www.ncbi.nlm.nih.gov). Just as it is very easy to put together a picture puzzle if you know what a picture looks like, it is very easy to combine genes with genomes if you have a good idea of sequencing. In the human genome, genes occur in the same space on the chromosome, but there may be different numbers of copies and dynamic numbers of repetitive sequences that make it difficult to combine. Although the genome of the bacteria is very small, the genes do not actually exist in the same area and many copies of the same gene may appear in different parts of the genome. So even with the advent of commercial software and a ever-growing database, the process of genome integration can take much longer than the time to find the exact sequence.

4. RESULTS AND DISCUSSION

4.1 Performance

Let g define the maximum number of node nodes in each node in our graph. Additionally, it should be the shortest route between startNode and stopNode. Then this algorithm has a complex time O (g^d).
Why is that? BFS searches the graph for so-called levels. Every node level has the same distance to the starting point. It takes O (g) steps to reach level 1, O (g^2) steps to get to level 2, and so on. Therefore, it takes O (g^d) steps to reach level d. Using the n and e variables again, the operating time is still O (n + e). However, O (g^d) is a more accurate statement if you are looking for a much shorter route.
In some graphs, the line may contain all of its nodes. Therefore, it is also the space complex of O (n).
Minor comment: The actual operating time for the above usage is worse than O (n + e). The reason is that the same JavaScript members are used as a queue. The switch function takes the time of O, when s is the line size. However, it is possible to use a line in JavaScript that allows tasks to follow and stop at O (1)

We can say that the BFS has the following as advantagesThe solution will definitely be found by BFS If there is a solution.BFS will never be trapped in a blind spot, which means that the nodes are not needed. If there is more

than one solution then we will find a solution with small steps.
But regarding the disadvantages, Memory Issues As it keeps all nodes from the current level to the next level.And alsoIf the solution is far away it means it is time consuming.
BFS usage:
1. Finding the Shortcut.
2. A brief graph analysis.
3. Copying Cheney's algorithm.

5. CONCLUSION

BFS starts by visiting all nodes at level 1 from the first location. Then all the nodes in grade 2. Then all the nodes in grade 3, etc.
Horizontal graphing is a unique process that requires an algorithm to visit, scan, and / or update an un visited site in a tree-like structure. The BFS algorithm works on the same principle.The algorithm is useful for analyzing nodes on a graph and to construct a very short cross-section of these.
The algorithm breaks the graph with a small number of repetitions and a very short time.
BFS selects one node (first point or source) on the graph and visits all nodes near the selected location. BFS accesses these nodes one by one. Visited and marked data is stored by BFS online. The line is working for the first time. Thus, the element set on the graph first is first removed and printed as a result.The BFS algorithm will never be caught in an endless loop.Due to its high accuracy and robust implementation, BFS is used in many real-life solutions such as P2P networks, Web Crawlers, and Network Broadcast.

6.REFERENCES

[1] Baum, Johannes. “5 Ways to Find the Shortest Path in a Graph | by Johannes Baum | Better Programming.” *Medium*, Better Programming, 7 Feb. 2020, https://betterprogramming.pub/5-ways-to-find-the-shortest-path-in-a-graph-88cfefd0030f.

[2] Rahim1, Robbi. “Breadth First Search Approach for Shortest Path Solution in Cartesian Area - IOPscience.” *ShieldSquare Captcha*, https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012036. Accessed 18 Jan. 2022.

[3] “ROSALIND | Glossary | Algo: Breadth-First Search.” *ROSALIND | Problems | Locations*, https://rosalind.info/glossary/algo-breadth-first-search/. Accessed 18 Jan. 2022.

[4] “ShieldSquare Captcha.” *ShieldSquare Captcha*, https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012036/pdf. Accessed 18 Jan. 2022.

[5] Unadkat, Vatsal. “Advantages and Disadvantages of Search Algorithms | by Vatsal Unadkat | Medium.” *Medium*, Medium, 21 Nov. 2019, https://medium.com/@vatsalunadkat/advantages-and-disadvantages-of-ai-algorithms-d8fb137f4df2.

[6] WilliamFiset. *Breadth First Search Algorithm | Shortest Path | Graph Theory*. YouTube, 2 Apr. 2018, https://www.youtube.com/watch?v=oDqjPvD54Ss.

[7] “Breadth First Search (BFS) Algorithm with EXAMPLE.” *Guru99*, https://www.facebook.com/guru99com/,https://www.guru99.com/breadth-first-search-bfs-graph-example.html. Accessed 18 Jan. 2022.

7.Appendix

Code in python

```python
data = "/content/drive/MyDrive/rosalind_bfs.txt"
f = open(data, "r")
vertice, edge = map(int, f.readline().strip().split(" "))
graph = {i+1:[] for i in range(vertice)}
for line in f:
    l = list(map(int, line.strip().split(" ")))
    graph[l[0]].append(l[1])
f.close()

def BFS(start_vertice, vertice, graph):
    quene, order = [], [] # quene stores the data that needs to be
traversed, and order stores the traversed path
    distance = {i+1:0 for i in range(vertice)} # Initialize shortest
path
    quene.append(start_vertice)
    order.append(start_vertice)
    # Do a breadth-first traversal
    while quene:
        v = quene.pop(0)
        for n in graph[v]:
            if n not in order:
                distance[n] = distance[v] + 1
                order.append(n)
                quene.append(n)
    # 1 Unreachable point, set the distance to -1
    for k in distance.keys():
        if k not in order:
            distance[k] = -1
    # return order, and distance
    return order, distance

start_vertice = 1
order, distance = BFS(start_vertice, vertice, graph)
for i in range(int(vertice)):
    print(distance[i+1], end = " ")
```