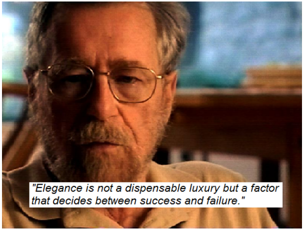

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 15 Breadth First Search

Jiamou Liu
The University of Auckland

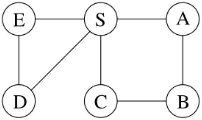


Traversing a Graph

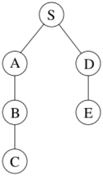
Uses for Graph Traversals

- Searching (DFS)
- Reachability (DFS)
- Decomposing graphs (DFS)
- Calculating distances (DFS is not suitable)

Graph G



The DFS Tree of G

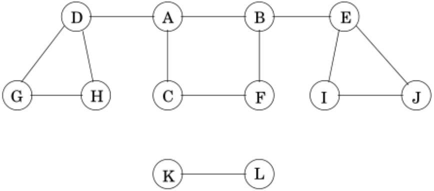


Distances Between Nodes

Definition

Let G be a graph

- The length of a path is the number of edges ("steps") in it.
- The distance from a node u to v, $dist(u, v)$, is the length of the shortest path from u to v. If no path exist, then $dist(u, v) = \infty$.

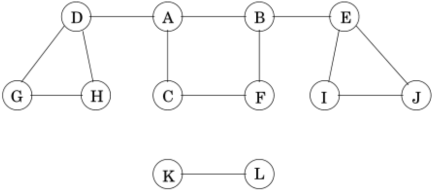


Distances Between Nodes

Definition

Let G be a graph

- The length of a path is the number of edges ("steps") in it.
- The distance from a node u to v, $dist(u, v)$, is the length of the shortest path from u to v. If no path exist, then $dist(u, v) = \infty$.



- Distance between D and J: 4. Shortest path: D, A, B, E, J
- Distance between A and K: ∞

We aim to solve the following two problems:

Distance Problem

INPUT: a graph G, and two nodes u, v
OUTPUT: the distance from u to v.

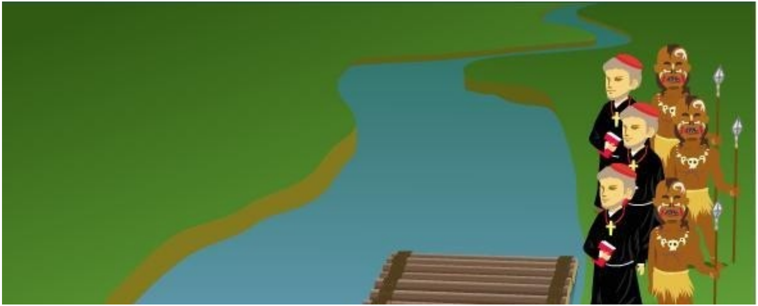
Shortest Path Problem

INPUT: a graph G, and two nodes u, v
OUTPUT: the shortest path from u to v.

Case Study: Missionary and Cannibals

Missionaries and Cannibals

There are 3 missionaries and 3 cannibals coming to a river with only one boat that can hold only 2 people. At any instance the number of cannibals cannot be more than the number of missionaries. What is the least number of boat rides for all the people to cross the river?



This is a graph problem:

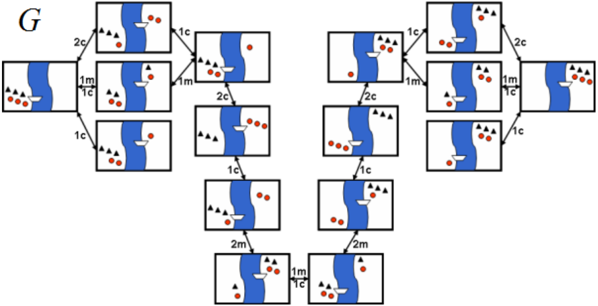
- We call a time stamp of the current situation a configuration.
- A configuration states how many missionaries and cannibals on each bank, and the location of the boat.

  E.g. $(MC\bullet, MMCC)$ indicates 1 missionary + 1 cannibal on left bank, 2 missionaries + 2 cannibals on right bank, boat on left bank

This is a graph problem:

- We call a time stamp of the current situation a configuration.
- A configuration states how many missionaries and cannibals on each bank, and the location of the boat.

  E.g. $(MC\bullet, MMCC)$ indicates 1 missionary + 1 cannibal on left bank, 2 missionaries + 2 cannibals on right bank, boat on left bank
- Define a graph $G = (V, E)$ where
  - The nodes are all configurations
  - Two configurations are connected by an edge if it is possible to move from one configuration to the other using one boat ride.



Goal: Find a shortest path from $(, \bullet MMMCCC)$ to $(MMMCCC\bullet,)$.

Breadth First Search

How to solve the distance and the shortest path problem?

Breadth First Search

How to solve the distance and the shortest path problem?

Breadth First Search Strategy

Traverse nodes by "layers":

1. Visit the start node s
2. Visit all nodes that have distance 1 from s (call them V_1)
3. Visit all nodes that have distance 1 from V_1 (call them V_2)
4. Visit all nodes that have distance 1 from V_2 (call them V_3)
5. ... ...

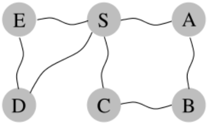
Breadth First Search

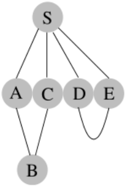
How to solve the distance and the shortest path problem?

Breadth First Search Strategy

Traverse nodes by "layers":

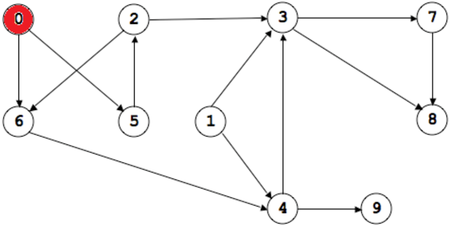
1. Visit the start node s
2. Visit all nodes that have distance 1 from s (call them V_1)
3. Visit all nodes that have distance 1 from V_1 (call them V_2)
4. Visit all nodes that have distance 1 from V_2 (call them V_3)
5. ... ...





Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:
- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



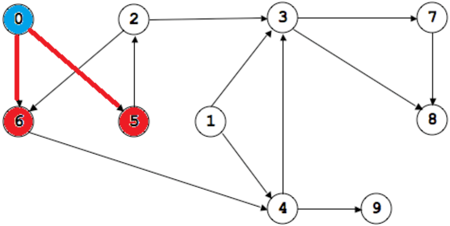
$Q = [0]$

Breadth First Search

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.

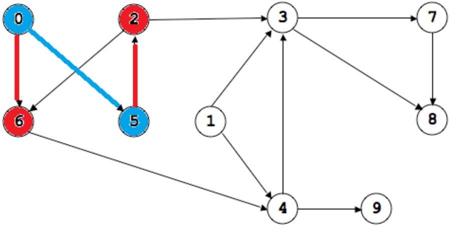


$Q = [5, 6]$

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



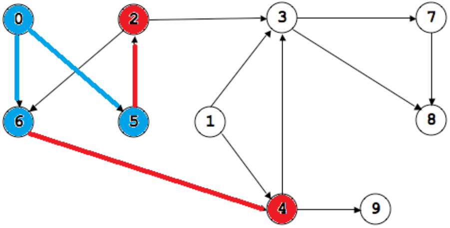
$Q = [6, 2]$

Breadth First Search

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.

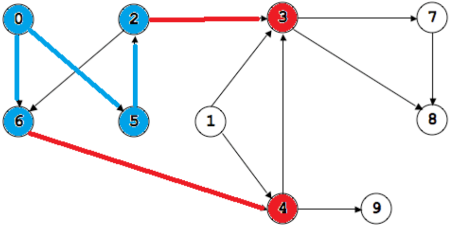


$Q = [2, 4]$

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.

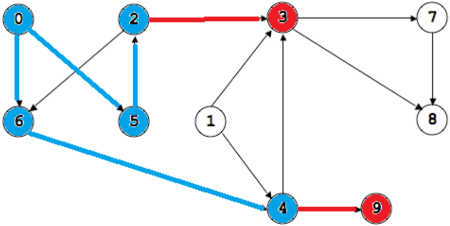


$Q = [4, 3]$

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



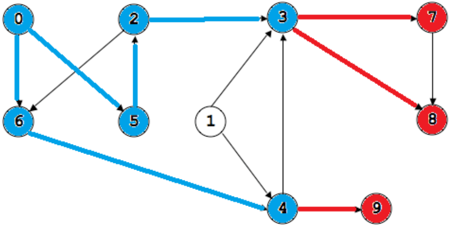
$Q = [3, 9]$

Breadth First Search

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

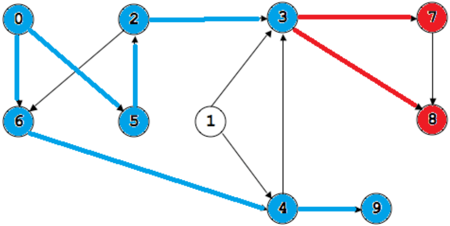
- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



$Q = [9, 7, 8]$

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:
- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



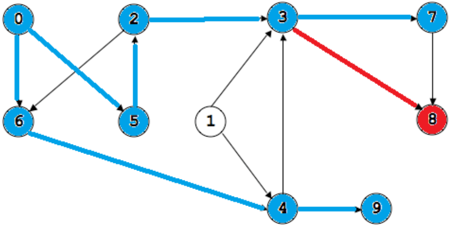
$Q = [7, 8]$

Breadth First Search

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



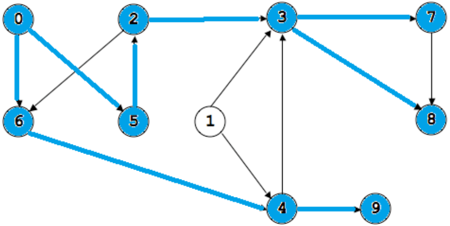
$Q = [8]$

Breadth First Search

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



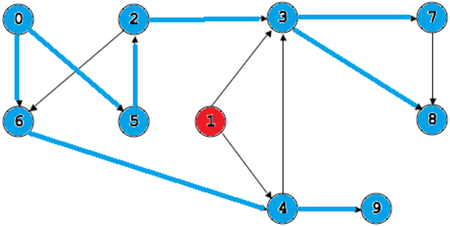
$Q = []$

Breadth First Search

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:

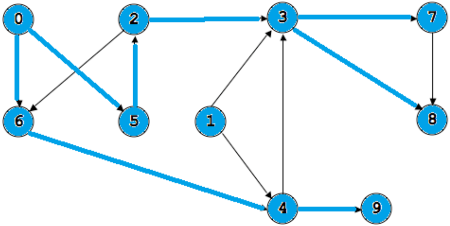
- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



$Q = [1]$

Queue implementation of BFS

Maintain a queue of to-be-explored nodes.
At each iteration:
- First finish the first element in the queue; dequeue.
- Then enqueue the out-neighbors of the dequeued element.



$Q = []$

Breadth First Search

Maintain a field $dist(u)$ for every node u.

Procedure bfs(G)

INPUT: A graph G and a starting node s
OUTPUT: Labeling $dist(u)$ for every u
for $u \in V$ **do** $dist(u) \leftarrow \infty$
for $u \in V$ **do**
    if $dist(u) = \infty$ **then** run bfs_explore(G, u)

Procedure bfs_explore(G, s)

INPUT: A graph G and a starting node s
$dist(s) \leftarrow 0$
Create an empty queue Q and enqueue(Q, s)
while Q is not empty **do**
    $u \leftarrow$ dequeue(Q) (Note u is first in Q)
    for any outgoing edge (u, v) **do**
        if $dist(v) = \infty$ **then**
            enqueue(Q, v)
            $dist(v) \leftarrow dist(u) + 1$

Breadth First Search

Theorem.

After running bfs_explore(G, s), the value of $dist(u)$ is the distance from s to u for every node u.

Breadth First Search

Theorem.

After running bfs_explore(G, s), the value of $dist(u)$ is the distance from s to u for every node u.

Proof. We use the notion of known region: Say a node u is in the known region if $dist(u) \neq \infty$.

- We need to prove that all reachable nodes u from s eventually enters the known region, and that $dist(u)$ will be correctly calculated.
- Suppose, for a contradiction, that u is a nearest node that will not enter the known region.
- Say $dist(u) = k > 0$. Then there is a node v with $dist(v) = k - 1$ and $(v, u) \in E$.
- Then v would enter the known region and all outgoing edges of v will be examined.
- Thus u will enter the known region and $dist(u)$ will be k. Contradiction $\square$

Breadth First Search - Complexity

Complexity Analysis

In running a BFS on G:

- Every node u in G may be enqueued and dequeued at most once
- Every edge may be checked at most once

Breadth First Search - Complexity

Complexity Analysis

In running a BFS on G:

- Every node u in G may be enqueued and dequeued at most once
- Every edge may be checked at most once

Therefore the running time of the BFS algorithm is $O(m + n)$.

Summary: DFS versus BFS

SUMMARY

DFS: The algorithm makes deep but narrow exploration into the graph.

Only retreat when it runs out of new nodes.

Implementation: Use a stack as an auxiliary data structure. Can also be implemented recursively.

Running Time: $O(m + n)$.

Applications: This could be used for analyzing reachability, linearizability, connectedness.

Summary: DFS versus BFS

DFS: The algorithm makes deep but narrow exploration into the graph.

Only retreat when it runs out of new nodes.

Implementation: Use a stack as an auxiliary data structure. Can also be implemented recursively.

Running Time: $O(m + n)$.

Applications: This could be used for analyzing reachability, linearizability, connectedness.

BFS: The algorithm makes shallow but broad exploration into the graph.

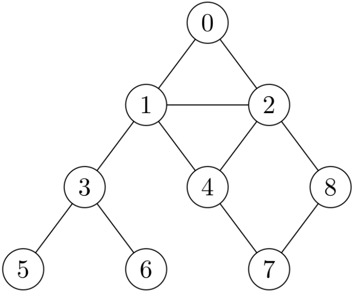
Visit nodes by increasing distances.

Implementation: Could use a queue as an auxiliary data structure.

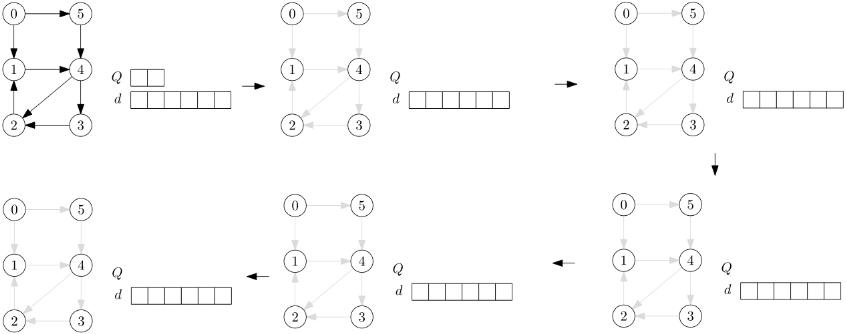
Running Time: $O(m + n)$

Applications: This could be used for computing distances.

Question 1. On the graph below, run both DFS and BFS starting from node 0. Draw the corresponding search trees.



Question 2. Execute BFS starting at 0 and fill out the values for the queue Q and distances d.




THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND