# Graph

- Introduction
- Graph API
- Elementary Graph Operations
  - DFS: Depth first search
  - **BFS: Breadth first search**
  - CC: Connected Components

Major references:
1. Fundamentals of Data Structures by Horowitz, Sahni, Anderson-Freed,
2. Algorithms 4th edition - Part 1 & Part 2 by Robert Sedgewick and Kevin Wayne
3. Wikipedia and many resources available from internet


Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

## Design pattern for graph processing

**Design pattern:** Decouple graph data type
**Idea:** Mimic maze exploration

**DFS (to visit a vertex v)**

- **Mark v as visited.**
- **Recursively visit all unmarked vertices w adjacent to v.**

**Typical applications:**

- Find all vertices connected to a given source vertex.
- Find a path between two vertices.

**Challenge:**

- How to implement?

## Breadth-first search

**Depth-first search:** Put unvisited vertices on a **stack**.
**Breadth-first search:** Put unvisited vertices on a **queue.**

**Shortest path:** Find path from s to t that uses fewest number of edges.

**BFS:** (from source vertex s)

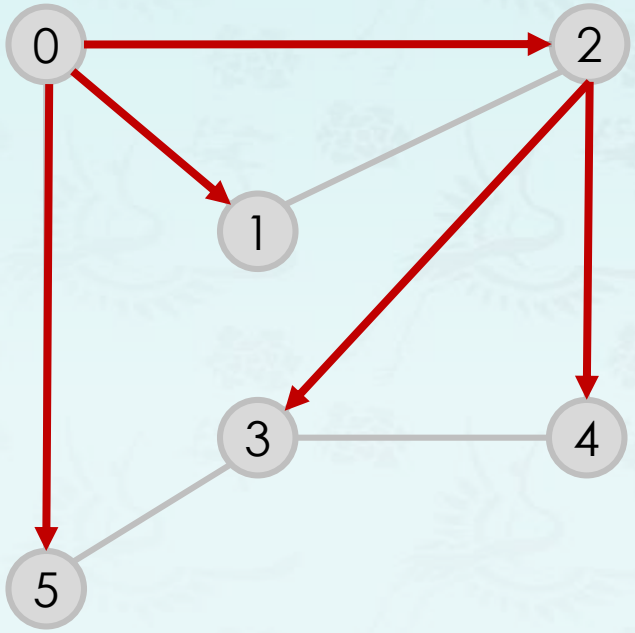
- Put s onto a FIFO queue, and mark s as visited.
- Repeat until the queue is empty:
  - remove the least recently added vertex v
  - add each of v's unvisited neighbors to the queue,
    and mark them as visited.

**Intuition:** BFS examines vertices in increasing distance from s.

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




**graph4.txt**

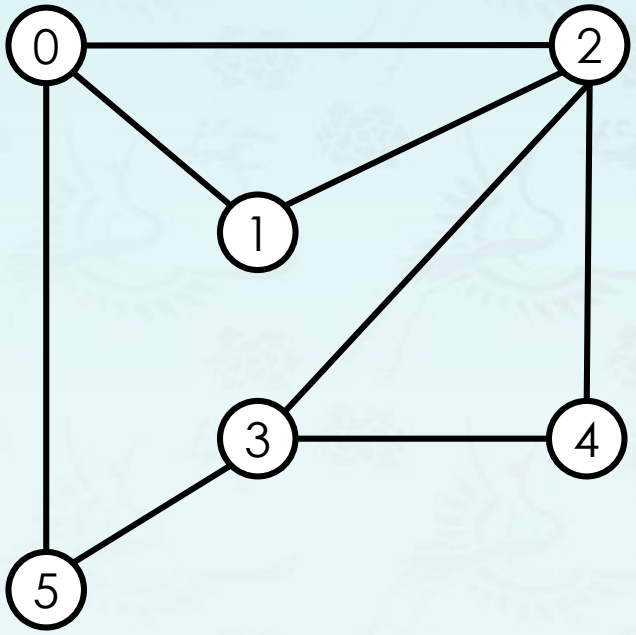
| v | parent[v] | distTo[] |
|---|---|---|
| 0 | - | 0 |
| 1 | 0 | 1 |
| 2 | 0 | 1 |
| 3 | 2 | 2 |
| 4 | 2 | 2 |
| 5 | 0 | 1 |

done

## Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




graph4.txt

```
6        <-- V
8        <-- E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

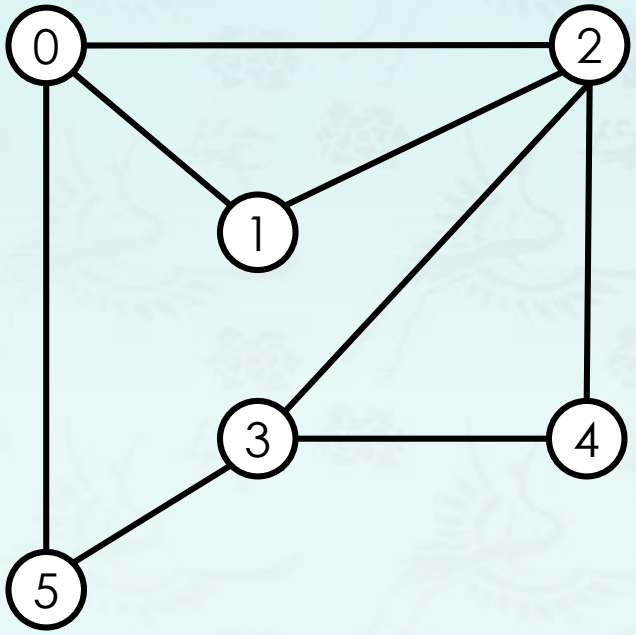
Graph g:

**Challenge:** build adjacency lists?

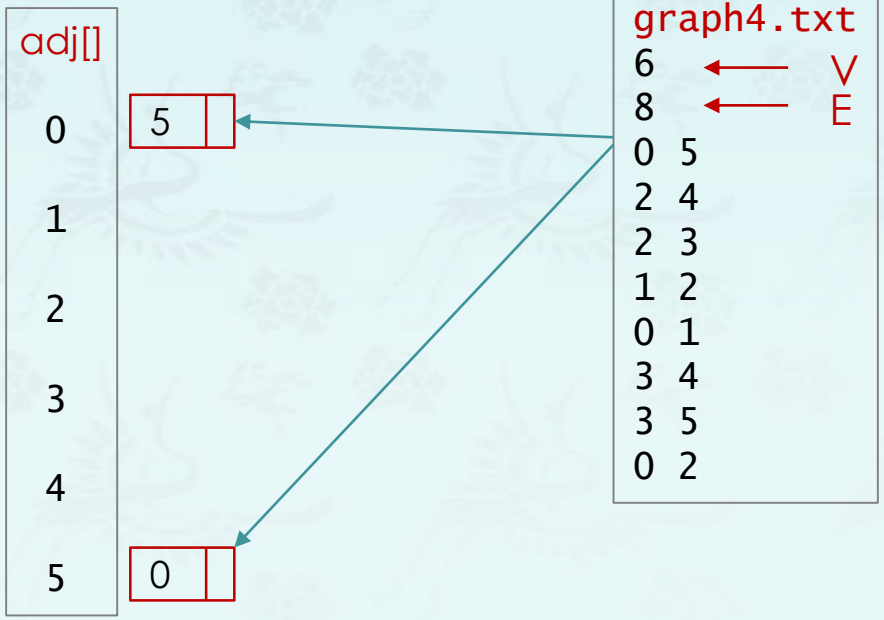
# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




Adjacency lists



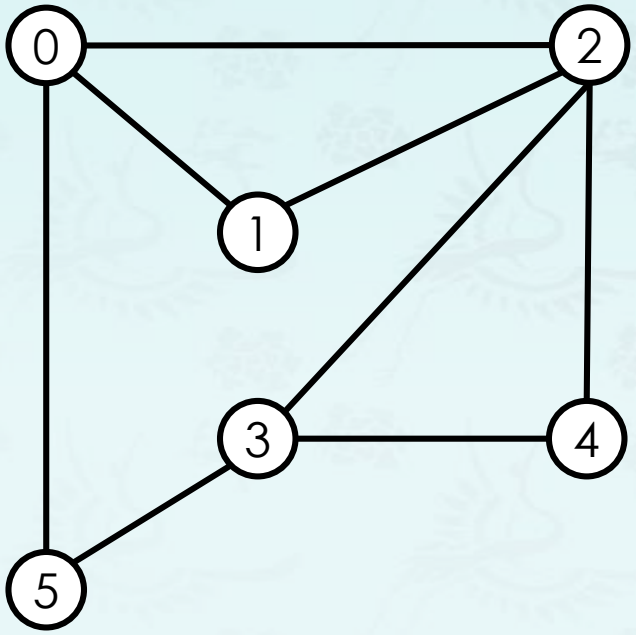

Graph g:

**Challenge:** build adjacency lists?

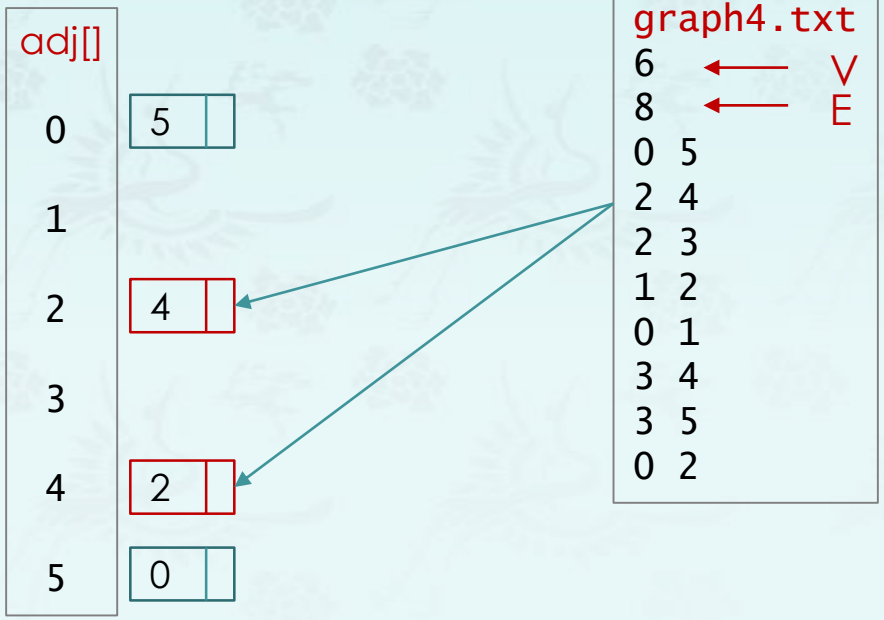
# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




Adjacency lists



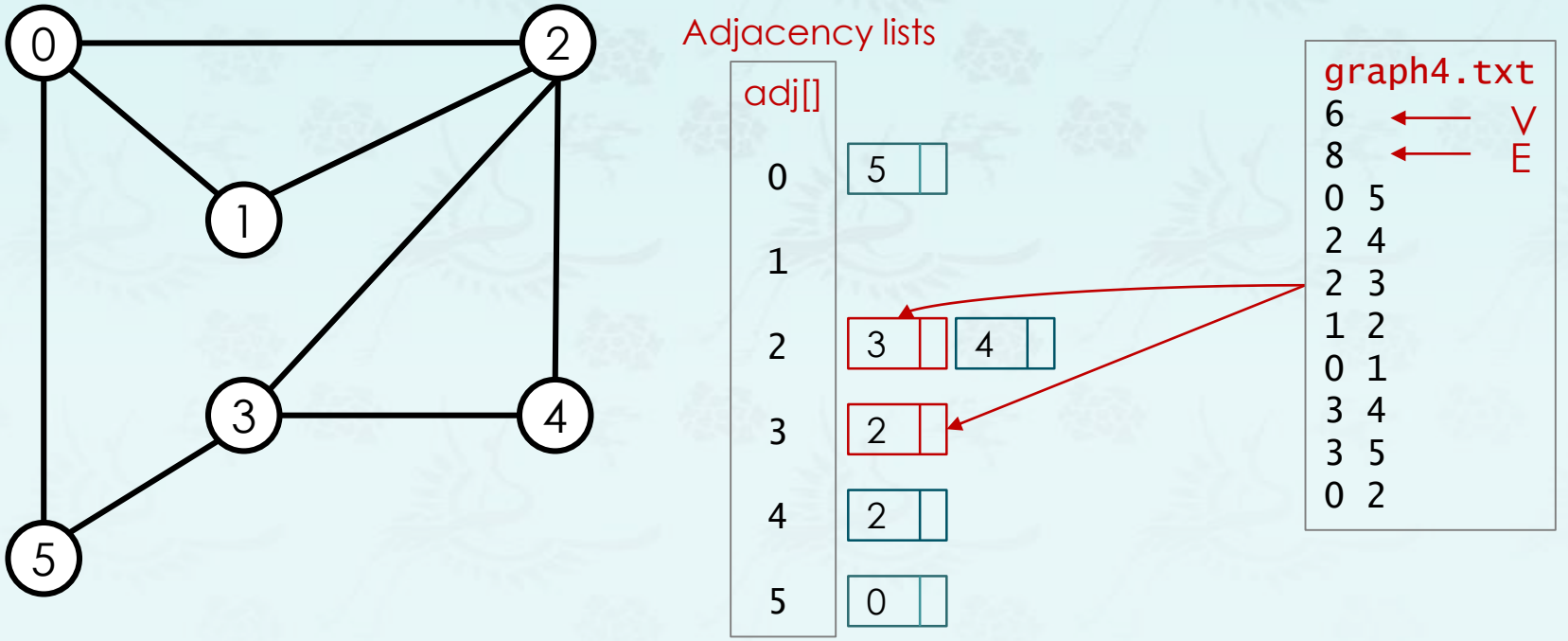

Graph g:

**Challenge:** build adjacency lists?

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.

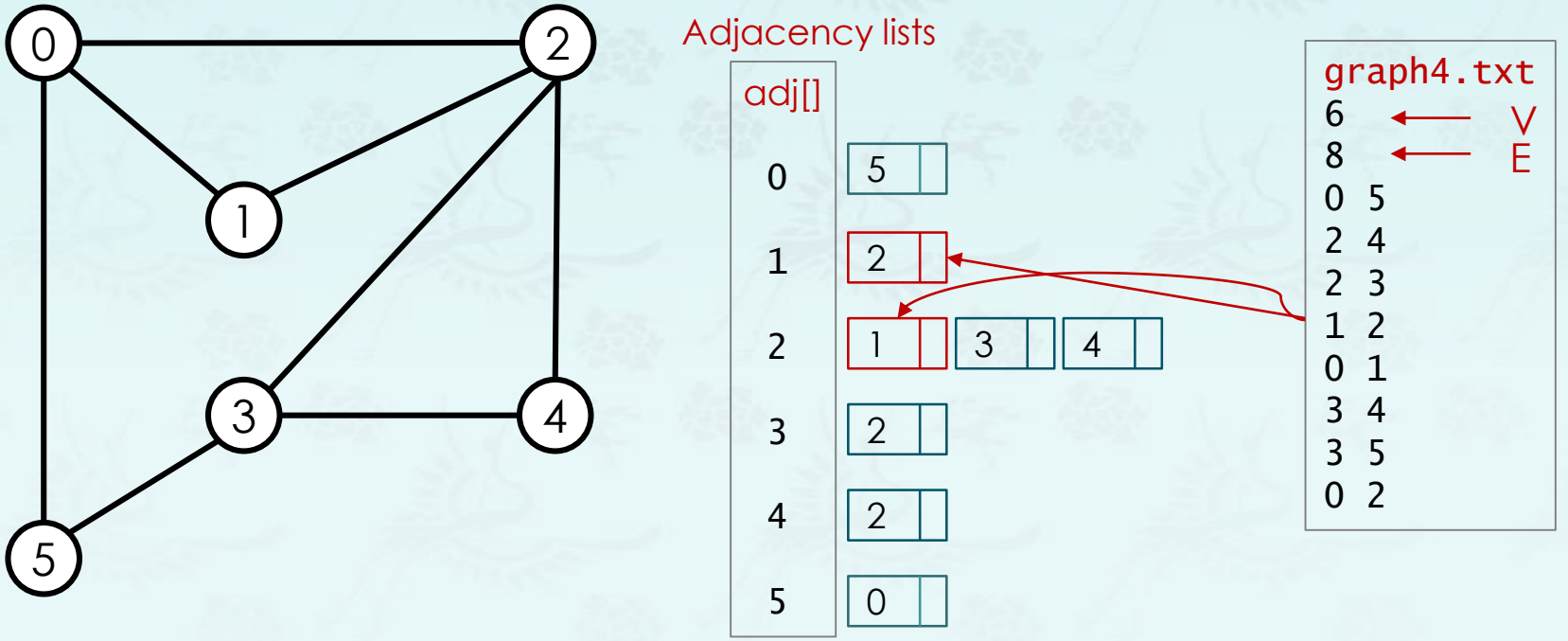



Graph g:

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.

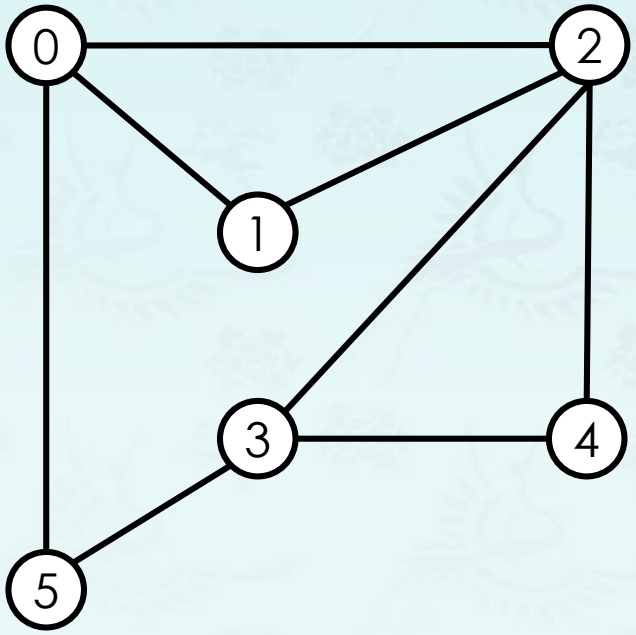



Graph g:

# Breadth-first search demo

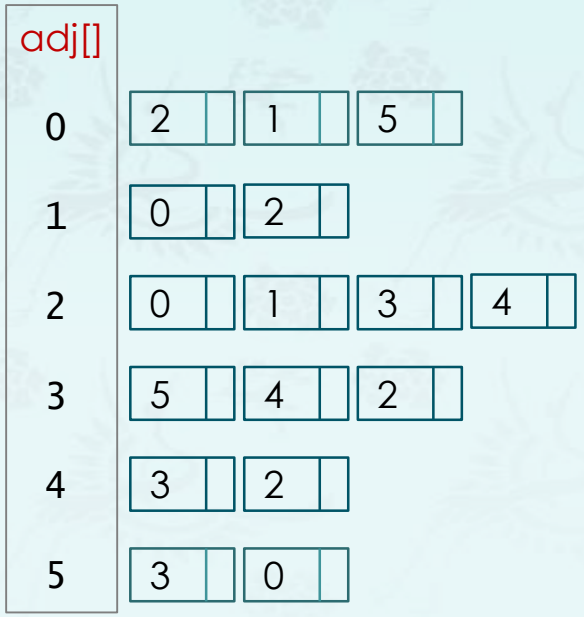
**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




Graph g:

Adjacency lists



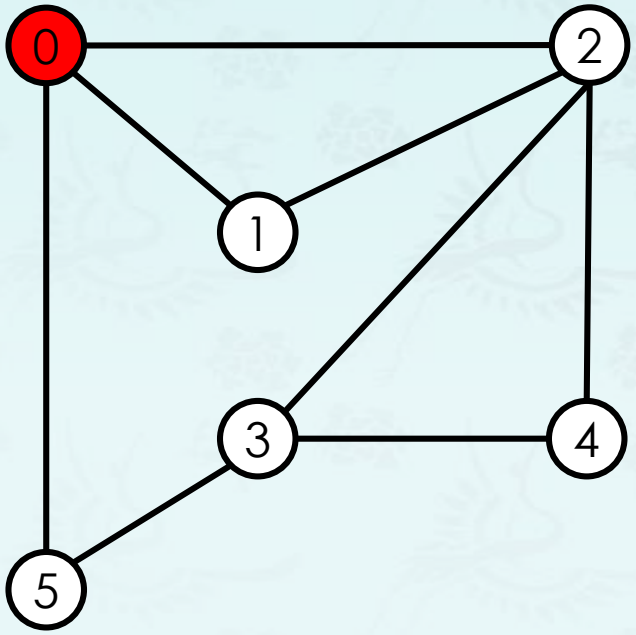

```
graph4.txt
6        <-- V
8        <-- E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

**Challenge:** build adjacency lists – Job done

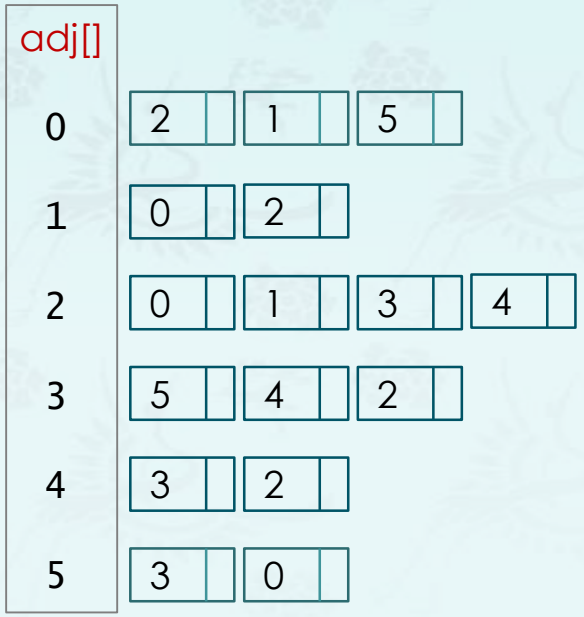
## Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




Adjacency lists



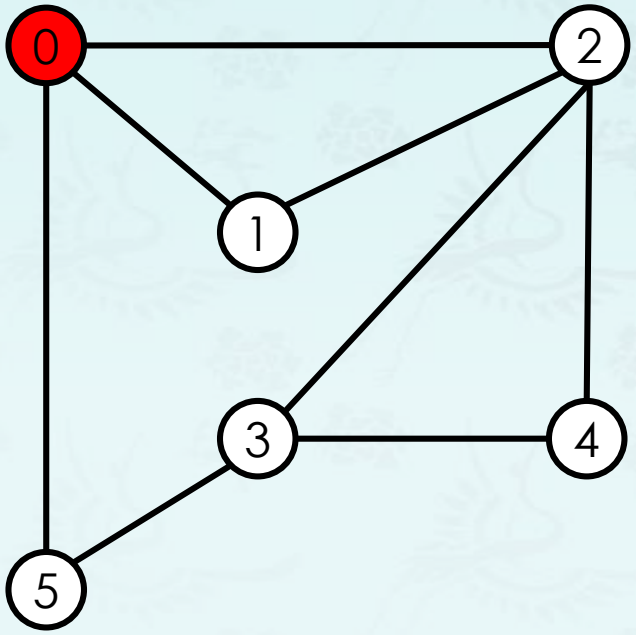

```
graph4.txt
6        <-- V
8        <-- E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

add 0 to queue:

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



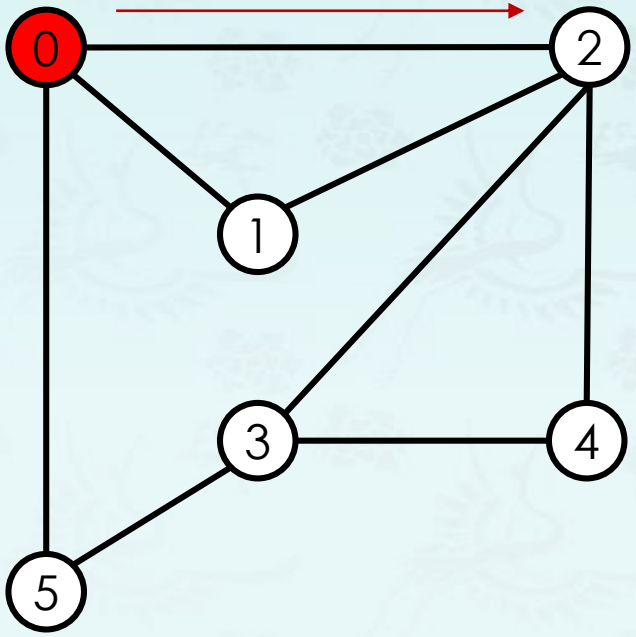

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | - | - |
| | 2 | - | - |
| | 3 | - | - |
| | 4 | - | - |
| 0 | 5 | - | - |

add 0 to queue:

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | - | - |
| | 2 | 0 | 1 |
| | 3 | - | - |
| | 4 | - | - |
| 0 | 5 | - | - |

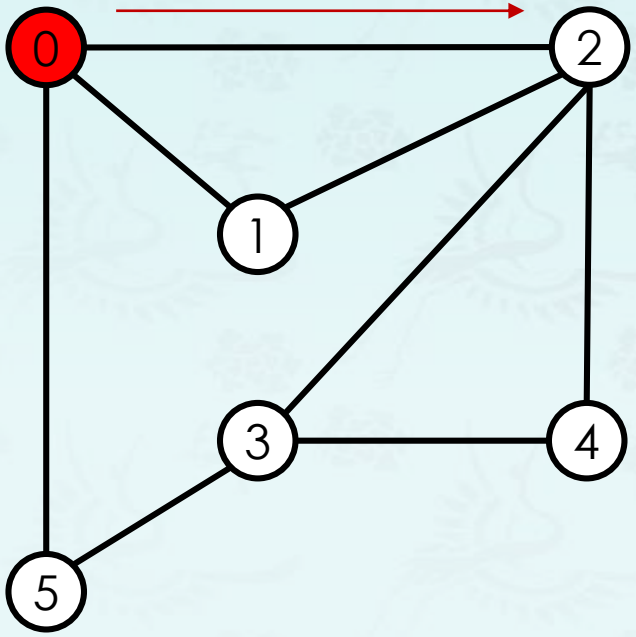
adj[0] 2 → 1 → 5

dequeue 0: check2, check 1 and check 5

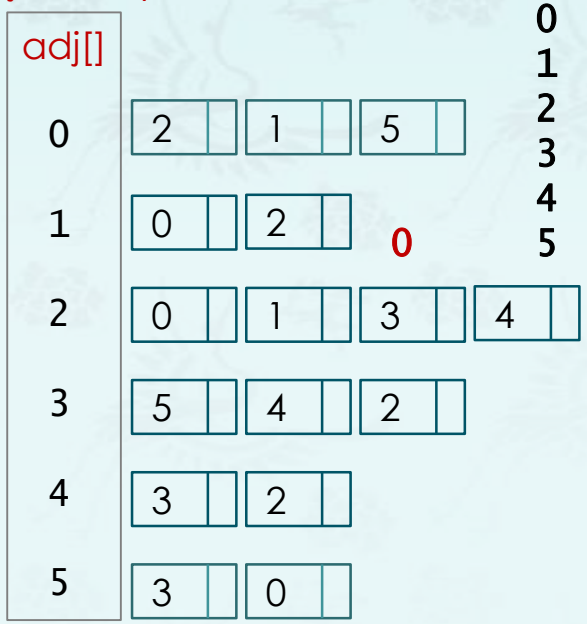
# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




Adjacency lists




| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | – | 0 |
| | 1 | – | – |
| | 2 | 0 | 1 |
| | 3 | – | – |
| | 4 | – | – |
| | 5 | – | – |
| 0 | | | |

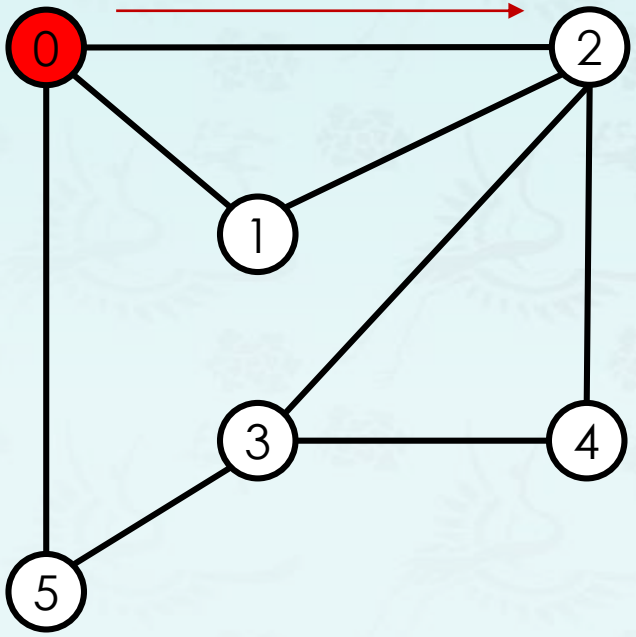
adj[0] 2 1 5

dequeue 0: check2, check 1 and check 5

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.




| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | - | - |
| | 2 | 0 | 1 |
| | 3 | - | - |
| | 4 | - | - |
| 2 | 5 | - | - |

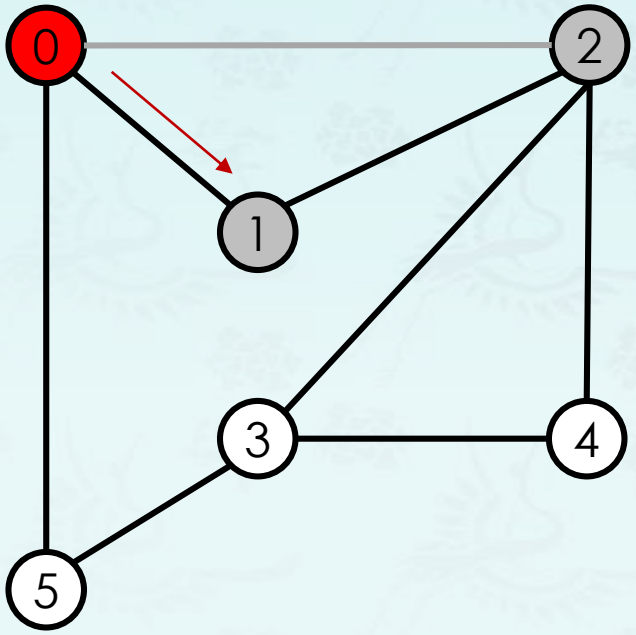
adj[0] 2 → 1 → 5

dequeue 0: check2, check 1 and check 5

# Breadth-first search demo

**Repeat until queue is empty:**

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.

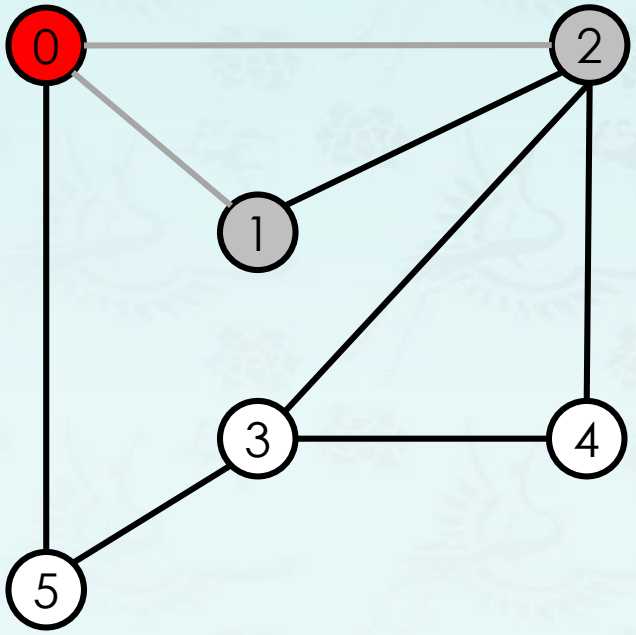



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | - | - |
| | 2 | 0 | 1 |
| | 3 | - | - |
| | 4 | - | - |
| 2 | 5 | - | - |

adj[0] 2 → 1 → 5

dequeue 0: check2, check 1 and check 5

# Breadth-first search demo

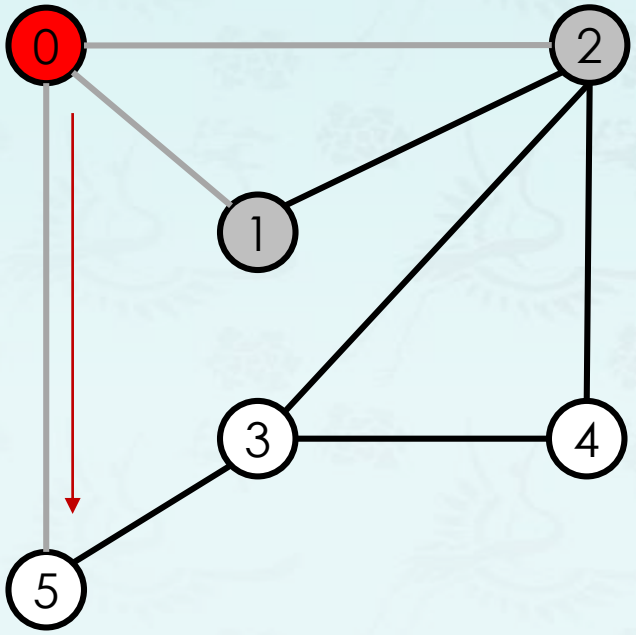



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | - | - |
| 1 | 4 | - | - |
| 2 | 5 | - | - |

adj[0] 2 → 1 → 5

dequeue 0: check2, check 1 and check 5

# Breadth-first search demo

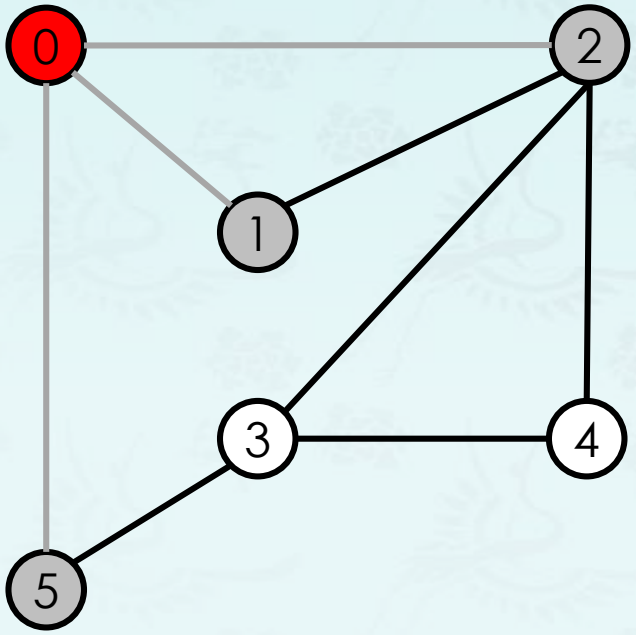



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | - | - |
| 1 | 4 | - | - |
| 2 | 5 | - | - |

adj[0] 2 → 1 → 5

dequeue 0: check2, check 1 and check 5

# Breadth-first search demo

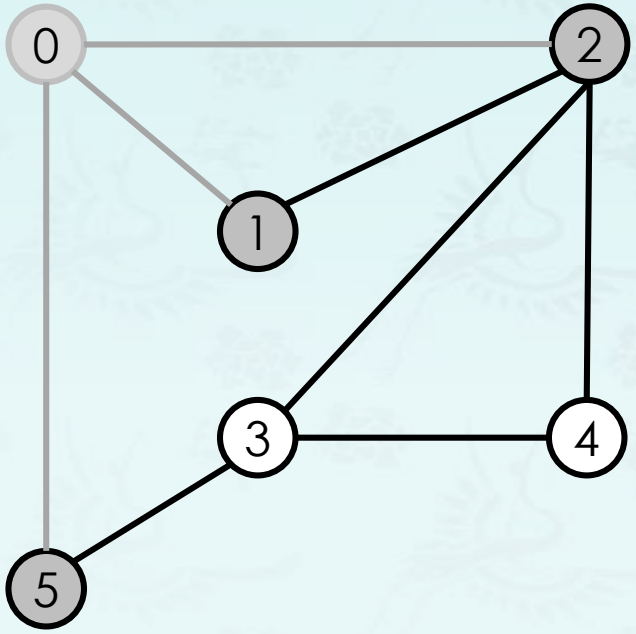



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| 5 | 3 | - | - |
| 1 | 4 | - | - |
| 2 | 5 | 0 | 1 |

adj[0] 2 → 1 → 5

dequeue 0: check2, check 1 and check 5

# Breadth-first search demo



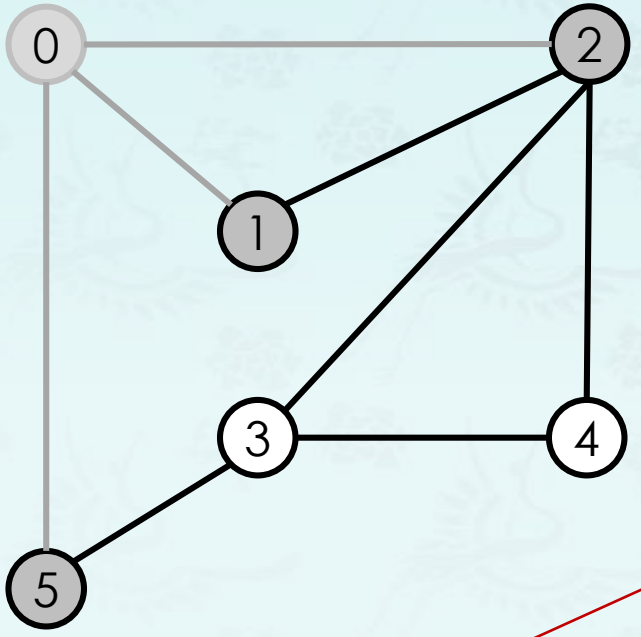

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| 5 | 3 | - | - |
| 1 | 4 | - | - |
| 2 | 5 | 0 | 1 |

adj[0] 2 → 1 → 5

0 done

BFS: 0

# Breadth-first search demo

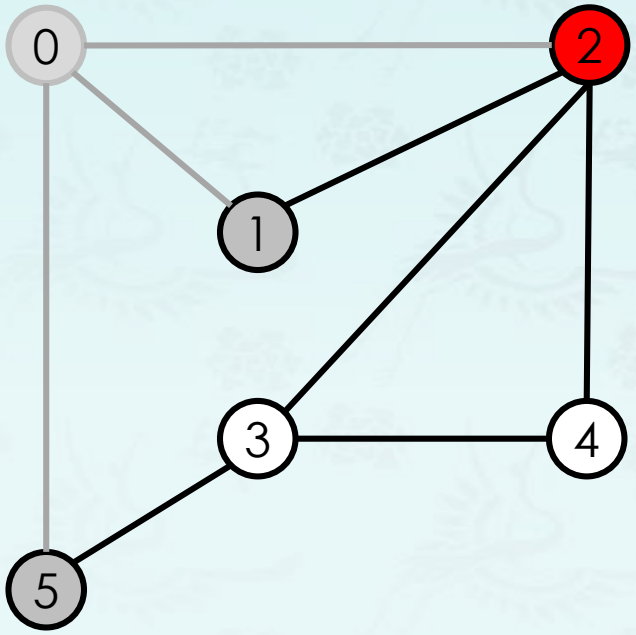



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| 5 | 3 | - | - |
| 1 | 4 | - | - |
| 2 | 5 | 0 | 1 |

dequeue 2:

BFS: 0

# Breadth-first search demo



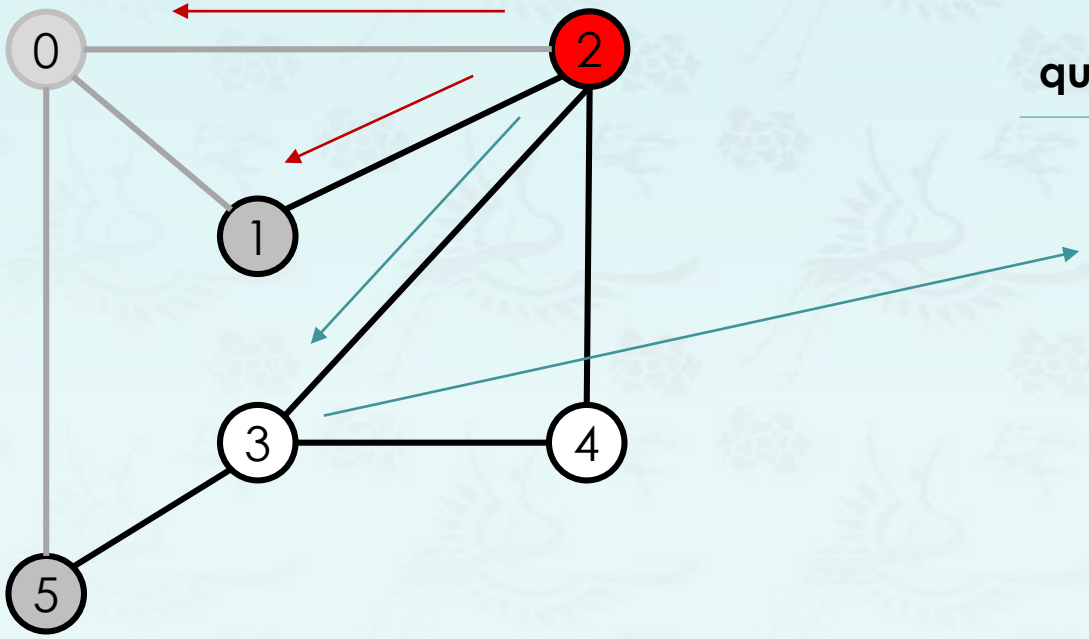

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| 5 | 3 | - | - |
| 1 | 4 | - | - |
| | 5 | 0 | 1 |

adj[2] 0 → 1 → 3 → 4

dequeue 2: check 0, check 1, check 3 and check 4

BFS: 0

# Breadth-first search demo



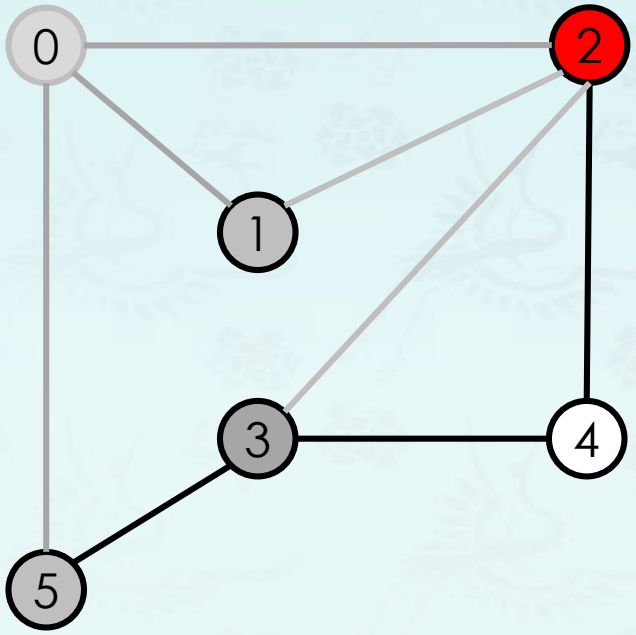

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| 5 | 3 | - | - |
| 1 | 4 | - | - |
| | 5 | 0 | 1 |

adj[2] 0 → 1 → 3 → 4

dequeue 2: check 0, check 1, check 3 and check 4

BFS: 0

# Breadth-first search demo



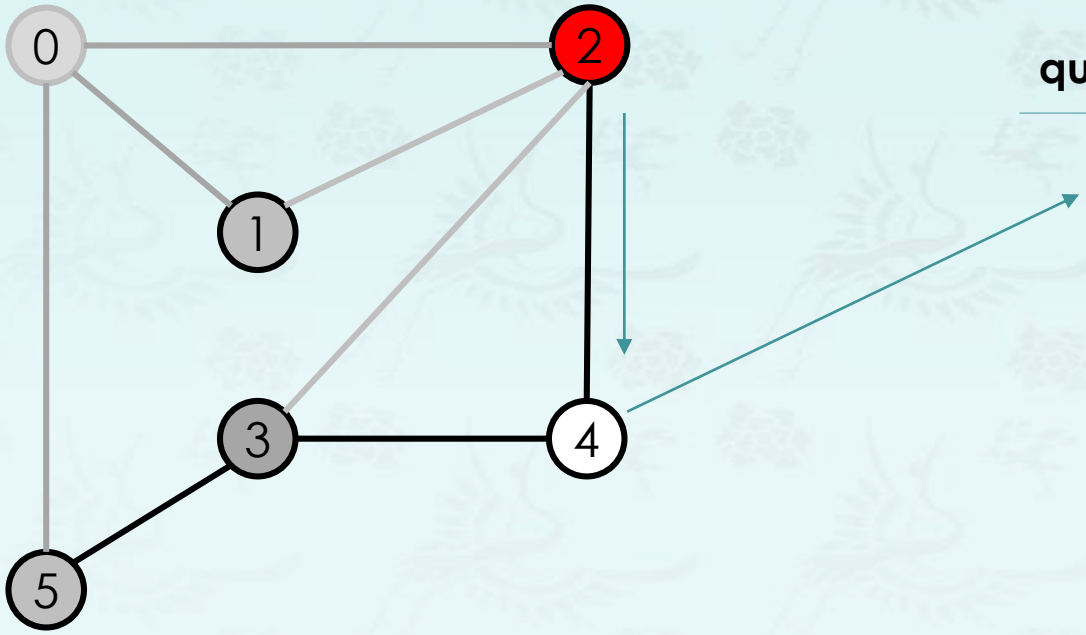

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| 1 | 4 | - | - |
| | 5 | 0 | 1 |

adj[2] 0 → 1 → 3 → 4

dequeue 2: check 0, check 1, check 3 and check 4

BFS: 0

# Breadth-first search demo



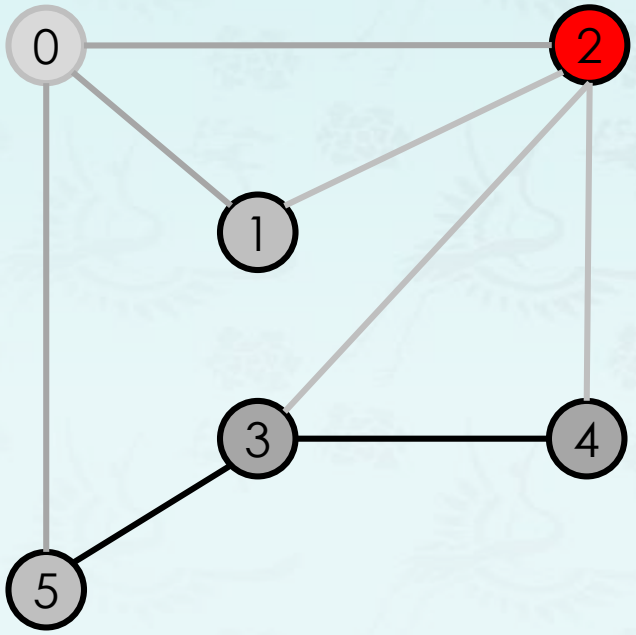

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| 1 | 4 | - | - |
| | 5 | 0 | 1 |

adj[2] 0 → 1 → 3 → 4

dequeue 2: check 0, check 1, check 3 and **check 4**

BFS: 0

# Breadth-first search demo



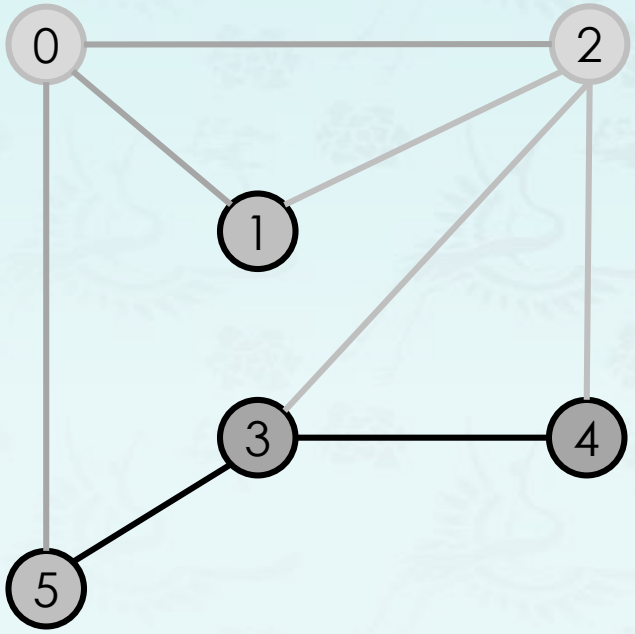

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| 1 | 4 | - | - |
| | 5 | 0 | 1 |

adj[2] 0 → 1 → 3 → 4

dequeue 2: check 0, check 1, check 3 and **check 4**

BFS: 0

# Breadth-first search demo



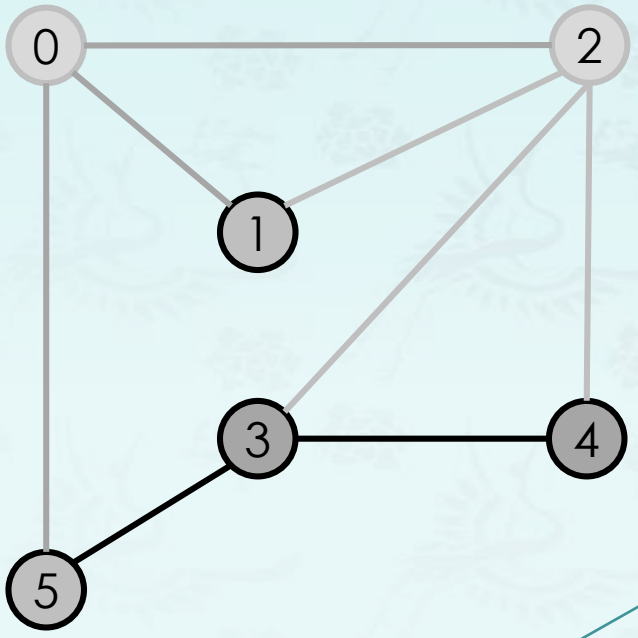

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| 1 | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[2] 0 → 1 → 3 → 4

2 done

BFS: 0 2

# Breadth-first search demo

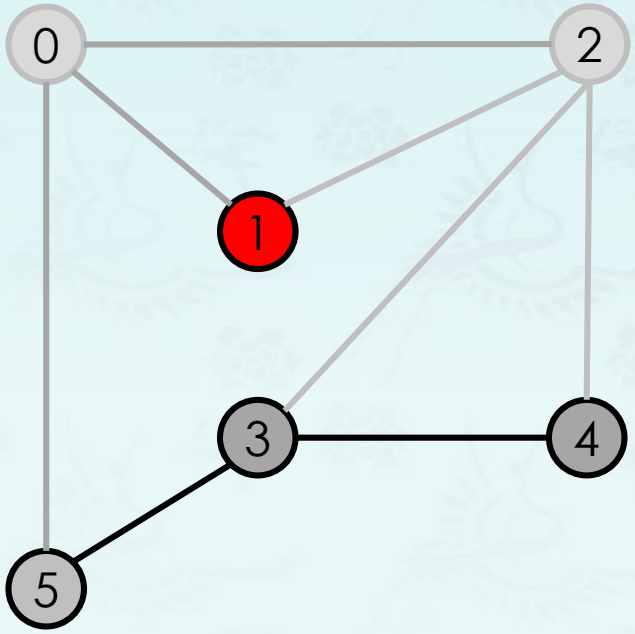



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| 1 | 4 | 2 | 2 |
| | 5 | 0 | 1 |

dequeue 1

BFS: 0 2

# Breadth-first search demo

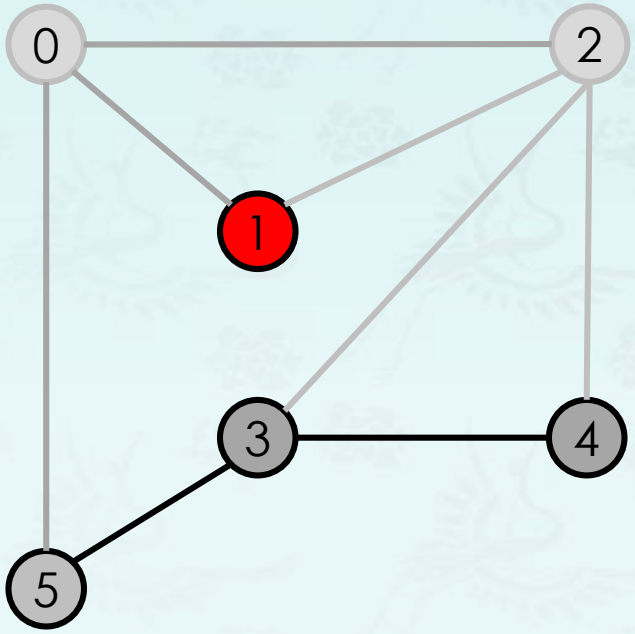



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

dequeue 1

BFS: 0 2

# Breadth-first search demo



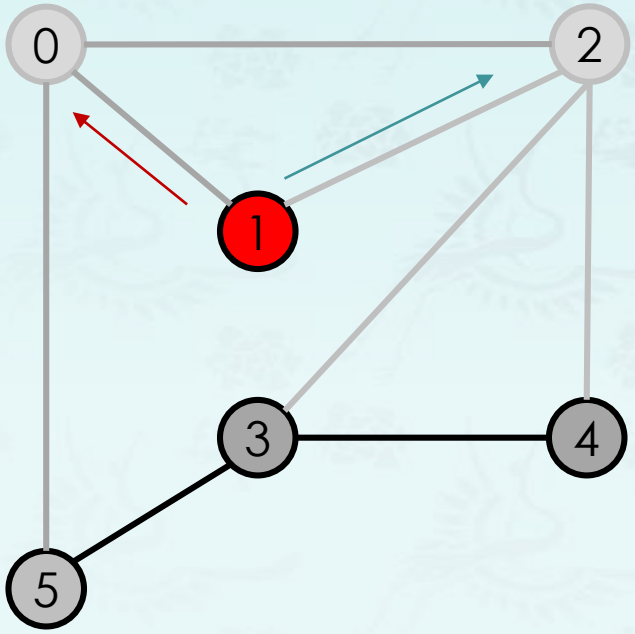

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[1] 0 | 2

dequeue 1: check 0, and check 2

BFS: 0 2

# Breadth-first search demo



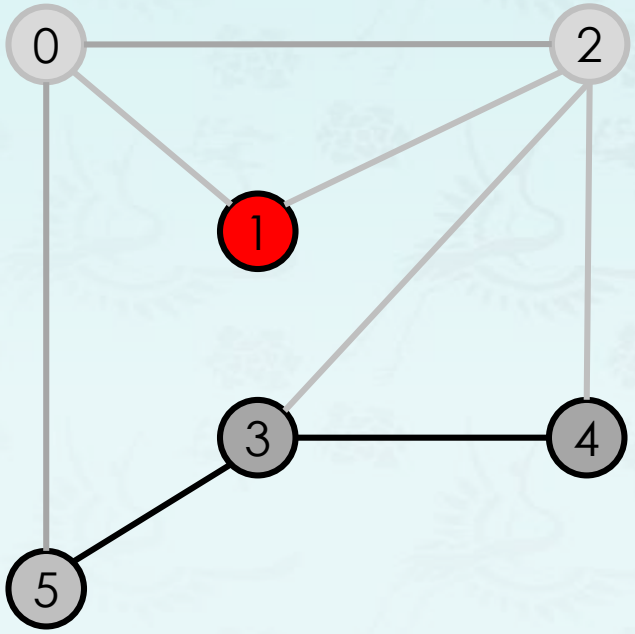

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[1] 0 2

dequeue 1: check 0, and check 2

BFS: 0 2

# Breadth-first search demo



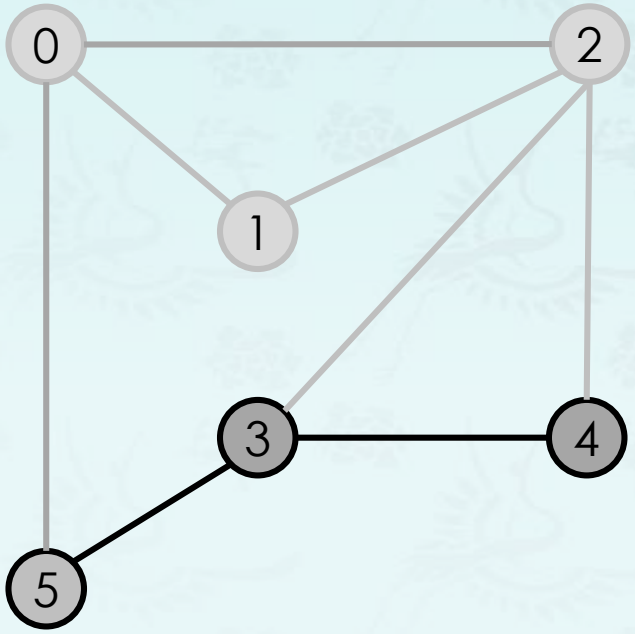

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[1] 0 | 2

dequeue 1: check 0, and check 2

BFS: 0 2

# Breadth-first search demo



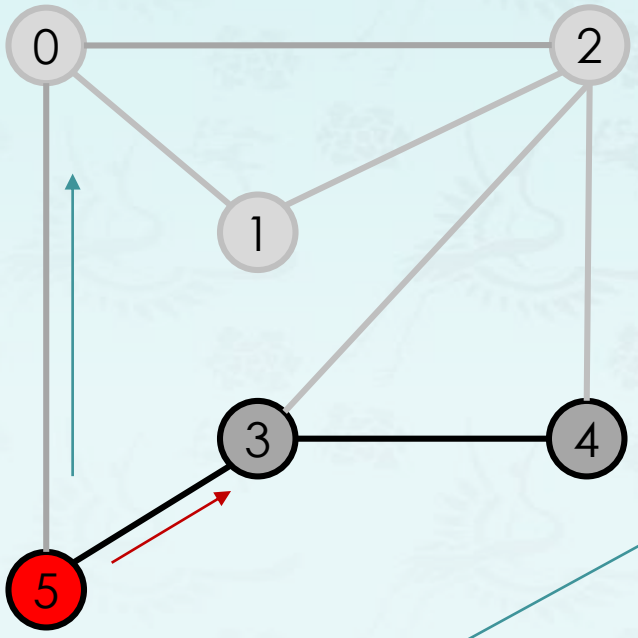

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[1] 0 2

1 done

BFS: 0 2 1

# Breadth-first search demo



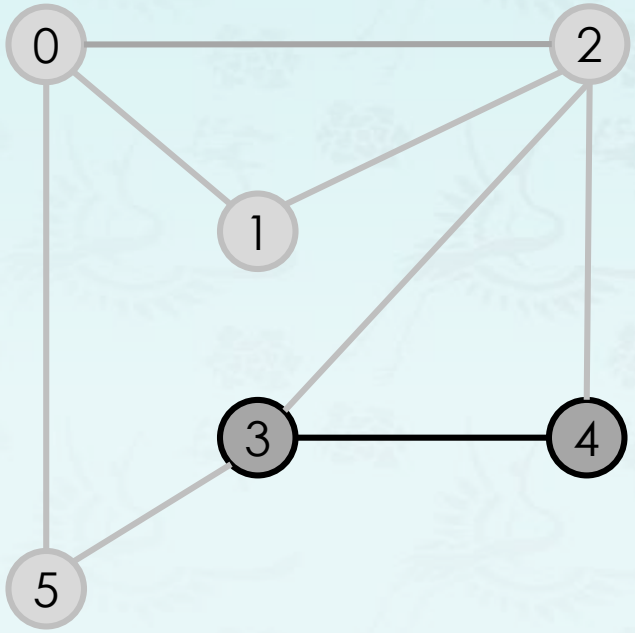

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| 5 | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[5] 3 0

dequeue 5: check 3 and check 0

BFS: 0 2 1

# Breadth-first search demo



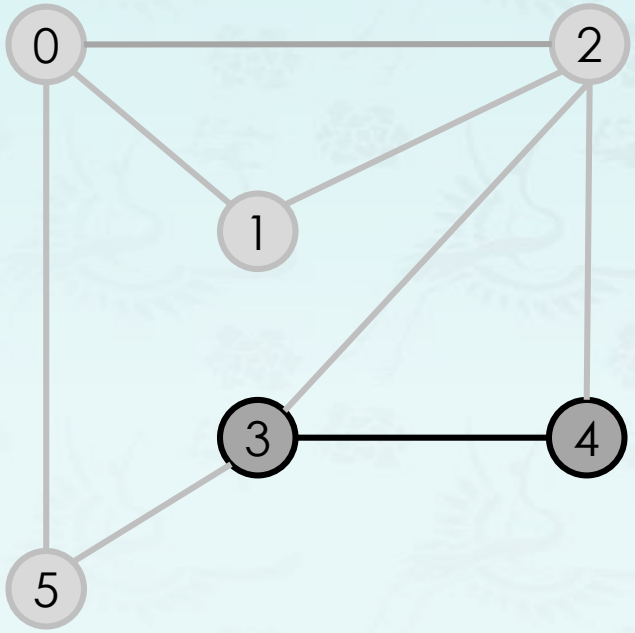

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[5] 3 | 0

5 done

BFS: 0 2 1 5

# Breadth-first search demo



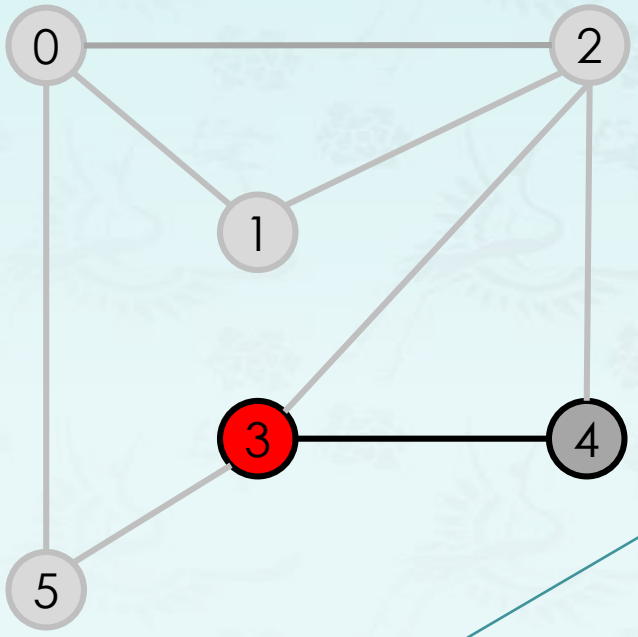

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[3] 5 → 4 → 2

dequeue 3: Check 5, Check 4, and Check 2

BFS: 0 2 1 5

# Breadth-first search demo

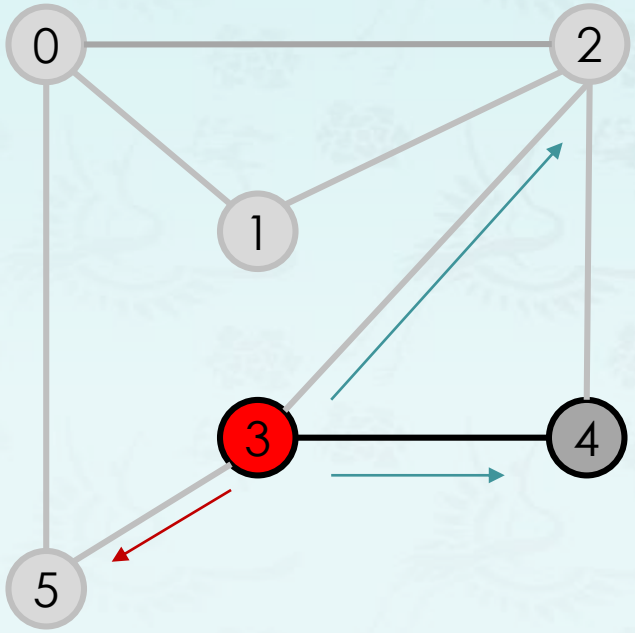



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| 3 | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

dequeue 3:

BFS: 0 2 1 5

# Breadth-first search demo



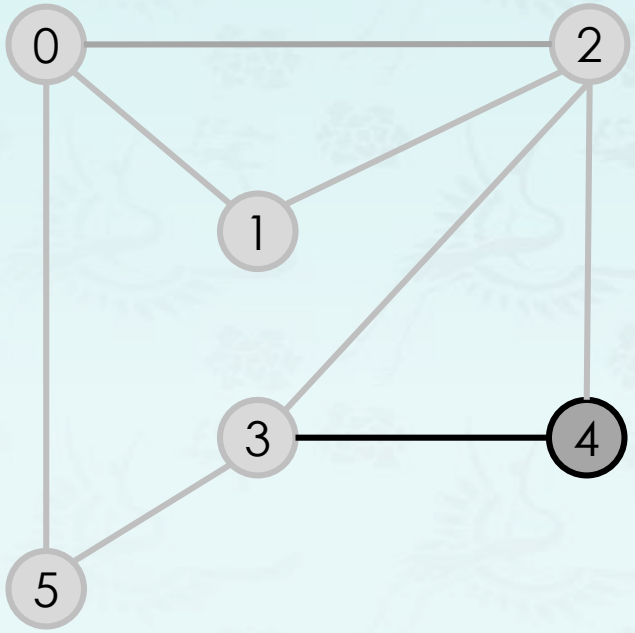

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[3] 5 → 4 → 2

dequeue 3: Check 5, Check 4, and Check 2

BFS: 0 2 1 5

# Breadth-first search demo



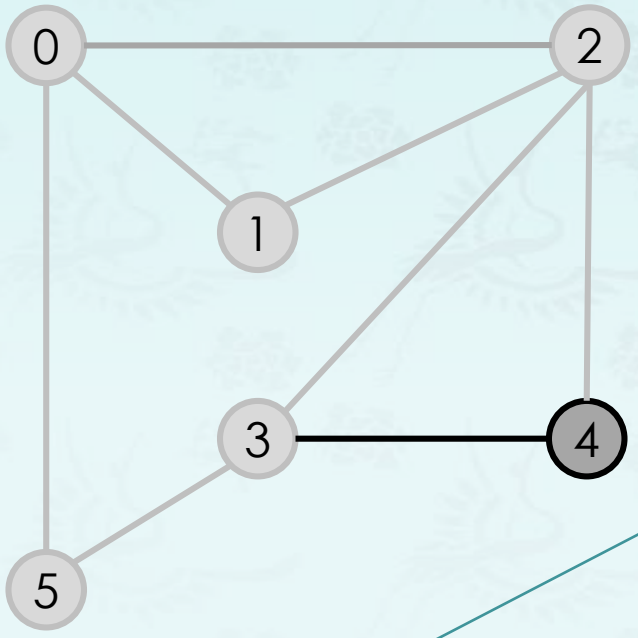

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[3] 5 → 4 → 2

3 done

BFS: 0 2 1 5 3

# Breadth-first search demo

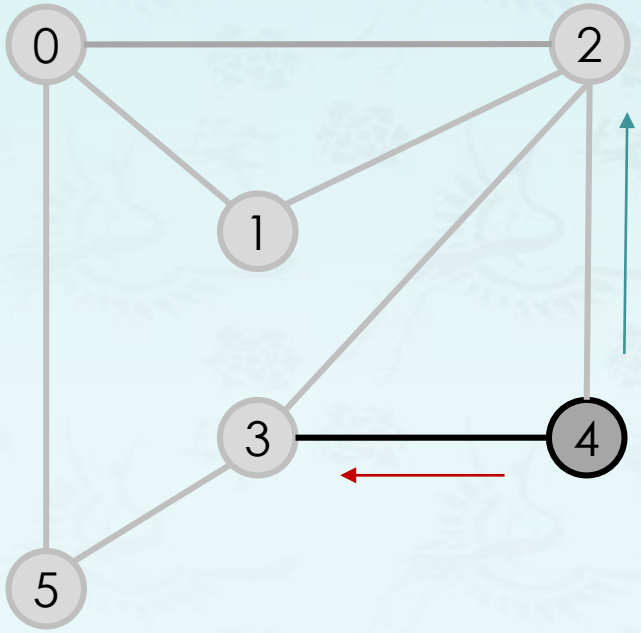



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| 4 | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

dequeue 4

BFS: 0 2 1 5 3

# Breadth-first search demo



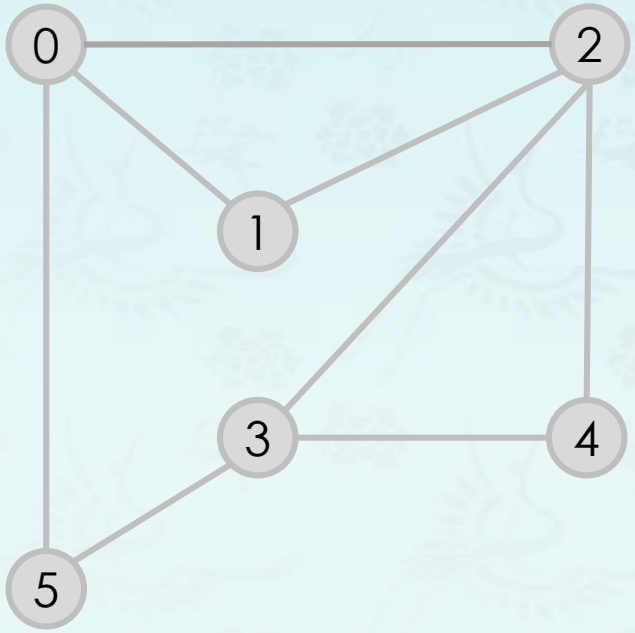

| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

adj[4] 3 | 2

dequeue 4: Check 3 and Check 2

BFS: 0 2 1 5 3

# Breadth-first search demo

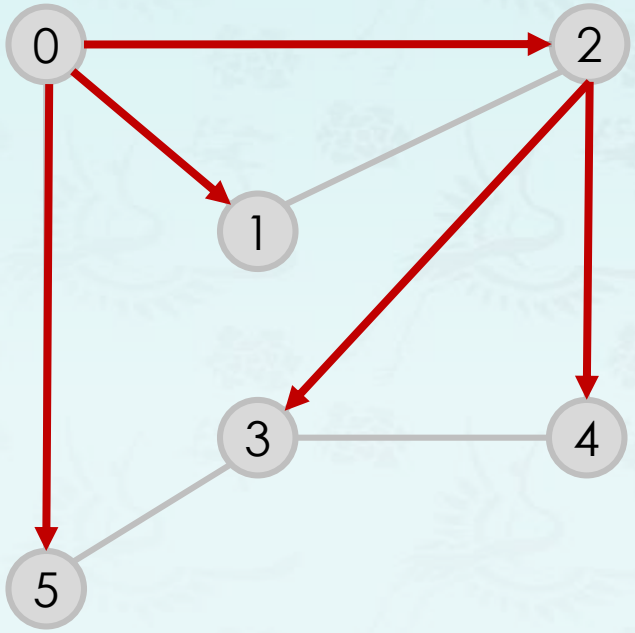



| queue | v | parent[v] | distTo[] |
|---|---|---|---|
| | 0 | - | 0 |
| | 1 | 0 | 1 |
| | 2 | 0 | 1 |
| | 3 | 2 | 2 |
| | 4 | 2 | 2 |
| | 5 | 0 | 1 |

4 done →

BFS: 0 2 1 5 3 4

# Breadth-first search demo




| v | parent[v] | distTo[] |
|---|---|---|
| 0 | - | 0 |
| 1 | 0 | 1 |
| 2 | 0 | 1 |
| 3 | 2 | 2 |
| 4 | 2 | 2 |
| 5 | 0 | 1 |

done

BFS: 0 2 1 5 3 4

## Breadth-first search

**Depth-first search:** Put unvisited vertices on a **stack**.
**Breadth-first search:** Put unvisited vertices on a **queue.**

**Shortest path:** Find path from s to t that uses fewest number of edges.

**BFS:** (from source vertex s)

- Put s onto a FIFO queue, and mark s as visited.
- Repeat until the queue is empty:
  - remove the least recently added vertex v
  - add each of v's unvisited neighbors to the queue,
    and mark them as visited.

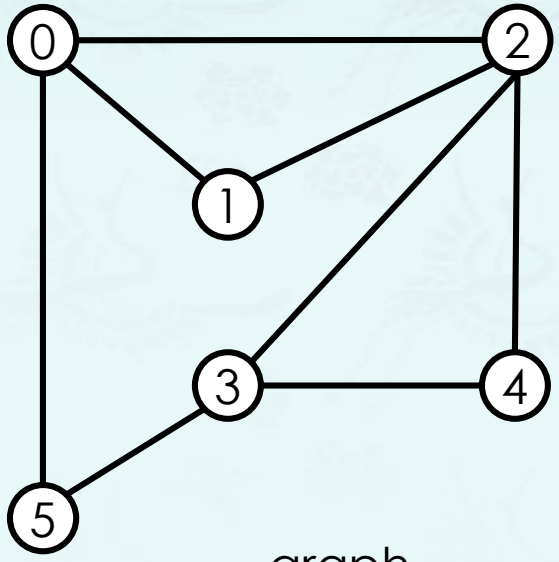
**Intuition:** BFS examines vertices in increasing distance from s.

## Breadth-first search properties

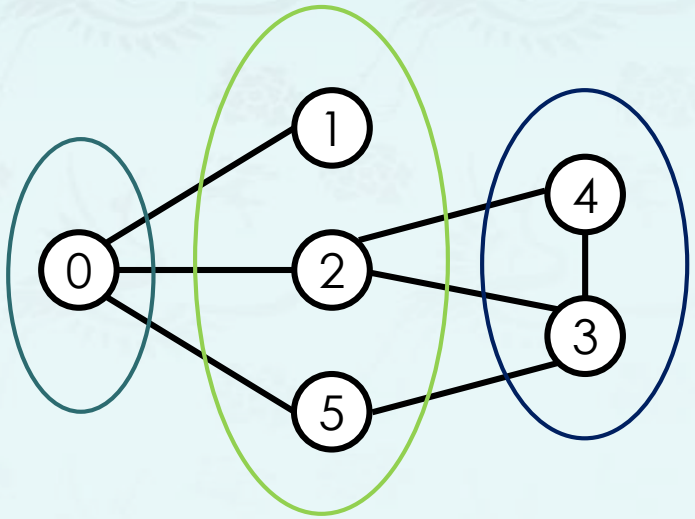
**Proposition:** BFS computes shortest paths (fewest number of edges) from s to all other vertices in a graph in time proportional to E + V.

**Proof: [correctness]** Queue always consists of zero or more vertices of distance k from s, followed by zero or more vertices of distance k + 1.

**Proof: [running time]** Each vertex connected to s is visited once.




graph

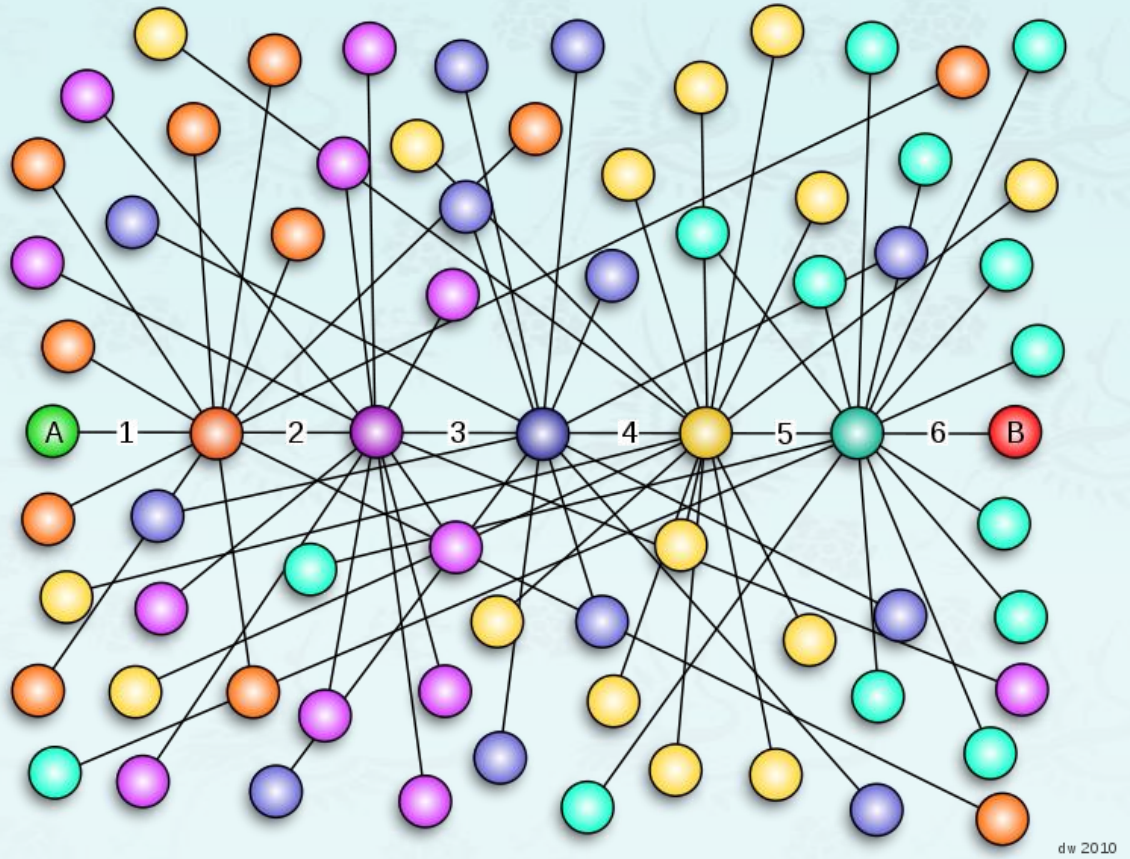



dist = 0 dist = 1 dist = 2

## Breadth-first search implementation in Java

```
// runs BFS at v and produces distTo[] & parentBFS[]
void BFS(graph g, int v) {
  queue<int> que;              // to process each vertex
  queue<int> q;                // BFS result saved
  g->marked[v] = true;
  g->distTo[v] = 0;
  que.push(v);
  q.push(v);
  while (!que.empty()) {
    int cur = que.front(); que.pop();  // remove it since processed
    for (gnode w = g->adj[cur].next; w; w = w->next) {
        if (!g->marked[w->item]) {
          que.push(w->item);   // queued to process next
          q.push(w->item);
          g->distTo[w->item] = g->distTo[cur] + 1;
          g->parentBFS[w->item] = cur;
          g->marked[w->item] = true;
        }
    }
  }
  g->BFSv = q;                 // save the result at v
  setBFSx(g, v, q);            // save the result
}
```

# Breadth-first search application: Kevin Bacon numbers




six degrees of separation?

# Breadth-first search application: Kevin Bacon numbers

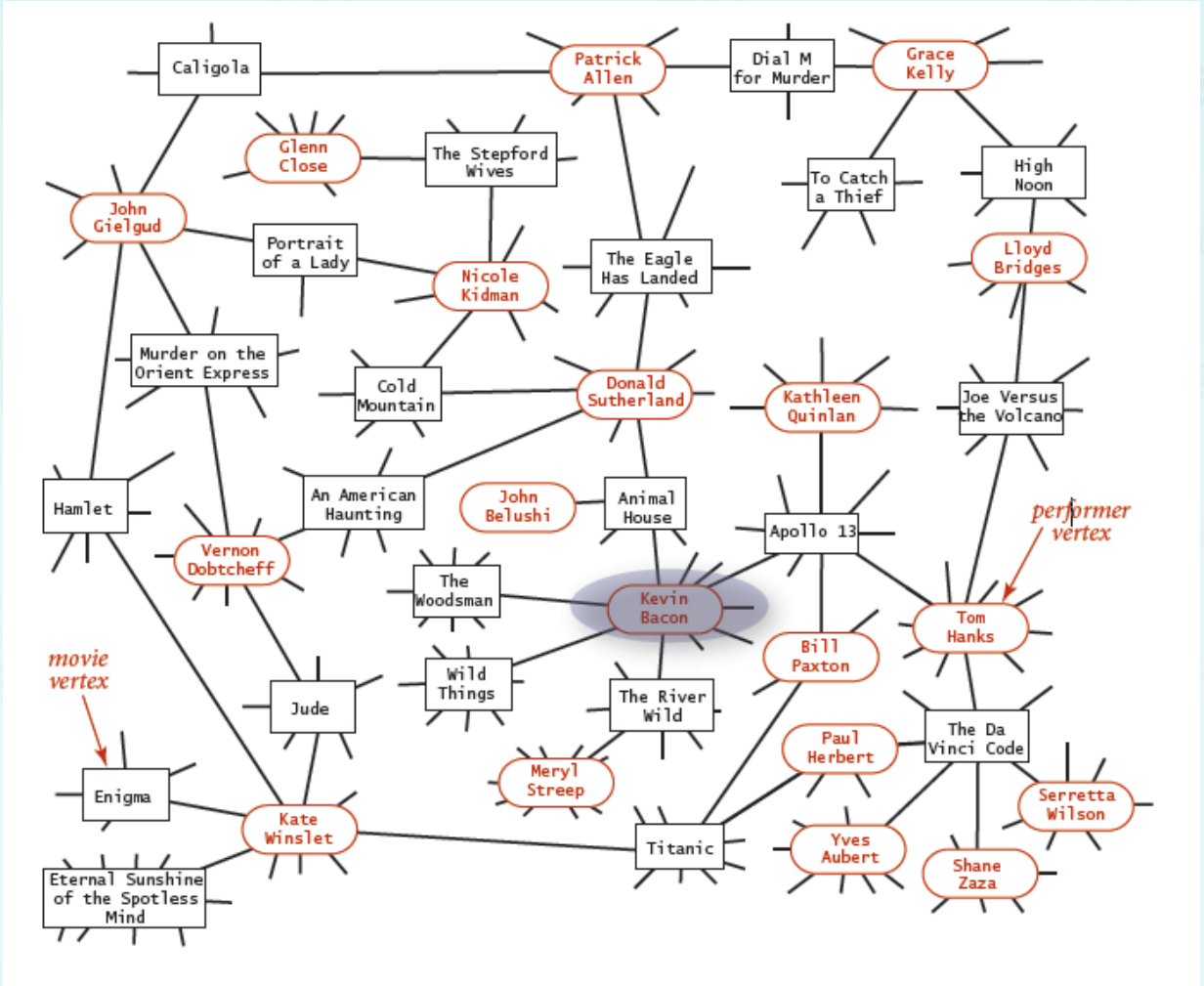

## About the Oracle of Bacon

This is the most comprehensive version of the Kevin Bacon game on the web. The object of the game is to start with any actor or actress who has been in a movie and connect them to Kevin Bacon in the smallest number of **links** possible. Two people are linked if they've been in a movie together. We do not consider links through television shows, made-for-tv movies, writers, producers, directors, etc. For example, you might wonder how Mary Pickford can be connected to Kevin Bacon. One answer is that:

Mary Pickford
was in
Coquette (1929)
with
Louise Beavers
was in
All the Fine Young Cannibals (1960)
with
Robert Wagner (I)
was in
Wild Things (1998)
with
Kevin Bacon

## Breadth-first search properties

- Include one vertex for each performer **and** one for each movie.
- Connect a movie to all performers that appear in that movie.
- Compute shortest path from s = Kevin Bacon.



# Breadth-first search application: Kevin Bacon numbers

Social Facebook

## 4.74 — Facebook Wins By Getting Us Closer Than Six Degrees

*Posted Nov 22, 2011 by Eric Eldon (@eldon)*

35 Like 0 Tweet 327 Share 0 Next Story

data facebook

Facebook users are getting more connected to each other as the service grows and matures, according to a new study by the company's data team and the University of Milan. Instead of the traditional "six degrees of separation" that researchers have historically observed between all people in the world (and Kevin Bacon), the number of degrees has been dropping since 2008 on the site, from 5.28 then to 4.74 now.

ADVERTISEMENT

at&t

Building you a better network.℠

Claim Basis

Building You A Better Network.℠

2008: 5.28 → 2011: 4.74 → 2016.2 :

http://www.bbc.co.uk/newsbeat/article/35500398/how-facebook-updated-six-degrees-of-separation-its-now-357

# Breadth-first search application: Kevin Bacon numbers

Social Facebook

## 4.74 — Facebook Wins By Getting Us Closer Than Six Degrees

*Posted Nov 22, 2011 by Eric Eldon (@eldon)*

35 Like 0 Tweet 327 Share 0 Next Story

data facebook

Facebook users are getting more connected to each other as the service grows and matures, according to a new study by the company's data team and the University of Milan. Instead of the traditional "six degrees of separation" that researchers have historically observed between all people in the world (and Kevin Bacon), the number of degrees has been dropping since 2008 on the site, from 5.28 then to 4.74 now.

ADVERTISEMENT

at&t

Building you a better network.℠

Claim Basis

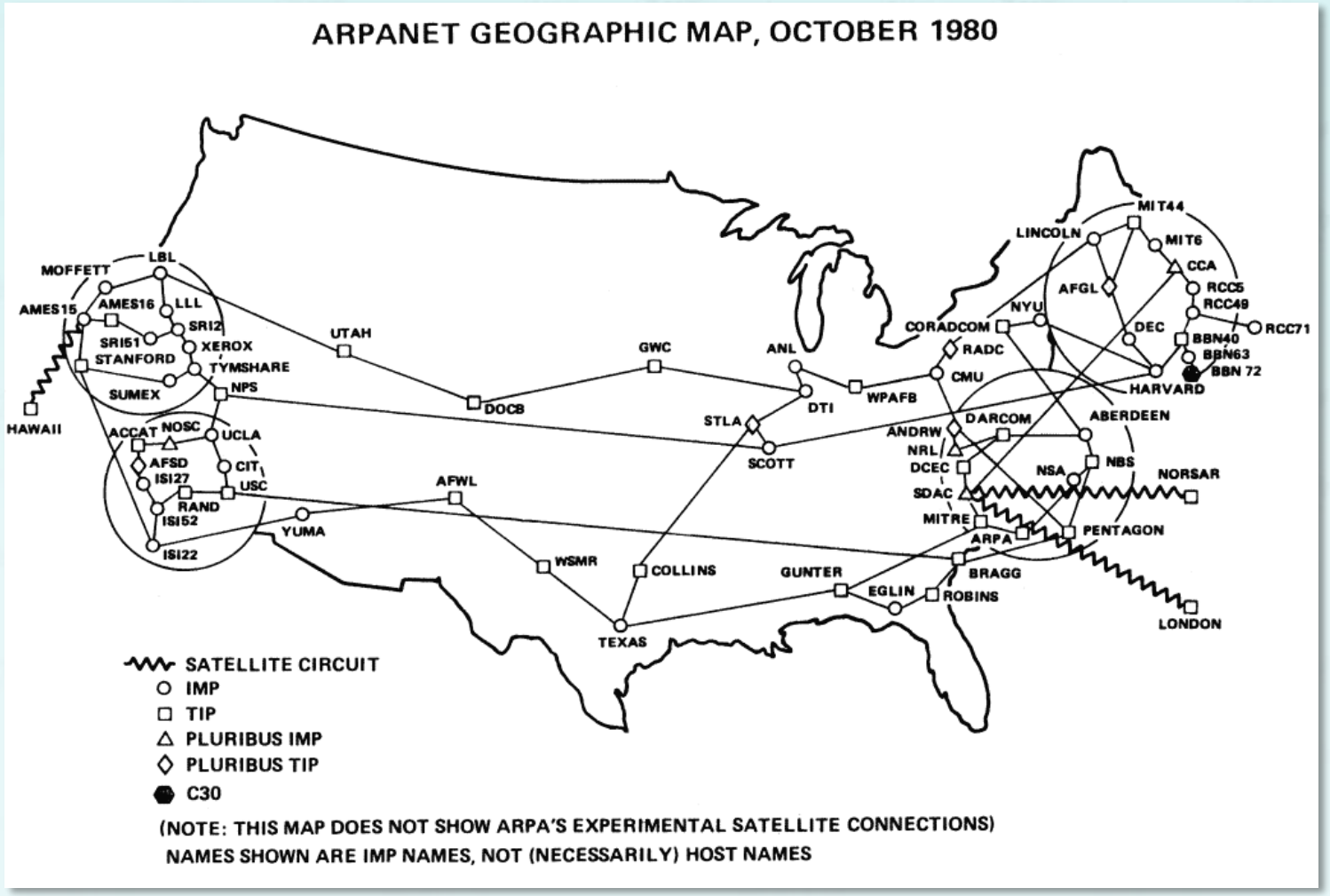
Building You A Better Network.℠

2008: 5.28 → 2011: 4.74 → 2016.2 : 3.57

http://www.bbc.co.uk/newsbeat/article/35500398/how-facebook-updated-six-degrees-of-separation-its-now-357

# Breadth-first search application: routing

Fewest number of hops in a communication network.



# Graph

- Introduction
- Graph API
- Elementary Graph Operations
  - DFS: Depth first search
  - **BFS: Breadth first search**
  - CC: Connected Components

Major references:
1. Fundamentals of Data Structures by Horowitz, Sahni, Anderson-Freed,
2. Algorithms 4th edition - Part 1 & Part 2 by Robert Sedgewick and Kevin Wayne
3. Wikipedia and many resources available from internet