Introduction to Data Structure and Algorithms

Hairong Wang

School of Computer Science & Applied Mathematics
University of the Witwatersrand, Johannesburg

2019-8-27

Contents

WITS UNIVERSITY

Outline

1. Breadth first search and depth first search
   - Basics of graphs
   - Breadth first search
   - Depth first search

WITS UNIVERSITY

Outline

1. Breadth first search and depth first search
   - Basics of graphs
   - Breadth first search
   - Depth first search

A few definitions on graphs

Some definitions:

- A graph $G = (V, E)$ consists of a set of vertices, V, and a set of edges, E.
- Each edge is a pair (v, w), where $v, w \in V$.
- If the pair is ordered, then the graph is directed.
- Vertex w is adjacent to v if $(v, w) \in E$.
- A path in a graph is a sequence of vertices $w_1, w_2, \ldots, w_N$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq i \leq N - 1$.
- A cycle in a directed graph is a path of length at least 1 such that $w_1 = w_N$.
- A cycle in a undirected graph has distinct edges.
- A directed graph is acyclic if it has no cycles.
- An undirected graph is connected if there is a path from every vertex to every other vertex.

Representation of graphs

- Adjacency matrix
- Adjacency list

Grids

- Many problems can be represented using a grid, for example, solving a maze.
- Grids are a form of (implicit) graph.
- We can represent a grid using adjacency list or adjacency matrix.

Grids cont.

- Problem: solve a maze on a grid where our task is to find a path from a source cell to a goal cell on a grid using BFS and DFS, respectively.
- For this problem, we represent a grid simply using 2D array, since for each cell, its neighbours are easily found by its spacial location, i.e., its array index in the case of using array data structure.

Outline

WITS UNIVERSITY

Breadth first search

- Breadth first search (BFS) is one of the simple search algorithms used to explore nodes and edges in a graph.
- It is particularly useful to find the shortest path in a undirected graph.
- BFS starts at a source node of a graph and explores all the neighbours of the current node before moving on to the next level neighbours.
- BFS algorithm uses a queue data structure to track which node to visit next. Upon reaching a new node, the algorithm adds it to the queue to visit it later

BFS algorithm

Algorithm 1: Grid search using BFS

```
q ← new Queue
Initialise a Distance array of NumRows by NumCols to MAX_INT
Initialise a Parent array of NumRows by NumCols to (−1, −1)
Distance[startR][startC] ← 0
Paren[startR][startC] ← (−2, −2)
Enqueue(startR, startC)
while q isn't empty and the goal is not found do
    curr ← Dequeue(q)
    if curr != goal then
        for For each valid neighbour tmp of curr do
            if tmp is open and unvisited then
                Distance(tmp) ← Distance(curr) +1
                Parent(tmp) ← curr /* visited(tmp) ← True */
                Enqueue(tmp)
            end if
        end for
    else
        set goal is found
    end if
end while
if the queue is empty and the goal is not found then
    return No path to goal
else
    curr ← parent(goal)
    while curr hasn't reached the start do
        data[currRow][currCol] =' *'
        curr ← Parent(curr)
    end while
end if
```

Time complexity of BFS

- Each cell (or a vertex in a graph) is enqueued at most once, and hence dequeued at most once – the total time is $O(|V|)$.
- When a cell is dequeued, all of its neighbours are examined – the total time is $O(|E|)$.
- The total time complexity is $O(|V| + |E|)$.

Note that, in a grid of M by N, where M is the number of rows and N is the number of columns, we can view it as a graph with $M \times N$ vertices (or nodes). As we consider 4 neighbours for each cell, we can view it as each node has at most 4 edges connected to its four neighbours.

Outline

1. Breadth first search and depth first search
   - Basics of graphs
   - Breadth first search
   - Depth first search

Depth first search

- Starting at an arbitrary node, DFS searches ‘deeper’ in a graph whenever possible, until it cannot go any further, at which point it backtracks and continues.

Grid search using DFS

Algorithm 2: Recursive depth first search

```
function RecursiveDFS(curr)
    if curr == goal then
        found ← TRUE
        return curr
    else
        down ← {curr.row + 1, curr.col}
        if down is valid and unvisited then
            parent[down] ← curr
            RecursiveDFS(down)
        end if
        left ← {curr.row, curr.col − 1}
        if left is valid and unvisited then
            parent[left] ← curr
            RecursiveDFS(left)
        end if
        up ← {curr.row − 1, curr.col}
        if up is valid and unvisited then
            parent[up] ← curr
            RecursiveDFS(up)
        end if
        right ← {curr.row, curr.col + 1}
        if right is valid and unvisited then
            parent[right] ← curr
            RecursiveDFS(right)
        end if
    end if
end function
```

Grid search using DFS cont.

Algorithm 3: Iterative depth first search

```
function IterativeDFS(curr)
    s ← a new stack
    parent(start) ← {−2, −2}
    s.push(start)
    while s is not empty and goal is unvisited do
        curr ← s.peek()
        if curr == goal then
            found ← TRUE
            return curr
        else
            down ← {curr.row + 1, curr.col}
            left ← {curr.row, curr.col − 1}
            up ← {curr.row − 1, curr.col}
            right ← {curr.row, curr.col + 1}
            if down is valid and unvisited then
                parent[down] ← curr
                s.push(down)
            else if left is valid and unvisited then
                parent[left] ← curr
                s.push(left)
            else if up is valid and unvisited then
                parent[up] ← curr
                s.push(up)
            else if right is valid and unvisited then
                parent[right] ← curr
                s.push(right)
            else
                s.pop()
            end if
        end if
    end while
end function
```

WITS UNIVERSITY

Time complexity of DFS

- For each grid cell (or a vertex in a graph), `RecursiveDFS` is called at most once – the time is $O(|V|)$.
- Within `RecursiveDFS`, for each cell, at most four edges are considered – the time is $O(|E|)$
- The running time of DFS is $O(|V| + |E|)$.

WITS UNIVERSITY