NANYANG TECHNOLOGICAL UNIVERSITY

CE2001/ CZ2001: Algorithms

Graphs

Ke Yiping, Kelly

NANYANG TECHNOLOGICAL UNIVERSITY

Traversal of Graphs

NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

At the end of this lecture, students should be able to:

- Solve the problem related to systematic graph traversals using Breadth First Search (BFS) and Depth First Search (DFS) algorithms
- Analyse the time complexities of BFS and DFS
- Use backtracking to solve artificial intelligence problems:
  - Eight-queens problem
  - Navigating through a maze

NANYANG TECHNOLOGICAL UNIVERSITY

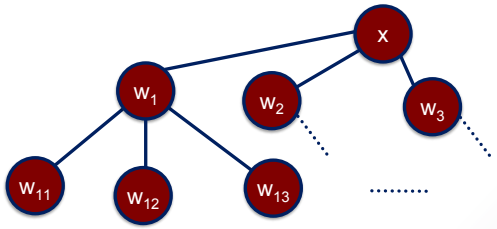
Traversal of Graphs

- To traverse a graph means to visit the vertices of the graph in some systematic order.
- In some applications, we may need to do some processing at every vertex of a graph.
- Two traversal strategies: **Breadth First Search (BFS)** and **Depth First Search (DFS)**, are two efficient ways to 'visit' each vertex and edge exactly once.

Main idea of BFS

Breadth First Search (BFS)

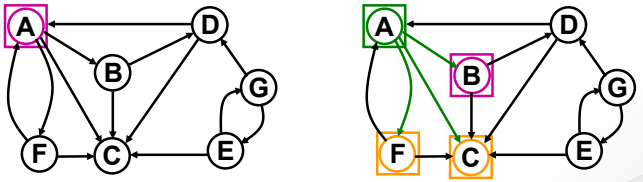
Given a graph **G** = (**V**, **E**) and a source vertex *x*, BFS systematically explores the edges directly connected to *x* before visiting vertices further away.



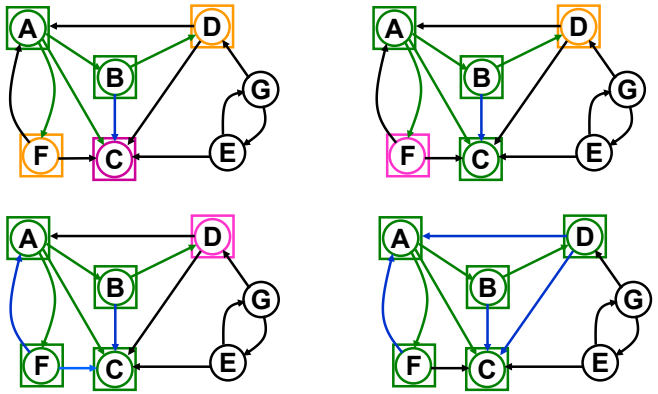
An Example of BFS

Breadth First Search (BFS)

- A queue is used to monitor which vertices to visit next.
- Action taken during "visit v" depends on specific applications.



An Example of BFS (continue)



Pseudocode of BFS

Breadth First Search (BFS)

```
void BFS(graph G, int s, queue L) {
        mark vertex s;  /* starting vertex */
        put s into the empty queue L;
        tree T = s;
        while (L is not empty) {
                remove a vertex v from front of queue L;
                visit v;
                for (each neighbour w of v in graph G)
                        if (w is unmarked) {
                                mark w;
                                add w to end of queue L;
                                add edge vw to T;
                        }
        }
}
```

Using adjacency lists, w is found in O(1) time, what if using adjacency matrix?

Recap of BFS

Breadth First Search (BFS)

- If a vertex has several unmarked neighbours, it would be equally correct to visit them in any order.
- If the *shortest path* from ***s*** to any vertex ***v*** is defined as the path with the **minimum** number of edges, then BFS finds the shortest paths from ***s*** to all vertices reachable from ***s***.
- The tree built by BFS is called the **breadth first spanning tree** (when graph *G* is connected).

Time Complexity of BFS

Breadth First Search (BFS)

- Each edge is processed once in the while loop for a total cost of $\Theta(|E|)$
- Each vertex is queued and dequeued once, for a total cost of $\Theta(|V|)$
- Worst-case time complexity for BFS is
  - $\Theta(|V| + |E|)$, if graph is represented by an array of **adjacency lists**
  - $\Theta(|V|^2)$, if graph is represented by an **adjacency matrix**, since for each vertex it may take $\Theta(|V|)$ to scan for its neighbours

Depth First Search (DFS)

Main Idea of DFS

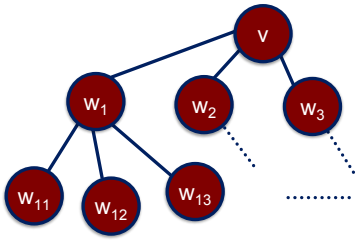
Depth First Search (DFS)

Suppose we start our traversal at vertex v and $w_1, w_2, \ldots, w_k$ are vertices adjacent to v:

- When visiting vertex **v**, we go to visit w_1 and keep $w_2, \ldots, w_k$ waiting
- When visiting w_1, we traverse all vertices adjacent to it before **backtracking** to **v** and traversing $w_2, \ldots, w_k$

Pseudocode of DFS

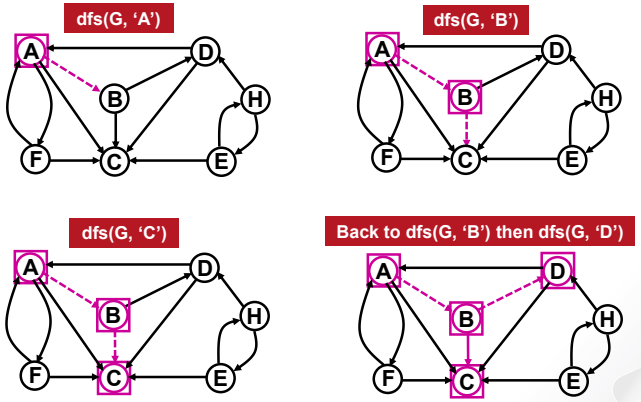
Depth First Search (DFS)



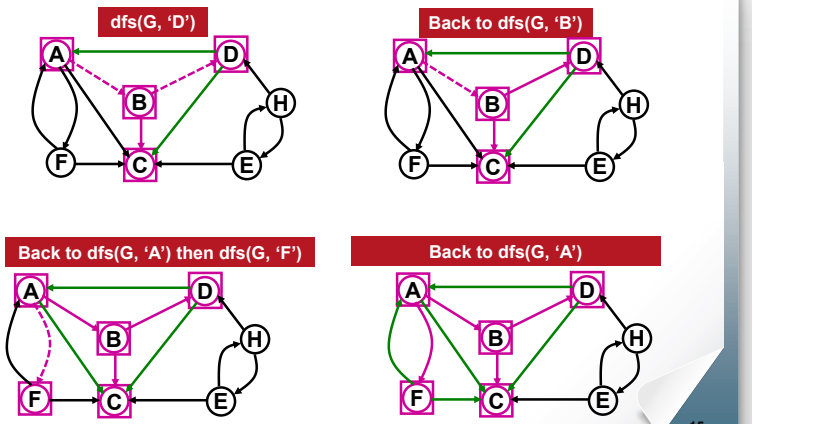

```
dfs(G,  v) {
    mark v as 'visited';
    for each neighbour w of v
        if w is unvisited  {
            dfs(G, w);
            add edge vw to tree T
        }
}
```

A stack is used implicitly

Example of Running DFS



Example of Running DFS



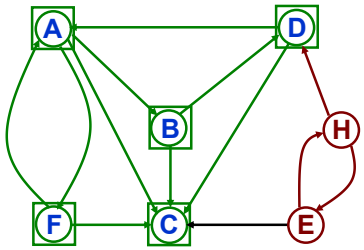
Traversal of Multiple Components

Depth First Search (DFS)

For a graph which has a few connected components, **dfs( )** needs to be repeated:

```
dfsSweep(G)
{
    mark all vertices 'unvisited';
    for each vertex v in G
        if  v is unvisited
            dfs(G, v);
}
```

Traversal of Second Component



Recap of DFS

Depth First Search (DFS)

Some Points About the DFS:

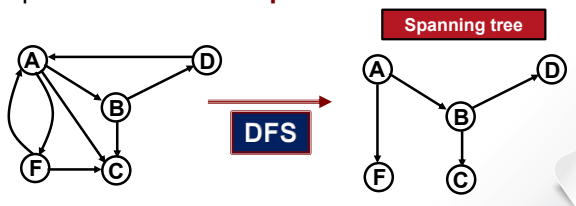
- If a vertex has several neighbours it would be equally correct to go through them in any order.
- When a DFS backs up from a dead end, it is supposed to **branch out** from the most recently discovered vertex before exploring new paths from vertices that were discovered earlier.

Recap of DFS

Depth First Search (DFS)

Some Points About the DFS (Cont.):

- If the graph is strongly connected, the tree T, constructed by the DFS algorithm is a spanning tree, i.e., a set of |V|-1 edges that connect all vertices of the graph. T is called the **depth first search tree**.



Time Complexity of DFS

- The DFS algorithm visits each node **exactly once**; **every edge** is traversed once in forward direction (**exploring**) and once in backward direction (**backtracking**).
- Therefore, using adjacency-lists, time complexity of DFS is $O(|V| + |E|)$.

```
dfs(G, v) {
    mark v as 'visited';
    for each neighbour w of v
        if w is unvisited {
            dfs(G, w);
            add edge vw to tree T;
        }
}
```

Every edge is checked

NANYANG TECHNOLOGICAL UNIVERSITY

Backtracking

NANYANG TECHNOLOGICAL UNIVERSITY

Backtracking

- Depth First Search applied to a dynamically generated tree is called **Backtracking**.
- It is a systematic way to search for the solution(s) to a problem.
  1. We first define a solution space for the problem. The space must include at least one (optimal) solution to the problem.
  2. The starting node is the initial state of the problem.
  3. Then we search the solution space in a depth first manner beginning at a starting node.

NANYANG TECHNOLOGICAL UNIVERSITY

Backtracking

- Depth First Search applied to a dynamically generated tree is called **Backtracking**.
- It is a systematic way to search for the solution(s) to a problem.
  4. During the search, whenever we encounter a dead end, we backtrack to the most recently seen node and continue the search.
  5. The search terminates when we have found the answer or when we run out of nodes to backtrack to.

NANYANG TECHNOLOGICAL UNIVERSITY

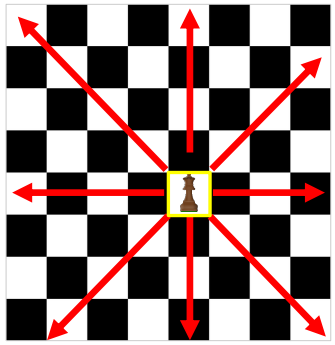
First Example of Backtracking

Eight-Queens Problem

- In chess, a queen can move as far as she pleases, horizontally, vertically, or diagonally.
- A chess board has 8 rows and 8 columns. The standard 8 by 8 Queen's problem asks **how to place 8 queens on an ordinary chess board so that *none* of them can conquer any other in *one* move.**

Reference: Staxringold. (2006). Chess board opening staunton [Photograph]. Retrieved from https://commons.wikimedia.org/wiki/File:Chess_board_opening_staunton.jpg

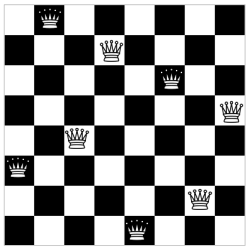
Directions of Queen's move

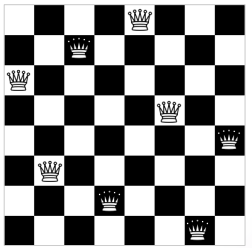


References:
- Chessboard black-white [Image]. (2013). Retrieved from https://pixabay.com/en/board-chess-chessboard-black-white-157165/
- Formax. (2010). Chess piece- queen [Image]. Retrieved from https://commons.wikimedia.org/wiki/File:Schach_Dame_schwarz.svg

Expected Output

Two Example Solutions to Eight-Queens Problem





References:
- Chessboard black-white [Image]. (2013). Retrieved from https://pixabay.com/en/board-chess-chessboard-black-white-157165/
- Cburnett. (2006). Light queen on transparent square [Image]. Retrieved from https://commons.wikimedia.org/wiki/File:Chess_qlt45.svg

Backtracking Algorithm

The First Step: Define a solution space before searching for solutions. Very often, the solution space of a problem may be represented by a graph.

The Algorithm:

1. Starts by placing a queen in the top left corner of the chess board.
2. It then places a queen in the second column and moves her until it finds a place where she cannot be hit by the queen in the first column.

Reference: MichaelMaggs. (2007). Chess piece- white queen Staunton design [Photograph]. Retrieved from https://commons.wikimedia.org/wiki/File:Chess_piece_-_White_queen.jpg

Backtracking Algorithm

The Algorithm (Cont.):

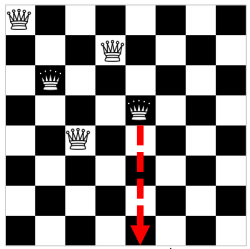
3. It then places a queen in the third column and moves her until she cannot be hit by either of the first two queens. Then it continues this process with the remaining columns.

4. If there is no place for a queen in the current column *i* the program goes back (**backtracks**), to move the queen in preceding column *i* – 1.
5. If the queen in column *i* – 1 is at the end of the column it removes that queen as well (**backtracks again**) and goes to the preceding column *i* – 2, and so on.

Reference: MichaelMaggs. (2007). Chess piece- white queen Staunton design [Photograph]. Retrieved from https://commons.wikimedia.org/wiki/File:Chess_piece_-_White_queen.jpg

Meet a Dead End

The Algorithm (Cont.):



Backtrack by moving to another slot; if no suitable position, backtrack the previous queen.

Unable to slot queen in this column

References:
- Chessboard black-white [Image]. (2013). Retrieved from https://pixabay.com/en/board-chess-chessboard-black-white-157165/
- Cburnett. (2006). Light queen on transparent square [Image]. Retrieved from https://commons.wikimedia.org/wiki/File:Chess_qlt45.svg

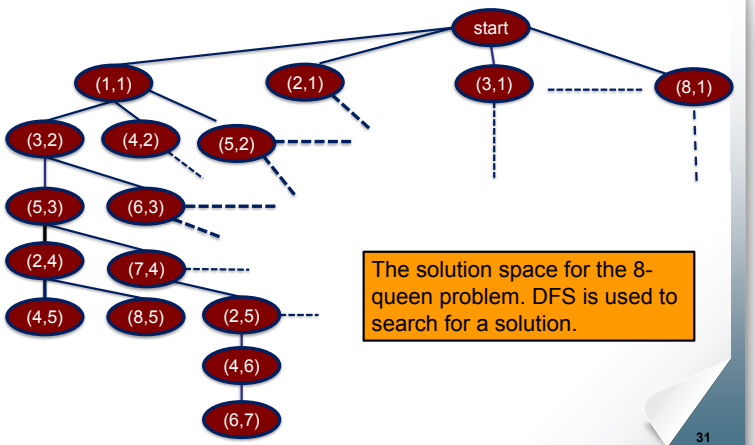
Backtracking Algorithm (End)

The Algorithm (Cont.):

6. If the current column is the last column and a safe place has been found for the last queen, then a solution to the puzzle has been found.

7. If the current column is the first column and its queen is being moved off the board then all possible configurations have been examined, all solutions have been found, and the algorithm terminates.

Solution Space



The solution space for the 8-queen problem. DFS is used to search for a solution.

Recursive Algorithm

Program Outline

Recursive function:

solve_from (Queens configuration) {
  if Queens configuration *already contains eight queens*
    print configuration
  else
    for *every chessboard square* p *that is unguarded* by configuration
    { *add a queen on square* p to configuration;
      solve_from(configuration);
      remove the queen from square p of configuration;
    }
}

2nd Example of Backtracking

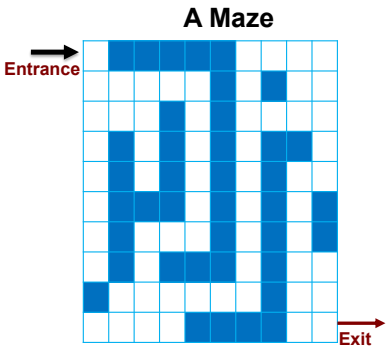
Another Example of Backtracking Application

Navigating Through a Maze

Objective: To walk through a maze containing walls or obstacles from a given entrance to exit, i.e. to find a path, which is a sequence of positions (none blocked) such that each position is **N/S/E/W** of the previous position.

Representation of a Maze

Another Example of Backtracking Application

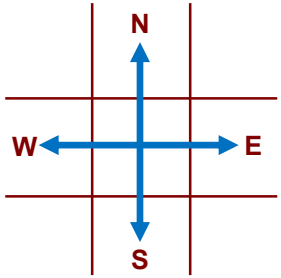


0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	1	0	1	0	0
0	0	0	1	0	1	0	0	0	0
0	1	0	1	0	1	0	1	1	0
0	1	0	1	0	1	0	1	0	0
0	1	1	1	0	1	0	1	0	1
0	1	0	0	0	1	0	1	0	1
0	1	0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0

Matrix representation of a maze. The **zero** entries are the nodes in the solution space.

Directions of Next Move

Four Options for a Move from Any Position (provided no obstacle is in the way):



Pseudocode of Algorithm

```
Boolean findPath() {    // find a path from (1, 1) to (m, m)
    position here = (1,1);
    stack s;  // initialised as an empty stack
    mark maze[here] as visited;

    while (here is not (m, m)) {
        find an unvisited neighbour to move to;
        if (there is such a neighbour to move to) {
            push here onto the stack s;
            here = neighbour;
            mark maze[here] as visited;
        } else {
            if (stack s is empty) return false;
            here = pop(stack s);  // backtrack
        }
    }
    return true;
}
```

Summary

- Two elementary algorithms for graph traversal
  - Breadth-first search (BFS): Use queue
  - Depth-first search (DFS): Use stack
  - Marking nodes to avoid duplicate visits
- Time complexity of BFS or DFS:
  - Using adjacency lists: Θ (|V| + |E|)
  - Using adjacency matrix: Θ ($|V|^2$)
- Applications of DFS in Artificial Intelligence:
  - Eight-queens problem
  - Navigating through a maze