COMP20230: Data Structures & Algorithms **2018-19 Semester 2**

Lecture 17: Graphs (2)

Lecturer: Dr. Andrew Hines *Scribes: Zhaolun Hui, Xun liu*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

17.1 Outline

This session introduces Graphs. It contains Breadth-First Search (BFS), Depth-First Search (DFS) and Dijkstra's algorithm (SPF). There are explanations and complexities for each algorithm. Also some examples are included.

17.2 Breadth-First Search (BFS)

17.2.1 Explanation

For a given graph G = (V, E), the breadth-first search starts from a source vertex, and by traversing (searching) its adjacency list, all vertices which are adjacent to the source vertex will be found. According to analogy, it is continuously expanded outward to find vertices adjacent to the adjacent points of the source vertex.

17.2.2 Complexity

BFS is a process of borrowing queues for storage, it searches hierarchically, and prioritize the points which are close to the starting point. Whether it is stored in the adjacency list or the adjacency matrix, an auxiliary queue is needed, and there are v vertices need to be enqueued. In the worst case, the space complexity is O(v).

In the adjacency matrix, each vertex needs to be searched at one time, and the time complexity T1=O(v). When the search starts, the nodes that have not been visited will be accessed. In the worst case, each vertex is accessed at least once, and each side is accessed at least once. This is because during the search process, if the child nodes have been visited, the node will be rolled back, so the time complexity is O(E), and the total time of the algorithm is O(—V—+—E—).

In the adjacency matrix, the time required to find the neighboring point of each vertex is O(V), which is the row of the row where the node is located. There are n vertices, so the total time complexity is O(—V—2).

17.3 Depth-First Search (DFS)

17.3.1 Explanation

When the algorithm is used, it is to find a head node, and then continue to find along the head node until it reaches the last sub-node that satisfies the condition, and then find another path, when it is not satisfied along a road. Will automatically jump to the upper node to judge

DFS implementation is much the same as BFS, just replace the queue with a stack, and the stack has a LIFO (Last Input First Output) feature.

17.3.2 Explanation

When the algorithm is used, it is to find a head node, and then continue to find along the head node until it reaches the last sub-node that satisfies the condition, and then find another path, when it is not satisfied along a road. Will automatically jump to the upper node to judge.

DFS implementation is much the same as BFS, just replace the queue with a stack, and the stack has a LIFO (Last Input First Output) feature.

17.3.3 Complexity

For a given graph G = (V, E), the DFS algorithm is a recursive algorithm that requires a recursive work stack, so its space complexity is O(V).

The process of traversing a graph is a process of finding its neighbors for each vertex, and the time it takes depends on the structure employed.

In the adjacency list, the time required to find the adjacent points of all vertices is O(E), and the time cost of accessing the adjacent points of the vertices is O(V). So, the total time complexity is O(V+E).

In the adjacency matrix, the time required to find the adjacent points of each vertex is O(V). To find the entire matrix, the total time is $O(V^2)$.

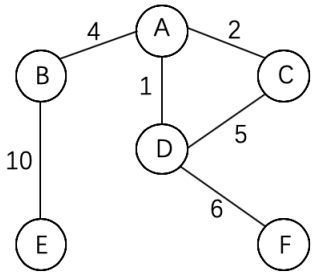
17.4 Weighted graphs

17.4.1 Explanation

Previous lecture introduced unweighted graphs in which the edges connecting nodes carry same weight. It means that if we traverse through a graph, it makes no differences which path is taken. A weighted graph is a graph where the edges have corresponding values. These are also referred to as weights, and they have an effect on how this graph is traversed.

17.4.2 Example

We designed and draw a simple weighted graph and the corresponding Adjacency List in figure 1.



Node	Adjacent Nodes
A	B(4), C(2), D(1)
B	A(4), E(10)
C	A(2), D(5)
D	A(1), C(5), F(6)
E	B(10)
F	D(6)

Figure 1 Graph and Adjacency List

The corresponding Adjacency Matrix is presented as figure 2.

	A	B	C	D	E	F
A		4	2	1		
B	4				10	
C	3			5		
D	1		5			6
E		10				
F				6		

17.5 Dijkstra's algorithm (SPF)

17.5.1 Explanation

Dijkstra's algorithm is an algorithm to find the shortest paths from one node to other nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger. Dijkstra in 1956 and published three years later. Its a greedy algorithm with n-1 times searching.

17.5.2 Steps

i. Set the distance to start vertex to 0 and other vertices to infinity.

ii. Set the current vertex to the source

iii. Flag the current vertex as visited.

iv. For all vertices adjacent to the current vertex, set the distance from the source to the adjacent vertex equal to the minimum of its present distance and the sum of the distance from the current vertex to the adjacent vertex and the distance from the source to the current vertex.

v. From the set of unvisited vertices, set one as the new current vertex, provided that there exists an edge to it such that it is the minimum of all edges from a vertex in the set of visited vertices to a vertex in the set of unvisited vertices.

vi. Repeat steps iii-v until all vertices are visited.

17.5.3 Algorithm

```
def Dijkstra(G, s):
        for x in V:
                d[x] = ∞
                p[x] = None
        d[s] = 0
        Q = V

        while Q not empty:
                Pick x in Q to minimize d[x]
                Q -= {x}

                for (x, y) in E:
                        if d[x] + w(x, y) < d[y]:
                                d[y] = d[x] + w(x, y)
                                p[y] = *x

        return d, p
```

17.5.4 Complexity

Basic Dijkstra is O(—V—^{2})$Canbeoptimised to O(|E| + |V|log|V|)$

17.6 References

1.Ita$_c$, *estenger*(2018).*Dijkstrasshortestpathalgorithm|GreedyAlgo−7−GeeksforGeeks*.[*online*]*Availableat* : *https* : *//www.geeksforgeeks.org/dijkstras−shortest−path−algorithm−greedy−algo−7/*[*Accessed*18*Apr*.2019].

2.Max (Ang) Li (2015). Dijkstra's Algorithm in Brief [online] Available at:

https://www.hackerearth.com/zh/practice/notes/dijkstras-algorithm/[Accessed 18 Apr. 2019].