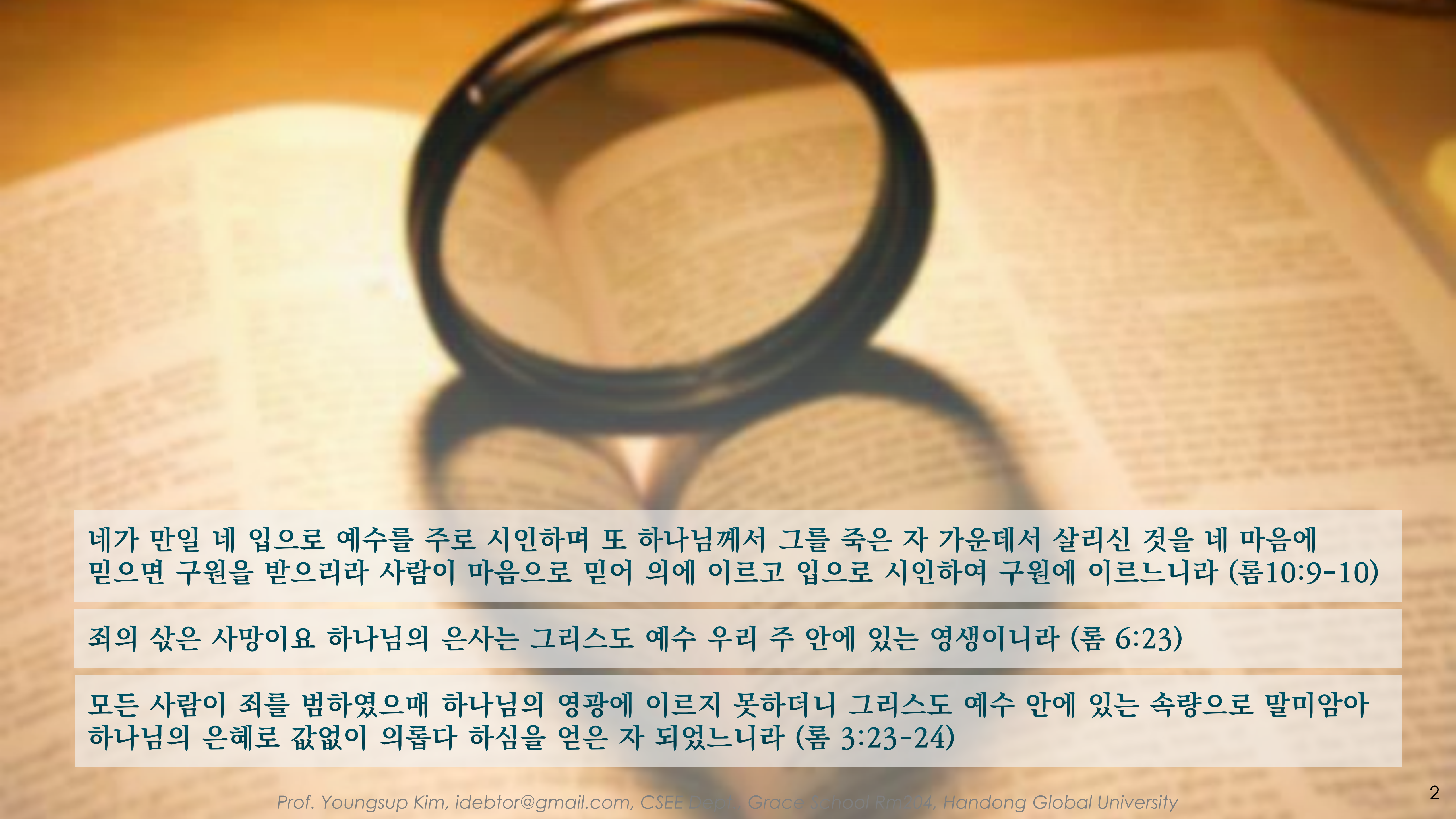
Data Structures
Chapter 7: Graph

1. Introduction
   - Terminology, Representation, ADT
2. Basic Operations
   - DFS, **CC**, **BFS**, Processing
3. Digraph and Applications
4. Minimum Spanning Tree(MST)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University



네가 만일 네 입으로 예수를 주로 시인하며 또 하나님께서 그를 죽은 자 가운데서 살리신 것을 네 마음에 믿으면 구원을 받으리라 사람이 마음으로 믿어 의에 이르고 입으로 시인하여 구원에 이르느니라 (롬10:9-10)

죄의 삯은 사망이요 하나님의 은사는 그리스도 예수 우리 주 안에 있는 영생이니라 (롬 6:23)

모든 사람이 죄를 범하였으매 하나님의 영광에 이르지 못하더니 그리스도 예수 안에 있는 속량으로 말미암아 하나님의 은혜로 값없이 의롭다 하심을 얻은 자 되었느니라 (롬 3:23-24)

Connectivity Queries

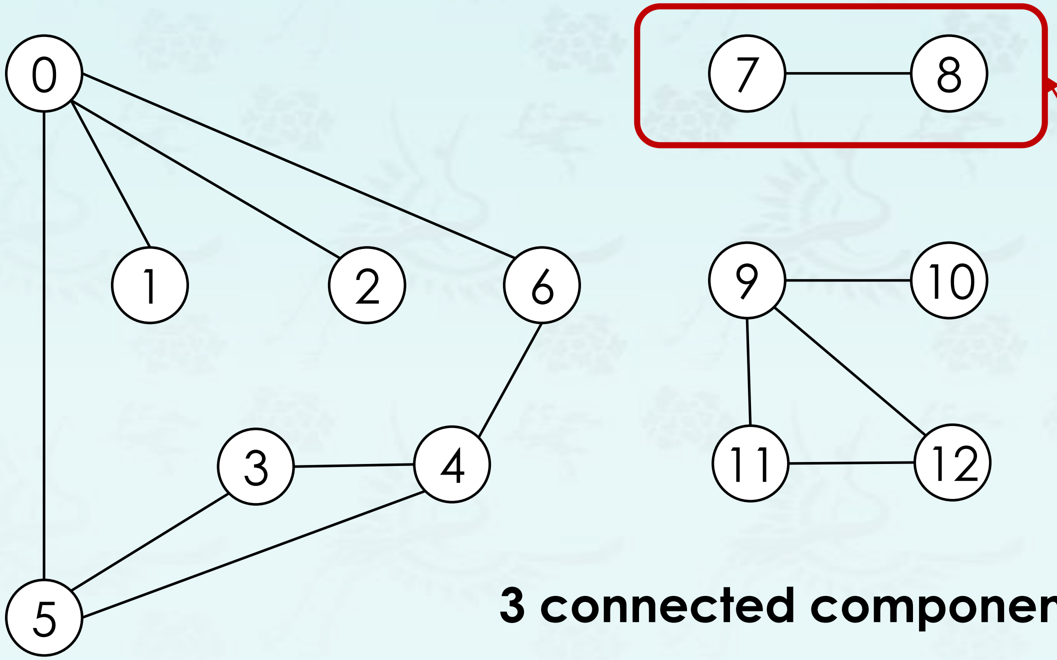
- **Def.:** Vertices v and w are connected if there is a path between them.
- **Goal:** Preprocess graph to answer queries of the form "*is v connected to w?*" in constant time.

	Connected Component	
	CC(Graph g)	*find connected component in g*
bool	connected(int v, int w)	*are v and w connected"*
int	count()	*member of connected components*
int	id(int v)	*component identifier for v*

Depth-first search? Yes …

Connected Components

The relation “is connected to” is equivalence relation:
Reflexive: v is connected to v.
Symmetric: if v is connected to w, then w is connected v.
Transitive: if v connected to w and w connected to x, then v connected to x



3 connected components

v	id[v]
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	2
8	2
9	3
10	3
11	3
12	3

Remark:
Given connected components, can answer queries in constant time.

Connected Components

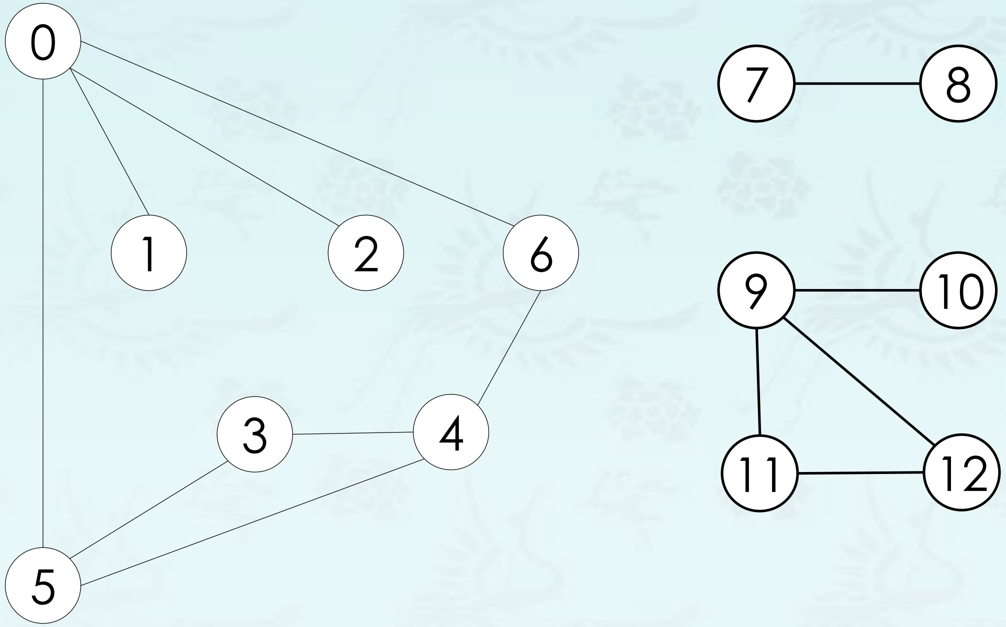
Goal: Partition vertices into connected components.

- Initialize all vertices v as unmarked.
- For each unmarked vertex v, run DFS to identify all vertices discovered as part of the same component.

Connected Components

To visit a vertex v:

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



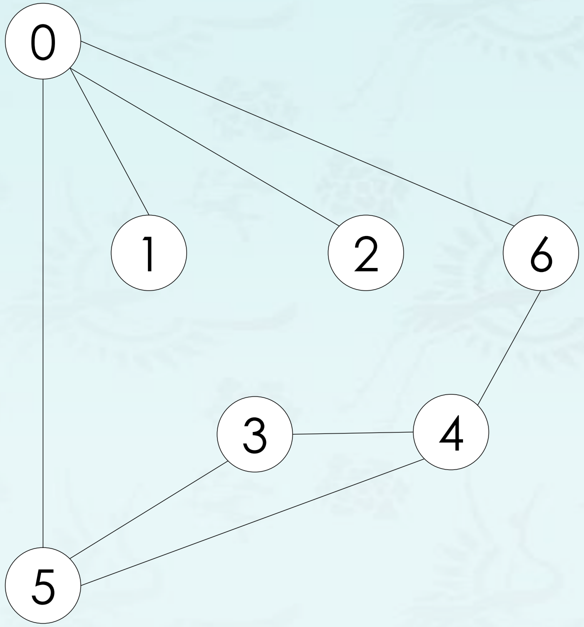
Graph g:

V-E lists

```
graph6.txt
13  ←  V
13  ←  E
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

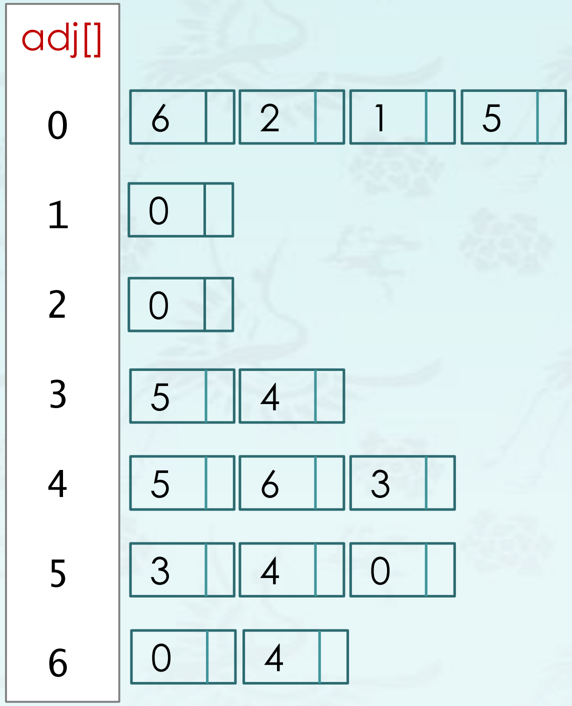
Challenge: build adjacency lists?

Connected Components



Graph g:

Adjacency lists



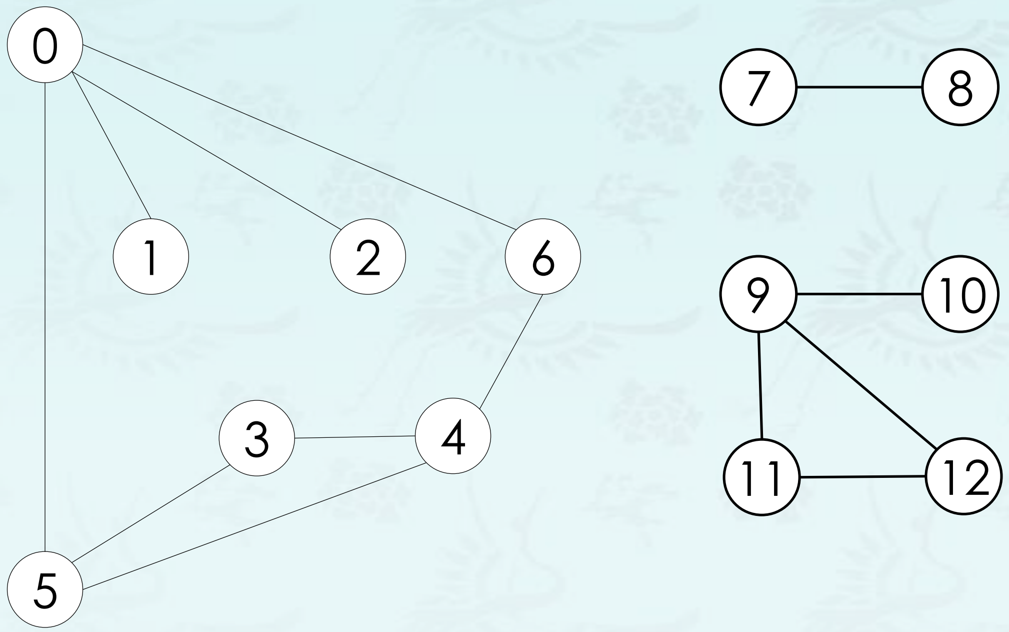
V-E lists

```
graph6.txt
13
13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

V

E

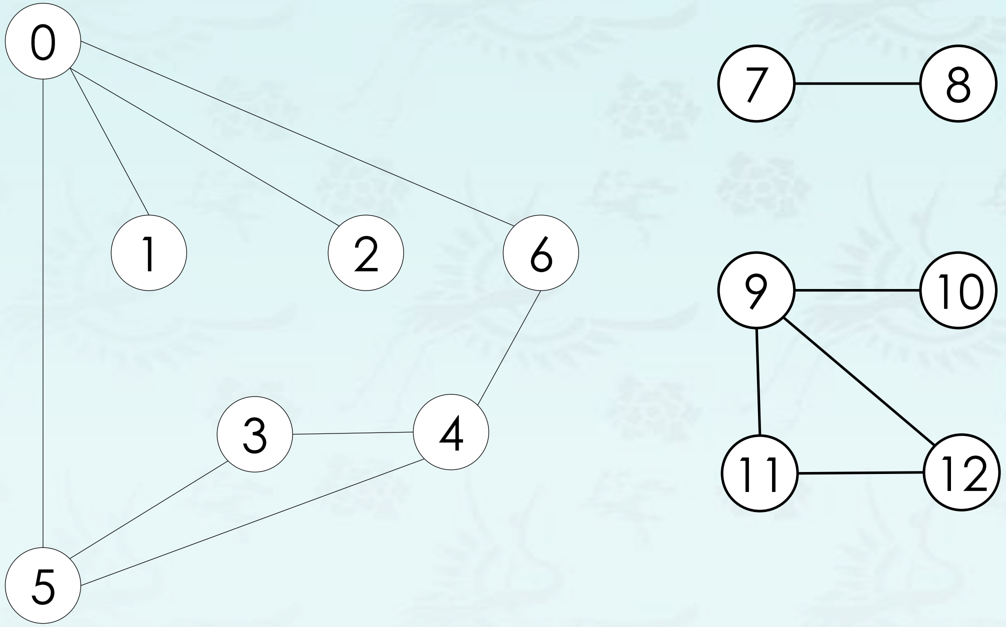
Connected Components



Graph g:

v	marked[]	id[]
0	F	–
1	F	–
2	F	–
3	F	–
4	F	–
5	F	–
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

Connected Components



v	marked[]	id[]
0	T	0
1	T	0
2	T	0
3	T	0
4	T	0
5	T	0
6	T	0
7	T	1
8	T	1
9	T	2
10	T	2
11	T	2
12	T	2

Graph g:

Connected Components – Coding

```
// returns true if v and w are connected.
bool connected(graph g, int v, int w) {
  if (empty(g)) return true;

  DFS_CCs(g);

  return g->CCID[v] == g->CCID[w];
}
```

```
// returns number of connected components.
int nCCs(graph g) {
  int id = g->CCID[0];
  int count = 1;
  for (int i = 0; i < V(g); i++)
    if (id != g->CCID[i]) {
      id = g->CCID[i];
      count++;
    }
  return id == 0 ? 0 : count;
}
```

Data Structures
Chapter 7: Graph

1. Introduction
   - Terminology, Representation, ADT
2. Basic Operations
   - DFS, CC, **BFS**, Processing
3. Digraph and Applications
4. Minimum Spanning Tree(MST)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, 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)

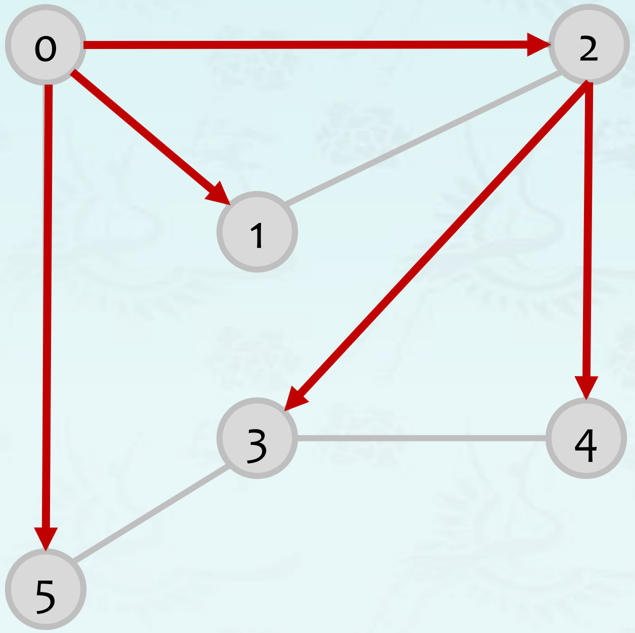
- Put s onto a FIFO queue, and mark s as visited.
- Repeat until the queue is empty:
  - remove the least recently added vertex v
  - add each of v's unvisited neighbors to the queue,
    and mark them as visited.

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.



graph2.txt

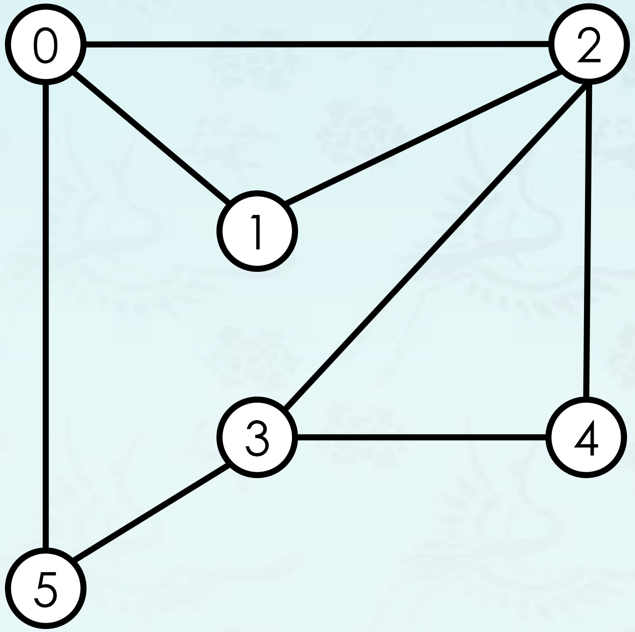
v	parent[v]	distTo[]
0	–	0
1	0	1
2	0	1
3	2	2
4	2	2
5	0	1

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.



Graph g:

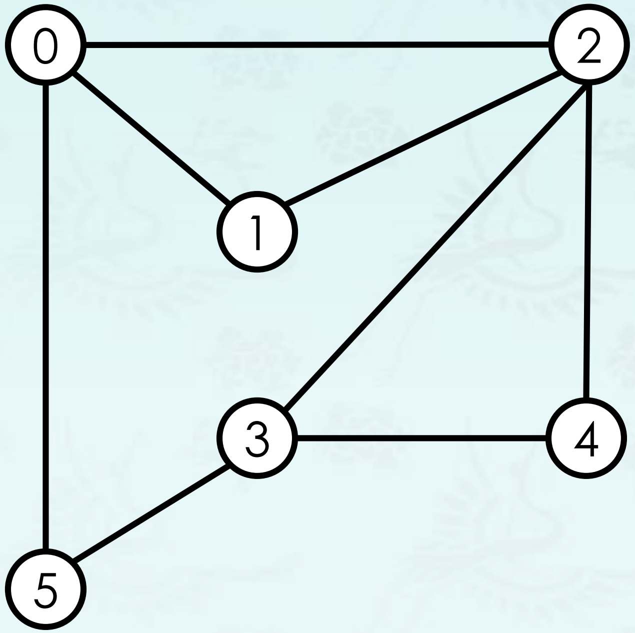
Adjacency lists

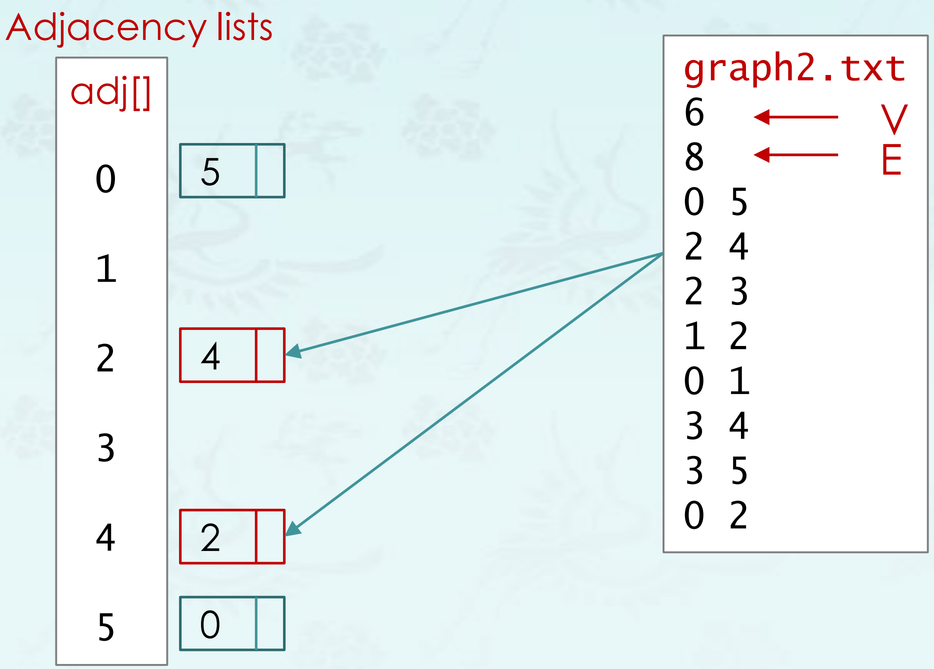

```
graph2.txt
6  ← V
8  ← E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

Challenge: build adjacency lists?

Repeat until queue is empty:

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



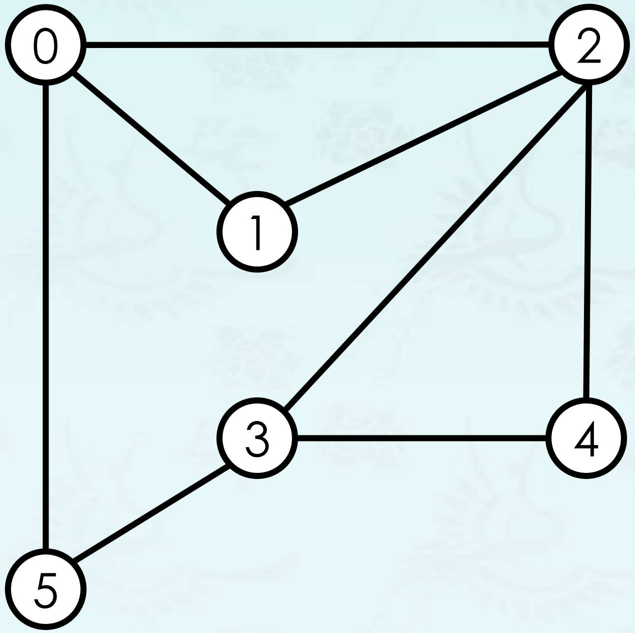


Graph g:

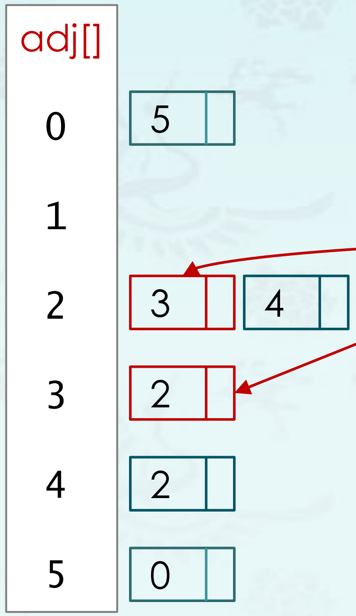
Challenge: build adjacency lists?

Repeat until queue is empty:

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



Adjacency lists



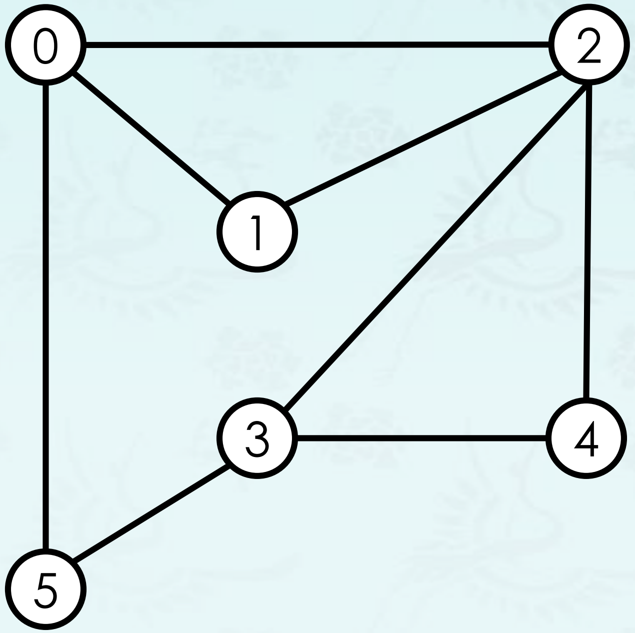

```
graph2.txt
6   V
8   E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

Graph g:

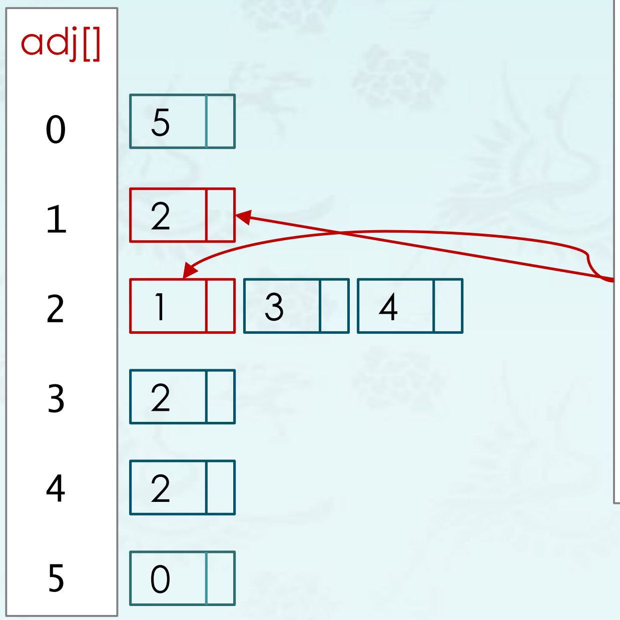
Challenge: build adjacency lists?

Repeat until queue is empty:

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



Adjacency lists



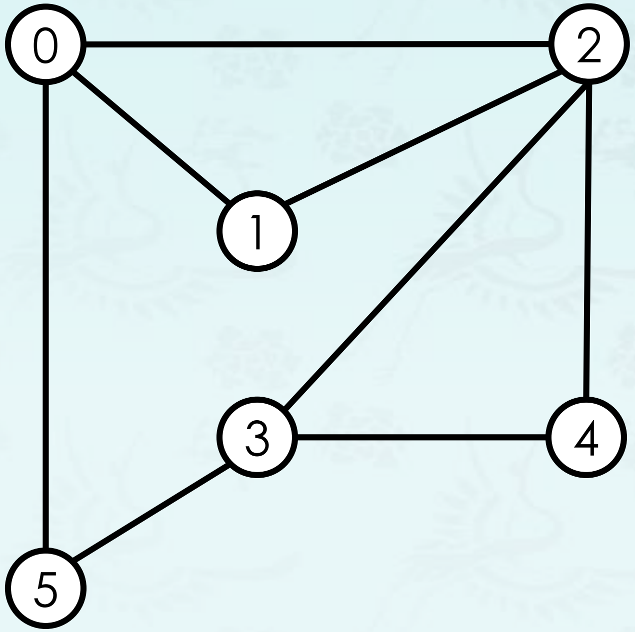

```
graph2.txt
6   V
8   E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

Graph g:

Challenge: build adjacency lists?

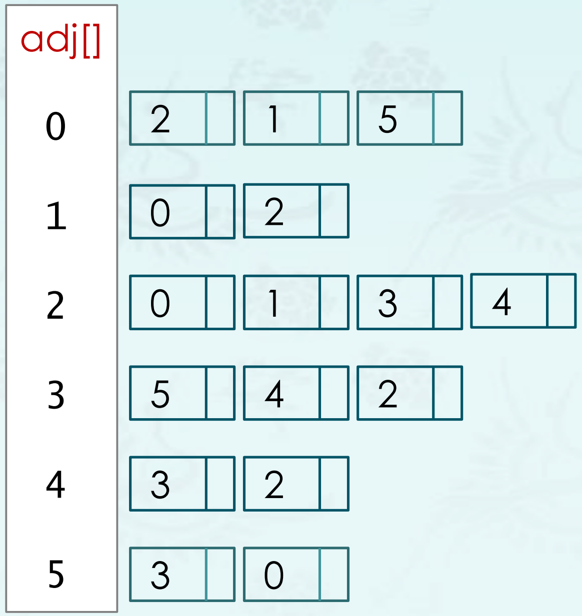
Repeat until queue is empty:

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



Graph g:

Adjacency lists

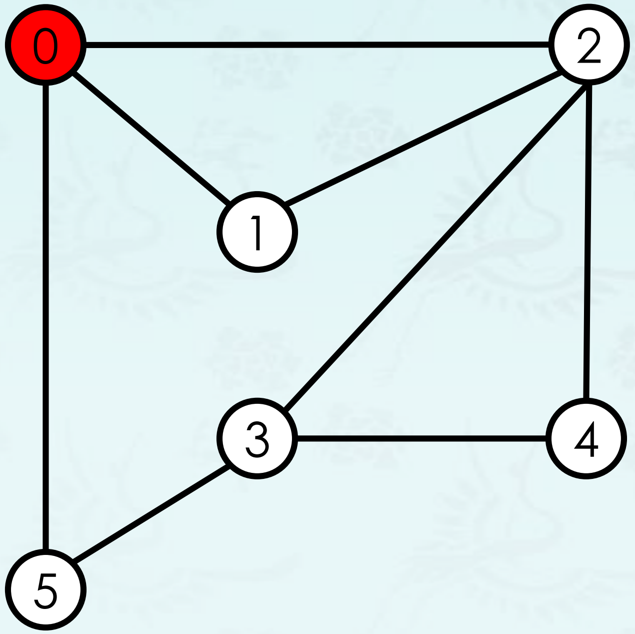



```
graph2.txt
6   ← V
8   ← E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

Challenge: build adjacency lists?
Job done

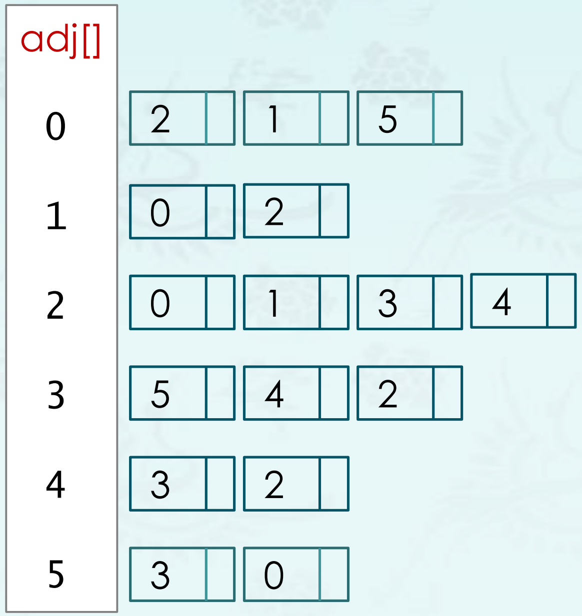
Repeat until queue is empty:

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



Graph g:

Adjacency lists

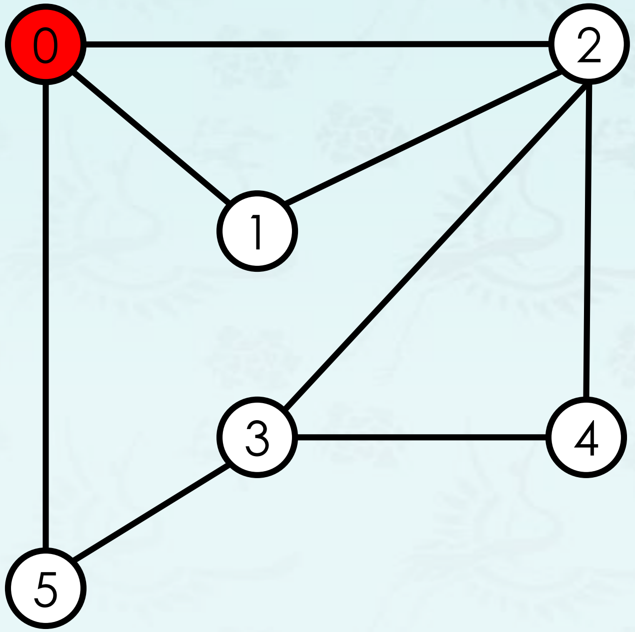



```
graph2.txt
6  ← V
8  ← E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

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.



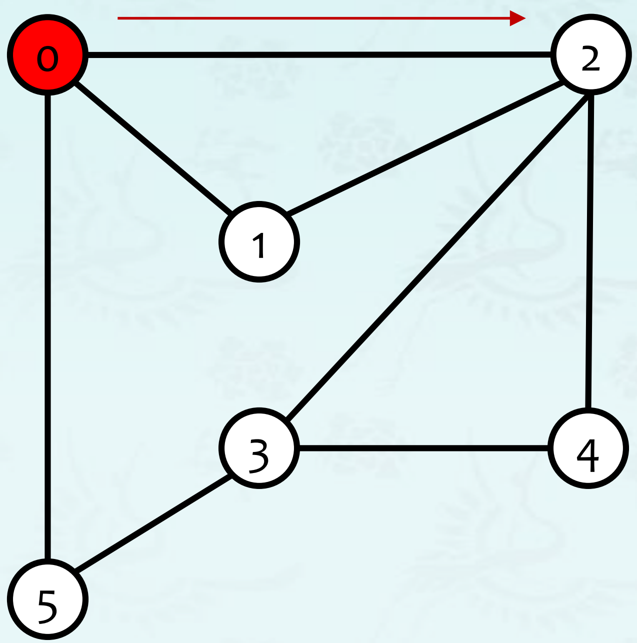
queue	v	parent[v]	distTo[]
	0	–	0
	1	–	–
	2	–	–
	3	–	–
	4	–	–
0	5	–	–

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	–	–

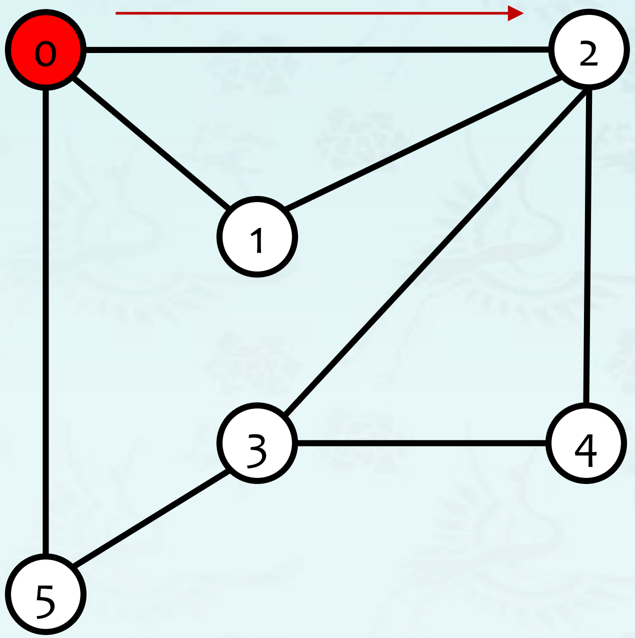
adj[0] | 2 | 1 | 5 |

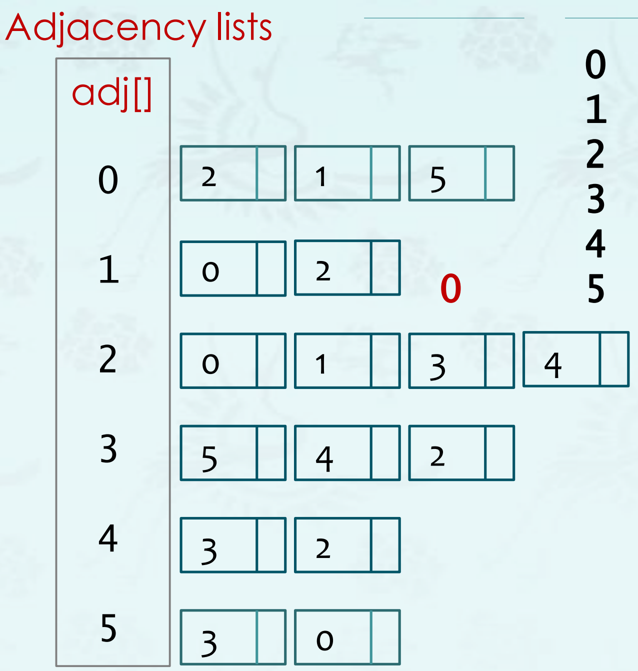
dequeue 0: check2, check 1 and check 5

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.



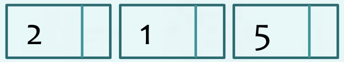


queue

0

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

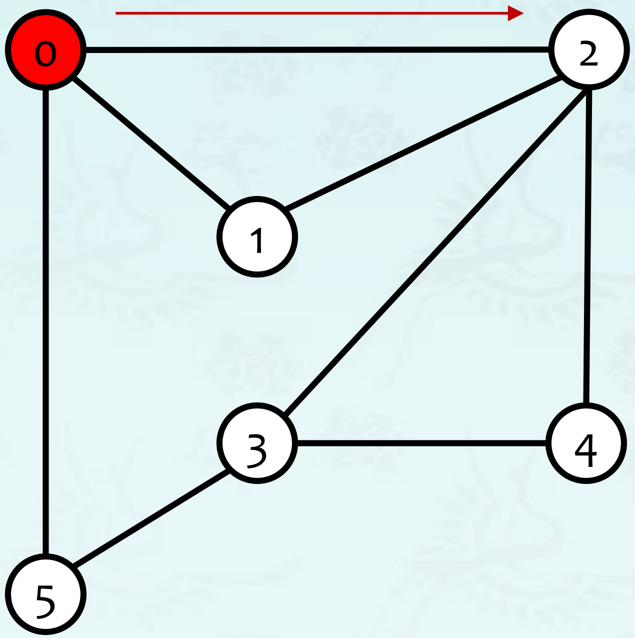
adj[0]



dequeue 0: check2, check 1 and check 5

Repeat until queue is empty:

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



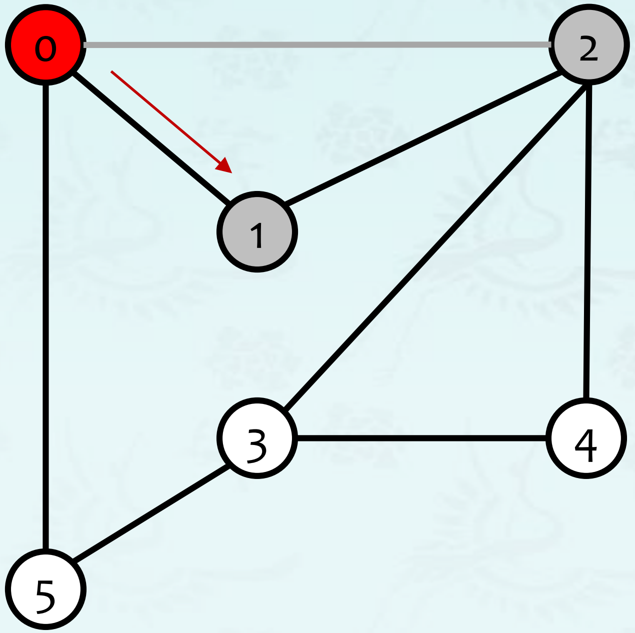
queue	v	parent[v]	distTo[]
	0	–	0
	1	–	–
	2	0	1
	3	–	–
	4	–	–
2	5	–	–

adj[0] | 2 | 1 | 5 |

dequeue 0: check2, check 1 and check 5

Repeat until queue is empty:

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



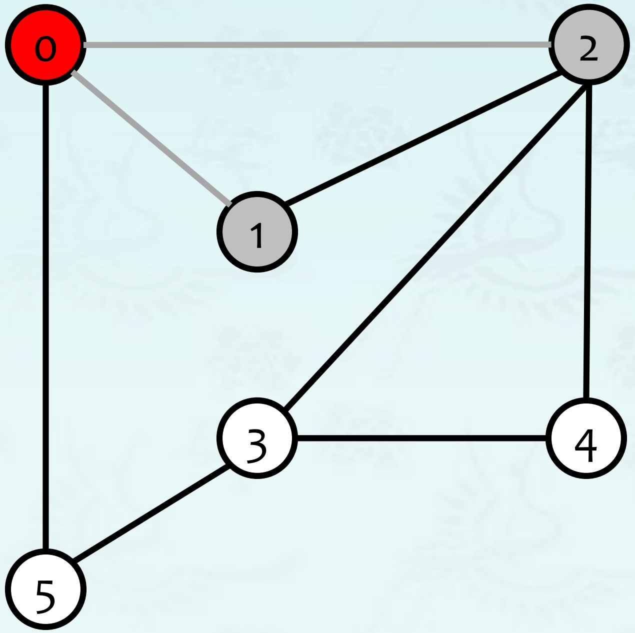
queue	v	parent[v]	distTo[]
	0	–	0
	1	–	–
	2	0	1
	3	–	–
	4	–	–
2	5	–	–

adj[0] | 2 | | 1 | | 5 | |

dequeue 0: check2, check 1 and check 5

Repeat until queue is empty:

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



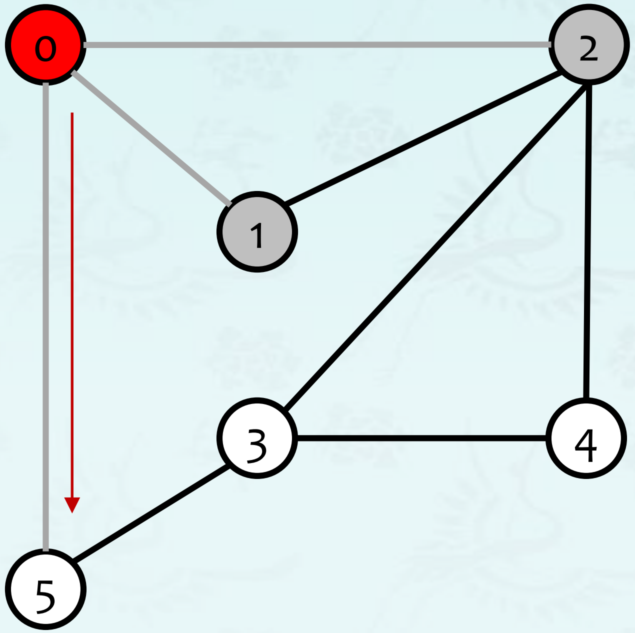
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
	3	–	–
1	4	–	–
2	5	–	–

adj[0] | 2 | | 1 | | 5 | |

dequeue 0: check2, check 1 and check 5

Repeat until queue is empty:

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

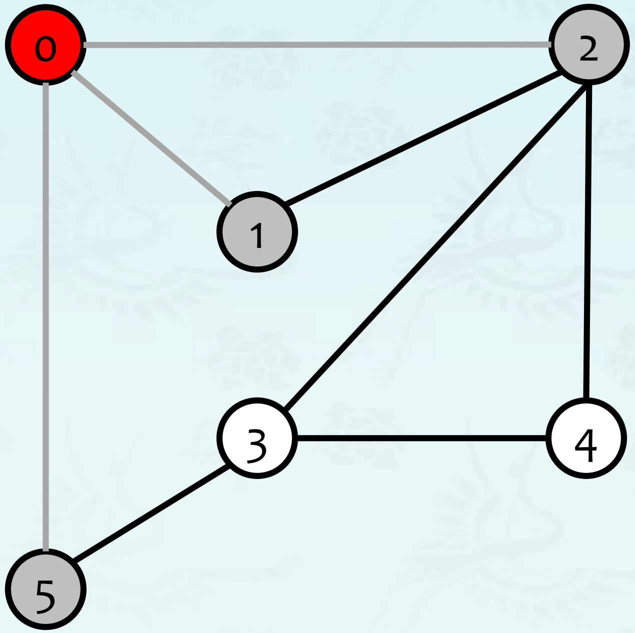


queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
	3	–	–
1	4	–	–
2	5	–	–

adj[0] | 2 | | 1 | | 5 |

dequeue 0: check2, check 1 and check 5

Breadth-first search demo

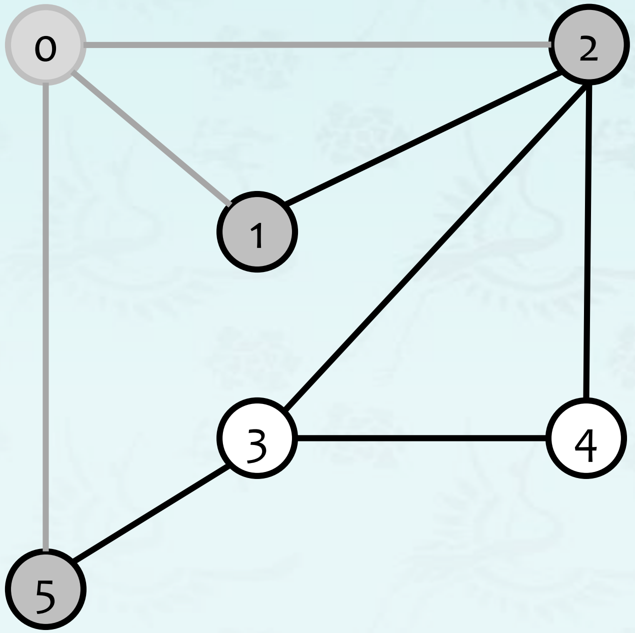


queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
5	3	–	–
1	4	–	–
2	5	0	1

adj[0] 2 1 5

dequeue 0: check2, check 1 and check 5

Breadth-first search demo



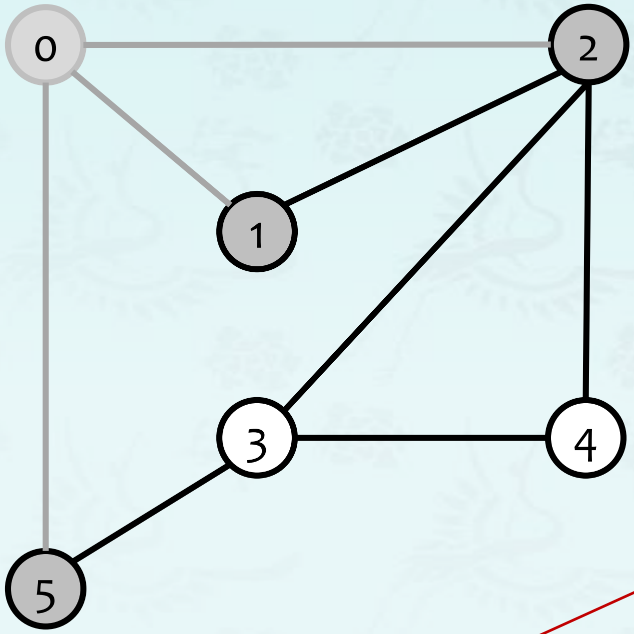
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
5	3	-	-
1	4	-	-
2	5	0	1

adj[0] | 2 | | 1 | | 5 |

0 done

BFS: 0

Breadth-first search demo

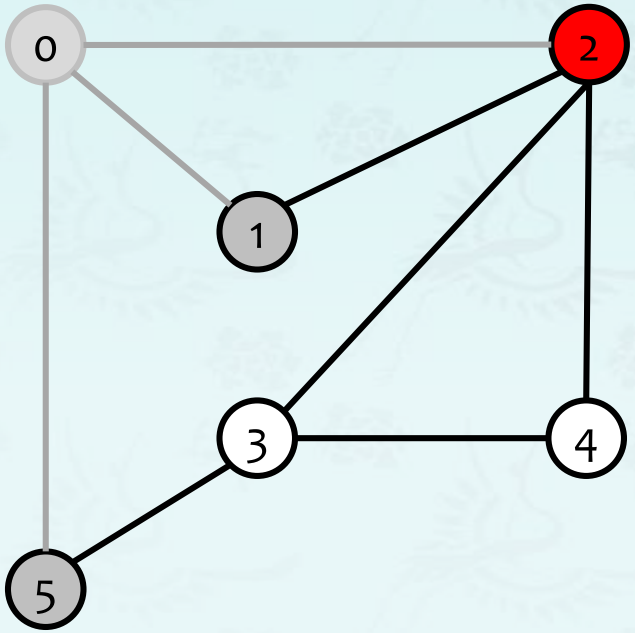


queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
5	3	–	–
1	4	–	–
2	5	0	1

dequeue 2:

BFS: 0

Breadth-first search demo



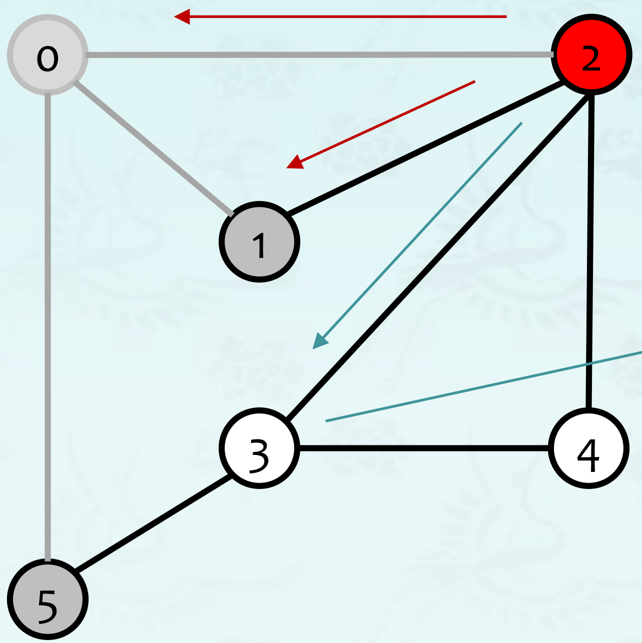
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
5	3	-	-
1	4	-	-
	5	0	1

adj[2] 0 | 1 | 3 | 4

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

BFS: 0

Breadth-first search demo



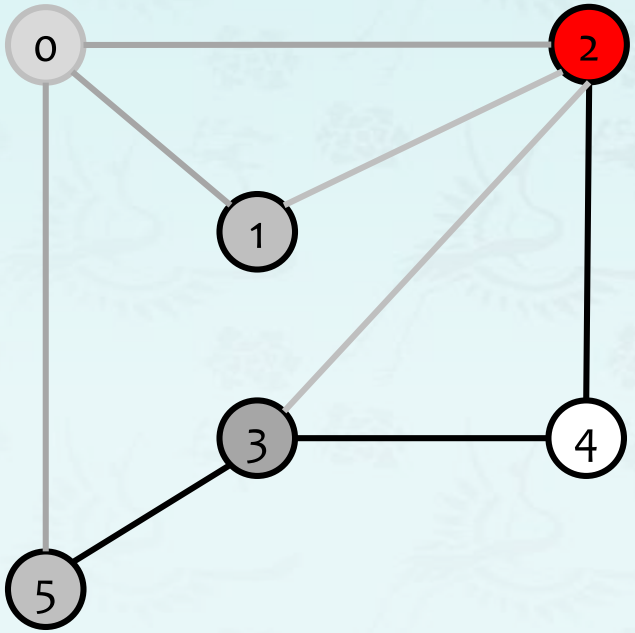
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
5	3	-	-
1	4	-	-
	5	0	1

adj[2] | 0 | 1 | 3 | 4 |

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

BFS: 0

Breadth-first search demo



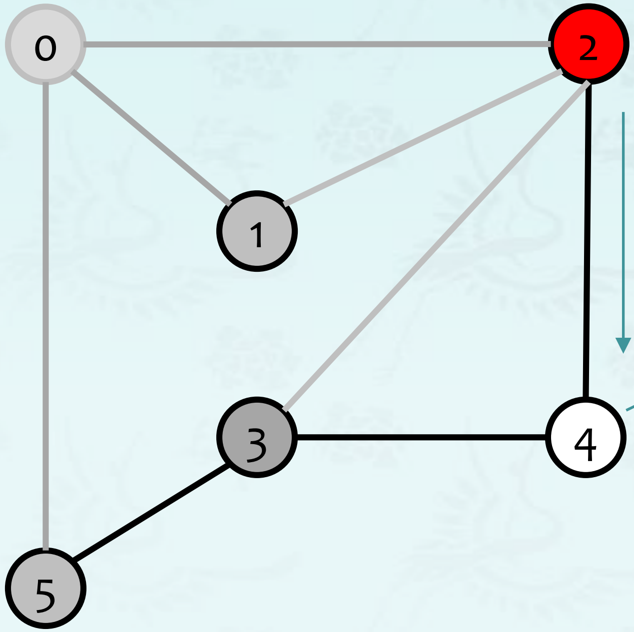
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
3	2	0	1
5	3	2	2
1	4	–	–
	5	0	1

adj[2] 0 | 1 | 3 | 4

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

BFS: 0

Breadth-first search demo



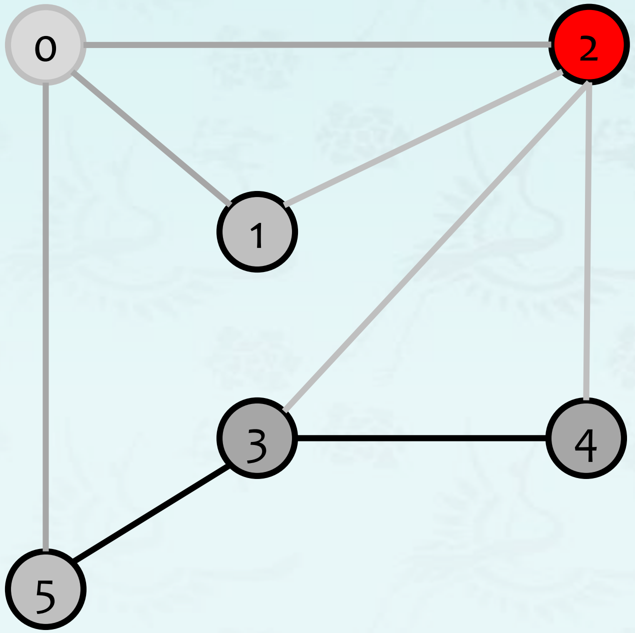
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
3	2	0	1
5	3	2	2
1	4	–	–
	5	0	1

adj[2] 0 | 1 | 3 | 4

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

BFS: 0

Breadth-first search demo



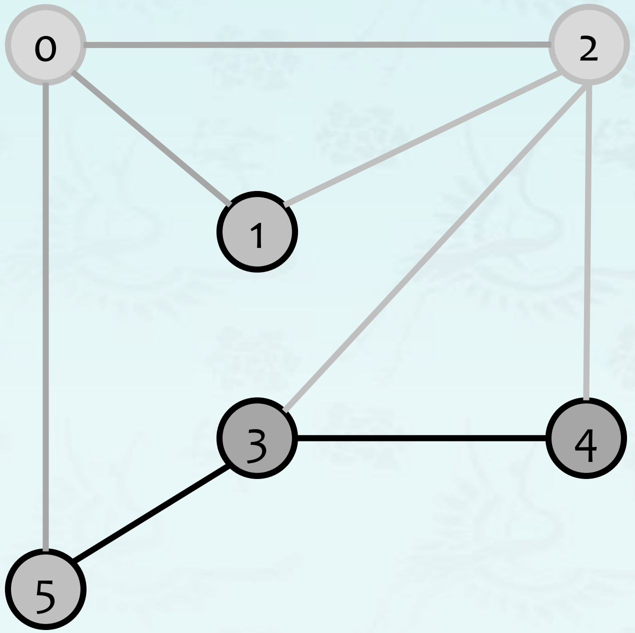
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
1	4	–	–
	5	0	1

adj[2] 0 → 1 → 3 → 4

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

BFS: 0

Breadth-first search demo

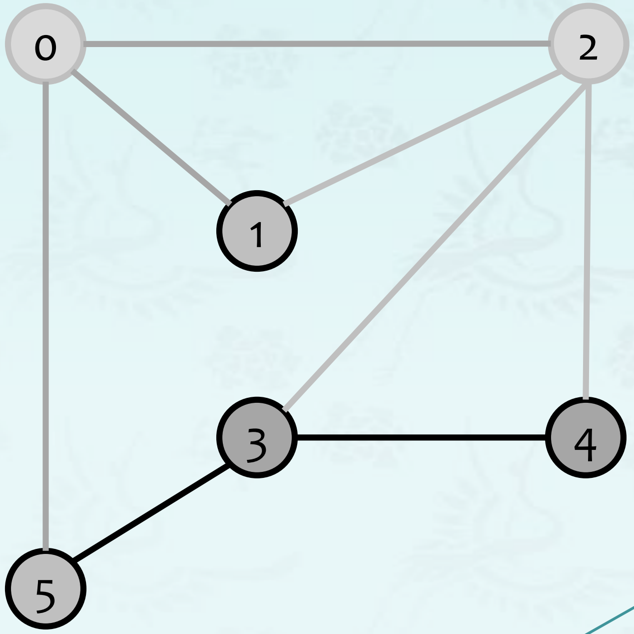


queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
1	4	2	2
	5	0	1

adj[2] | 0 | | 1 | | 3 | | 4 |

2 done → BFS: 0 2

Breadth-first search demo

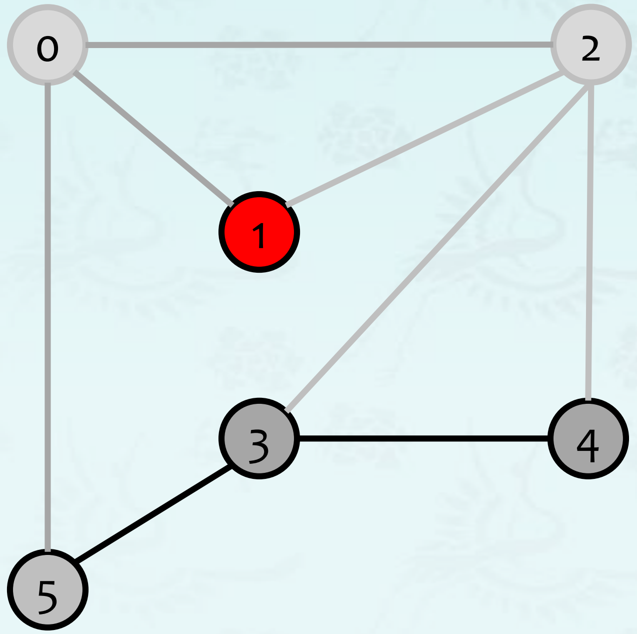


queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
1	4	2	2
	5	0	1

dequeue 1

BFS: 0 2

Breadth-first search demo

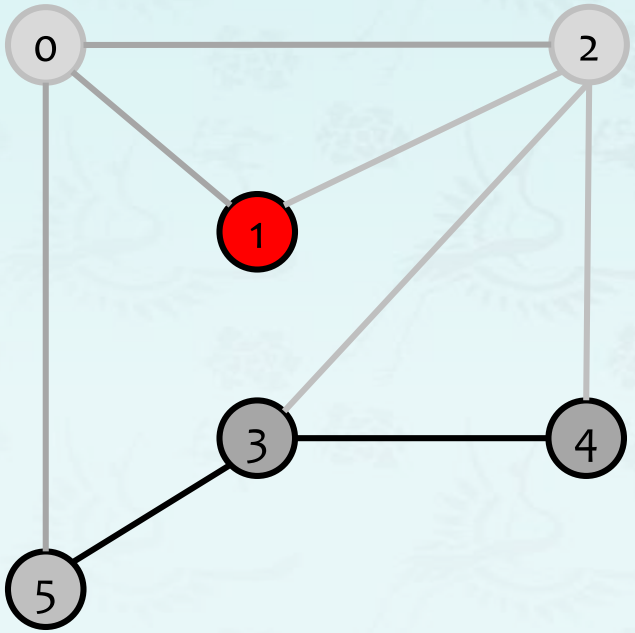


queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
	4	2	2
	5	0	1

dequeue 1

BFS: 0 2

Breadth-first search demo



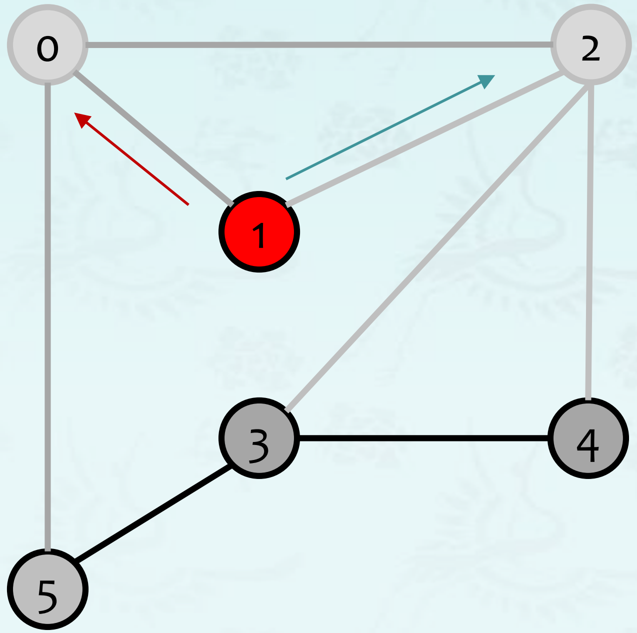
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
	4	2	2
	5	0	1

adj[1] | 0 | | 2 | |

dequeue 1: check 0, and check 2

BFS: 0 2

Breadth-first search demo



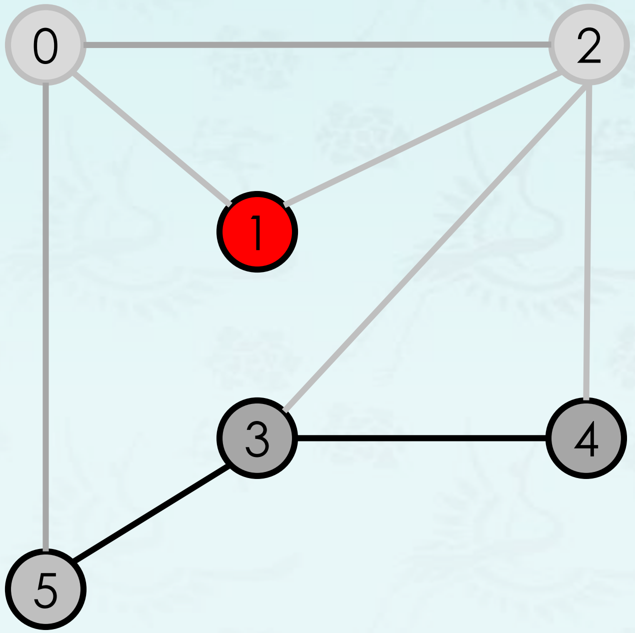
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
	4	2	2
	5	0	1

adj[1] | 0 | | 2 | |

dequeue 1: check 0, and check 2

BFS: 0 2

Breadth-first search demo



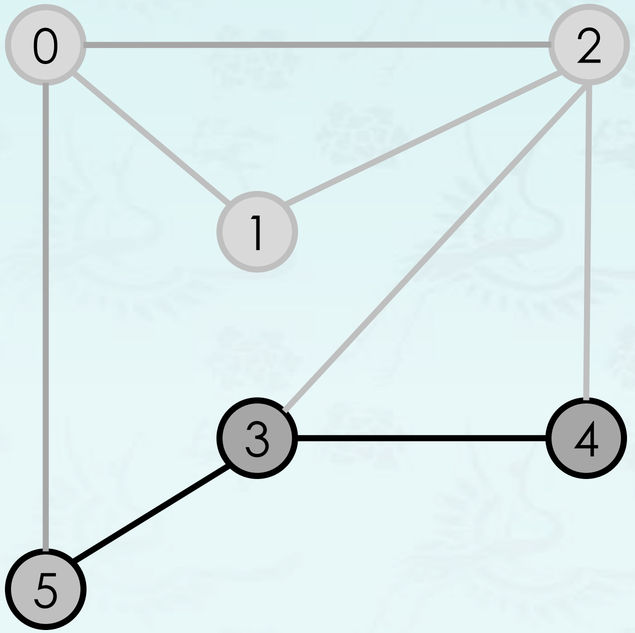
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
	4	2	2
	5	0	1

adj[1] 0 | 2

dequeue 1: check 0, and check 2

BFS: 0 2

Breadth-first search demo



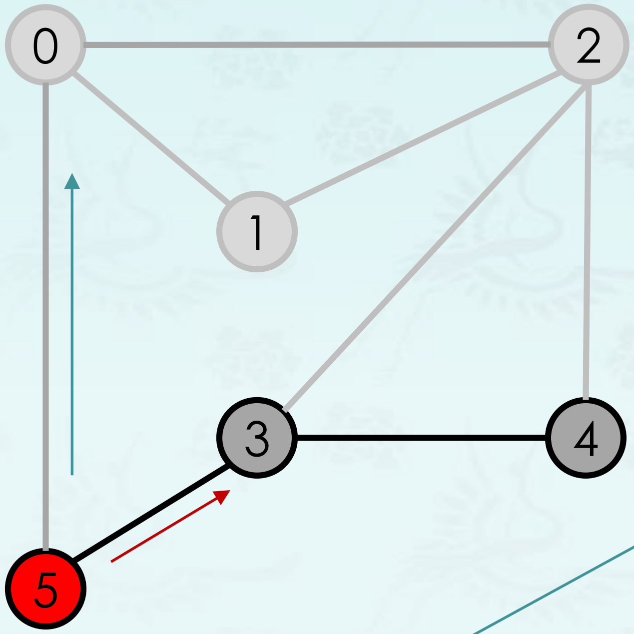
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
	4	2	2
	5	0	1

adj[1] 0 | 2

1 done

BFS: 0 2 1

Breadth-first search demo



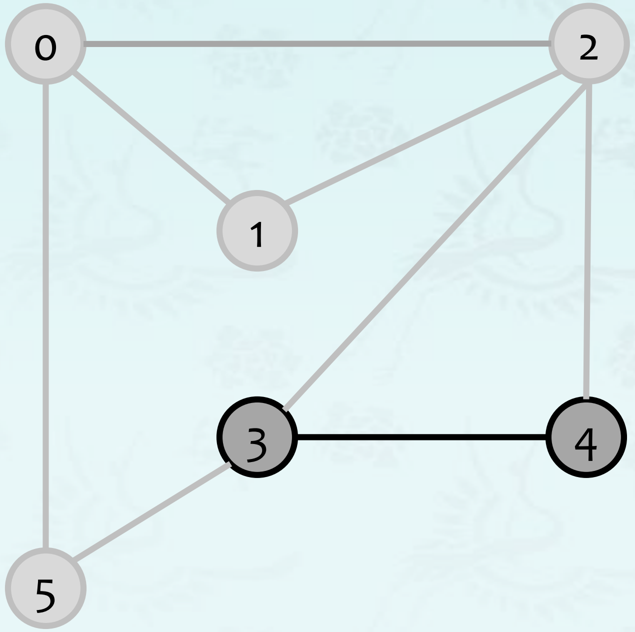
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
5	3	2	2
	4	2	2
	5	0	1

adj[5] 3 | 0

dequeue 5: check 3 and check 0

BFS: 0 2 1

Breadth-first search demo



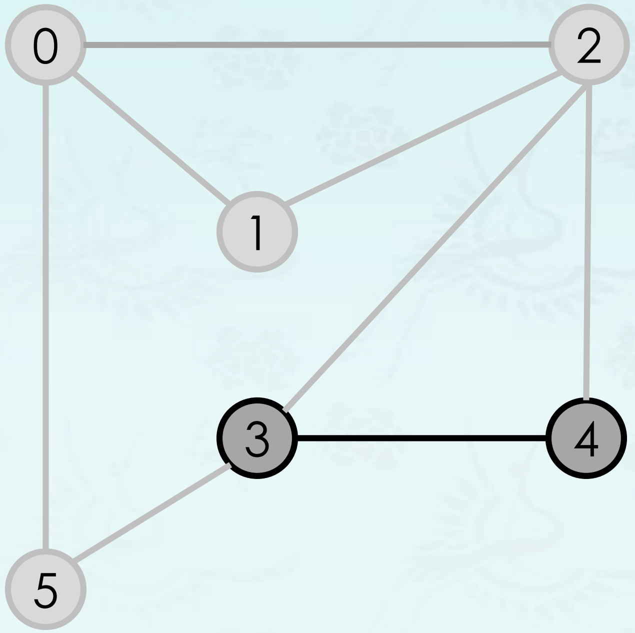
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
	3	2	2
	4	2	2
	5	0	1

adj[5] 3 0

5 done

BFS: 0 2 1 5

Breadth-first search demo



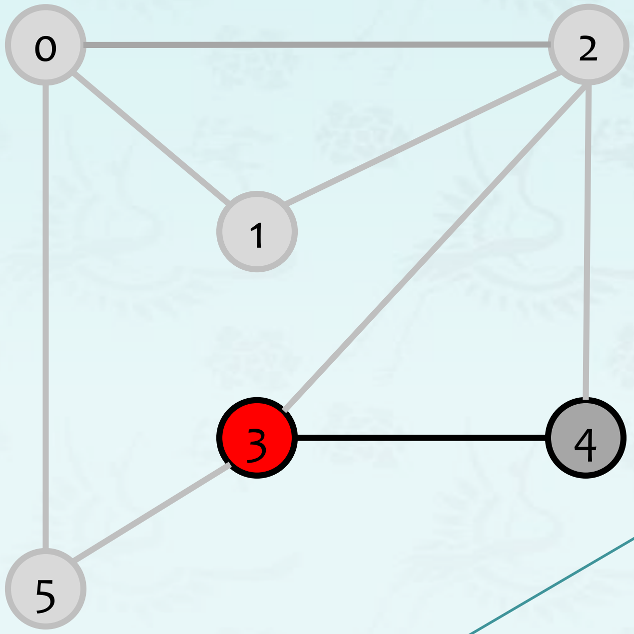
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
	3	2	2
	4	2	2
	5	0	1

adj[3] | 5 | | 4 | | 2 | |

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

BFS: 0 2 1 5

Breadth-first search demo

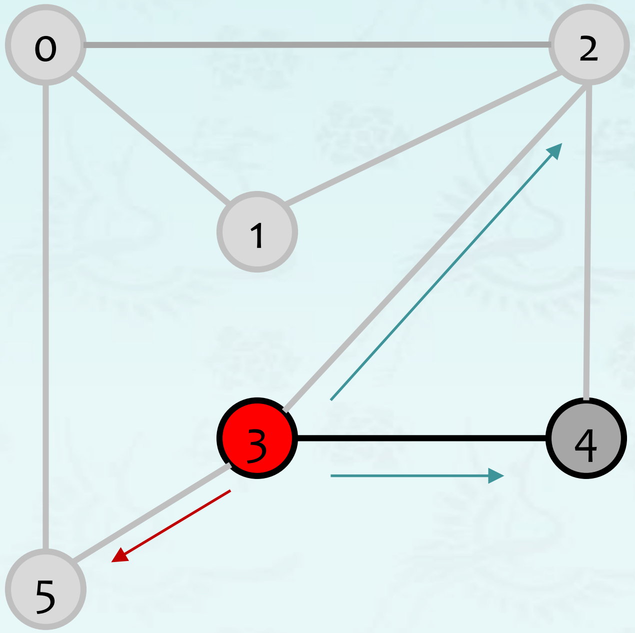


queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
3	2	0	1
	3	2	2
	4	2	2
	5	0	1

dequeue 3:

BFS: 0 2 1 5

Breadth-first search demo



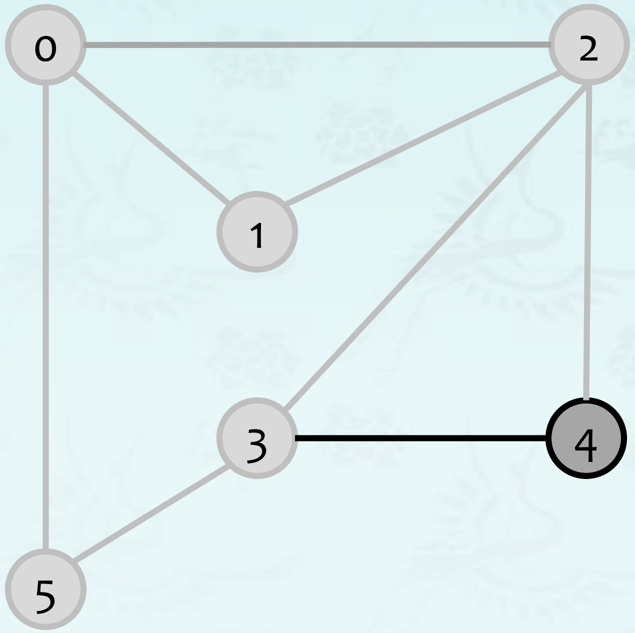
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
	2	0	1
	3	2	2
	4	2	2
	5	0	1

adj[3] | 5 | 4 | 2 |

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

BFS: 0 2 1 5

Breadth-first search demo



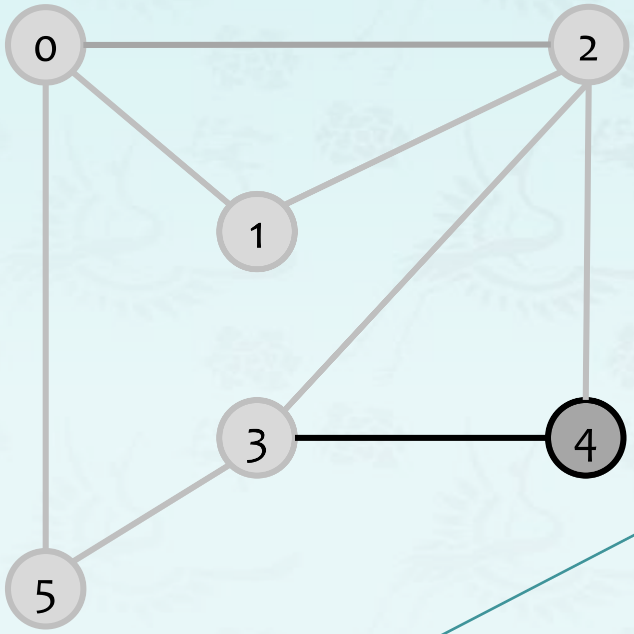
queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
	2	0	1
	3	2	2
	4	2	2
	5	0	1

adj[3] 5 → 4 → 2

3 done

BFS: 0 2 1 5 3

Breadth-first search demo

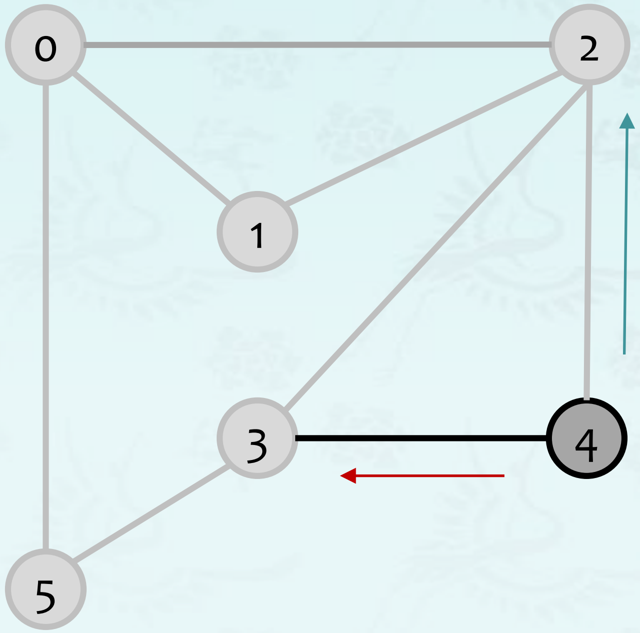


queue	v	parent[v]	distTo[]
	0	–	0
4	1	0	1
	2	0	1
	3	2	2
	4	2	2
	5	0	1

dequeue 4

BFS: 0 2 1 5 3

Breadth-first search demo



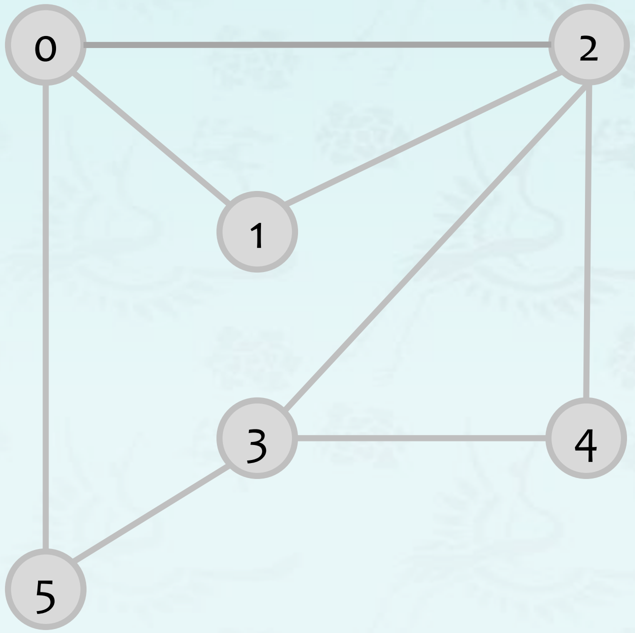
queue	v	parent[v]	distTo[]
	0	–	0
	1	0	1
	2	0	1
	3	2	2
	4	2	2
	5	0	1

adj[4] 3 2

dequeue 4: Check 3 and Check 2

BFS: 0 2 1 5 3

Breadth-first search demo



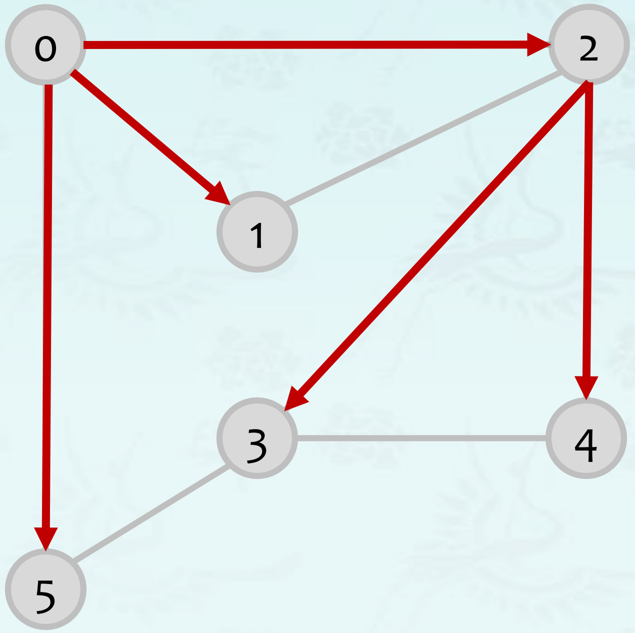
queue

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.

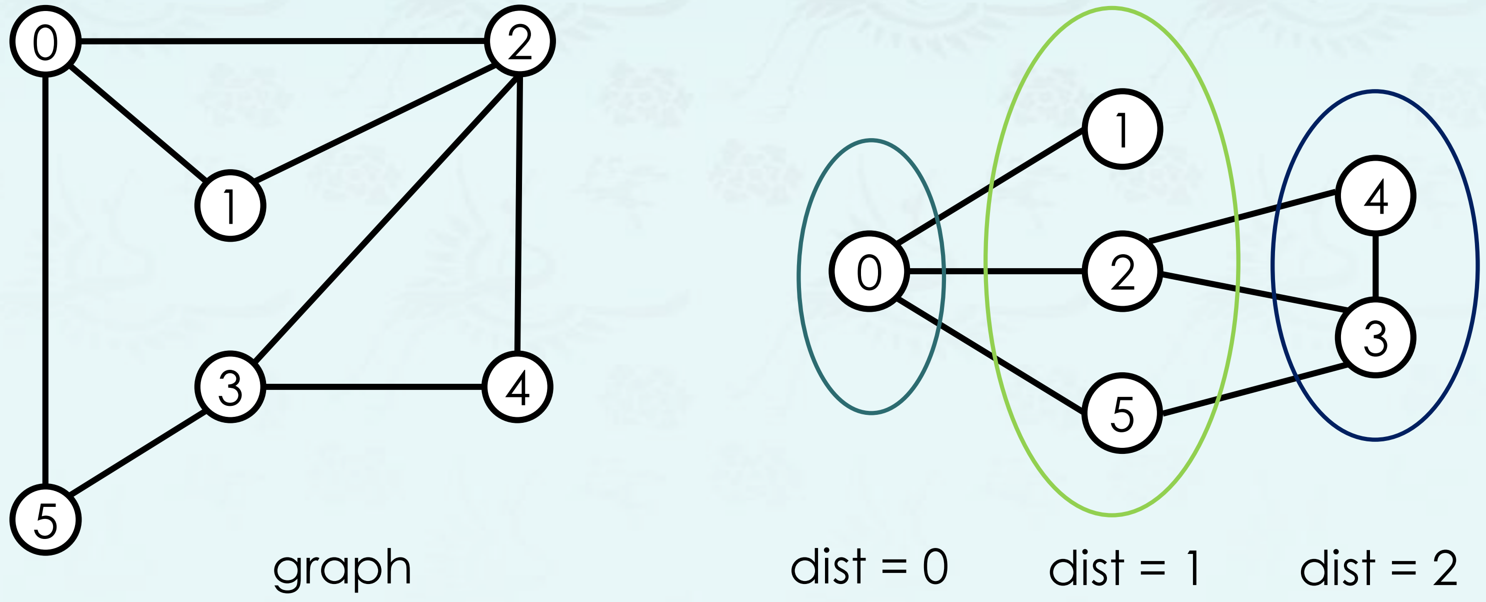
- **Intuition:** BFS examines vertices in increasing distance from s.

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.



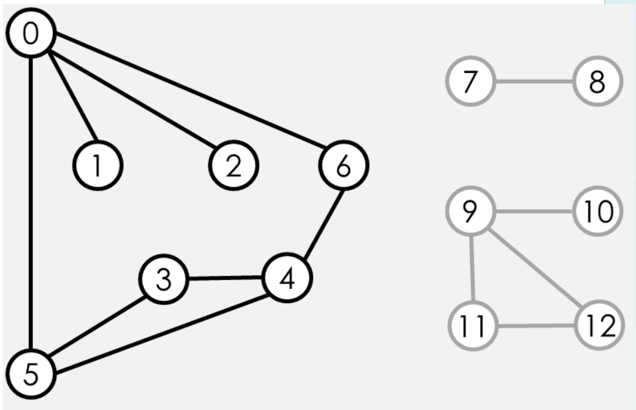
Breadth-first search implementation

```
// runs BFS at v and produces BFS0[], distTo[] & parentBFS[]
void BFS(graph g, int v) {
  queue<int> que;       // to process each vertex
  queue<int> sav;       // BFS result saved
  for (int i = 0; i < V(g); i++)  g->marked[i] = false;
  g->parentBFS[v] = -1;           g->marked[v] = true;
  g->distTo[v] = 0;               g->BFSv = {};
  que.push(v);                    sav.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]) {
        g->marked[w->item] = true;
        que.push(w->item);            // queued to process next
        sav.push(w->item);            // save the result
        cout << "your code here";     // set parentBFS[] & distTo[]
      }
    }
  }
  g->BFSv = sav;                  // save the result at v
  setBFS0(g, v, sav);
}
```

Breadth-first search implementation

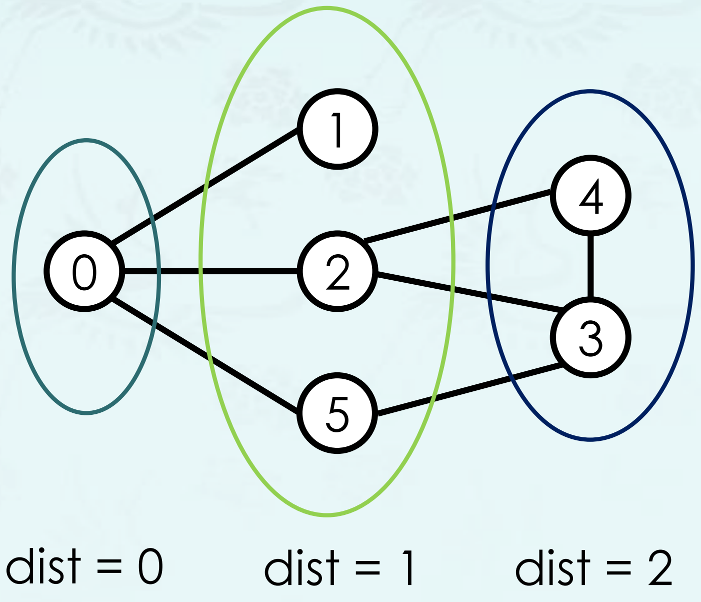
```
// runs BFS for all vertices or all connected components
// It begins with the first vertex 0 at the adjacent list.
// It produces BFS0[], distTo[] & parentBFS[].
void BFS_CCs(graph g) {
  if (empty(g)) return;

  for (int i = 0; i < V(g); i++) {
        g->marked[i] = false;
        g->parentBFS[i] = -1;
        g->BFS0[i] = -1;
        g->distTo[i] = -1;
  }

  BFS(g, 0);

  g->BFSv = {};
}
```

BFS() with a shortcoming



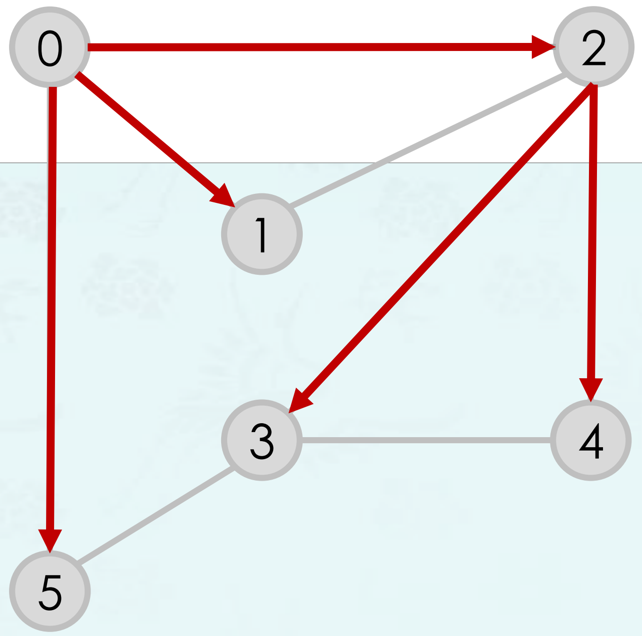
Breadth-first search implementation

```
// returns the number of edges in a shortest path between v and w
int distTo(graph g, int v, int w) {
  if (empty(g)) return 0;
  if (!connected(g, v, w)) return 0;

  BFS(g, v);

  cout << "your code here\n";

  return 0;
}
```

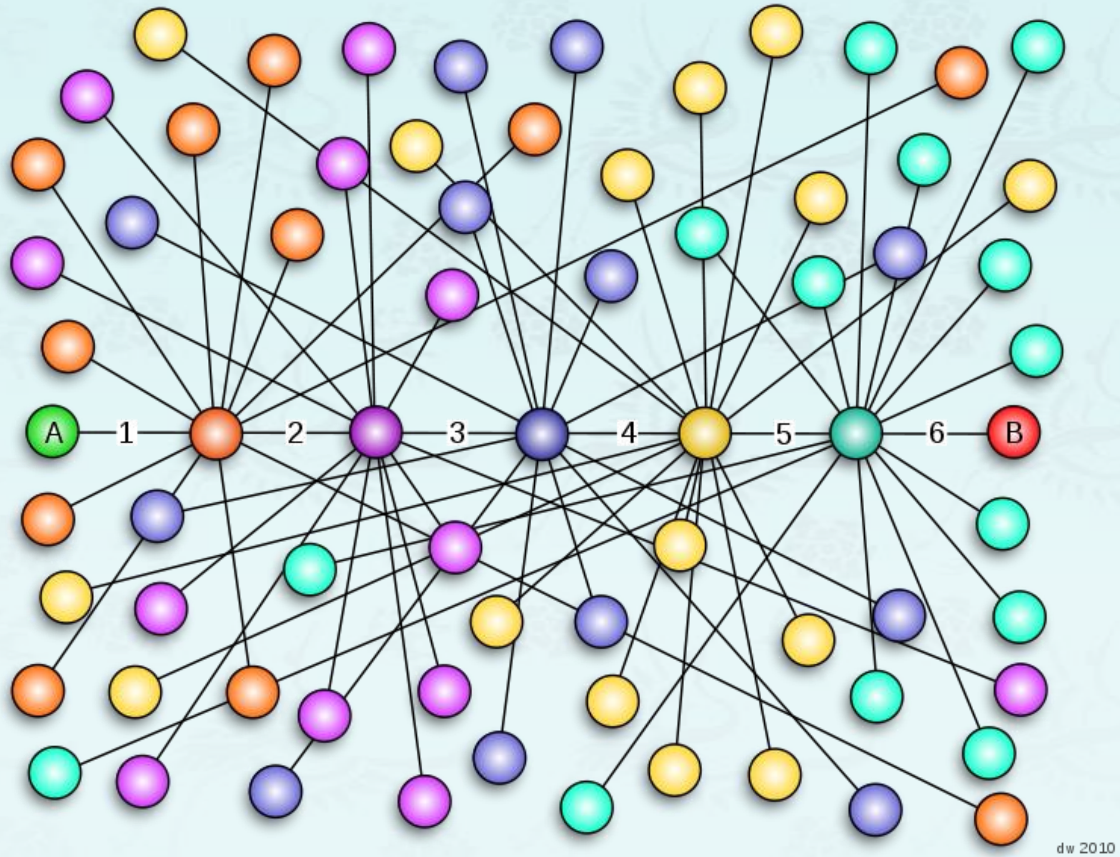


Breadth-first search implementation

```
// returns a path from v to w using the BFS result or parentBFS[].
// It has to use a stack to retrace the path back to the source.
// Once the client(caller) gets a stack returned,
void BFSpath(graph g, int v, int w, stack<int>& path) {
  if (empty(g)) return;

  BFS(g, v);                    // g->BFSv updated already.

  path = {};                    // clear path

  cout << "your code here\n";
}
```



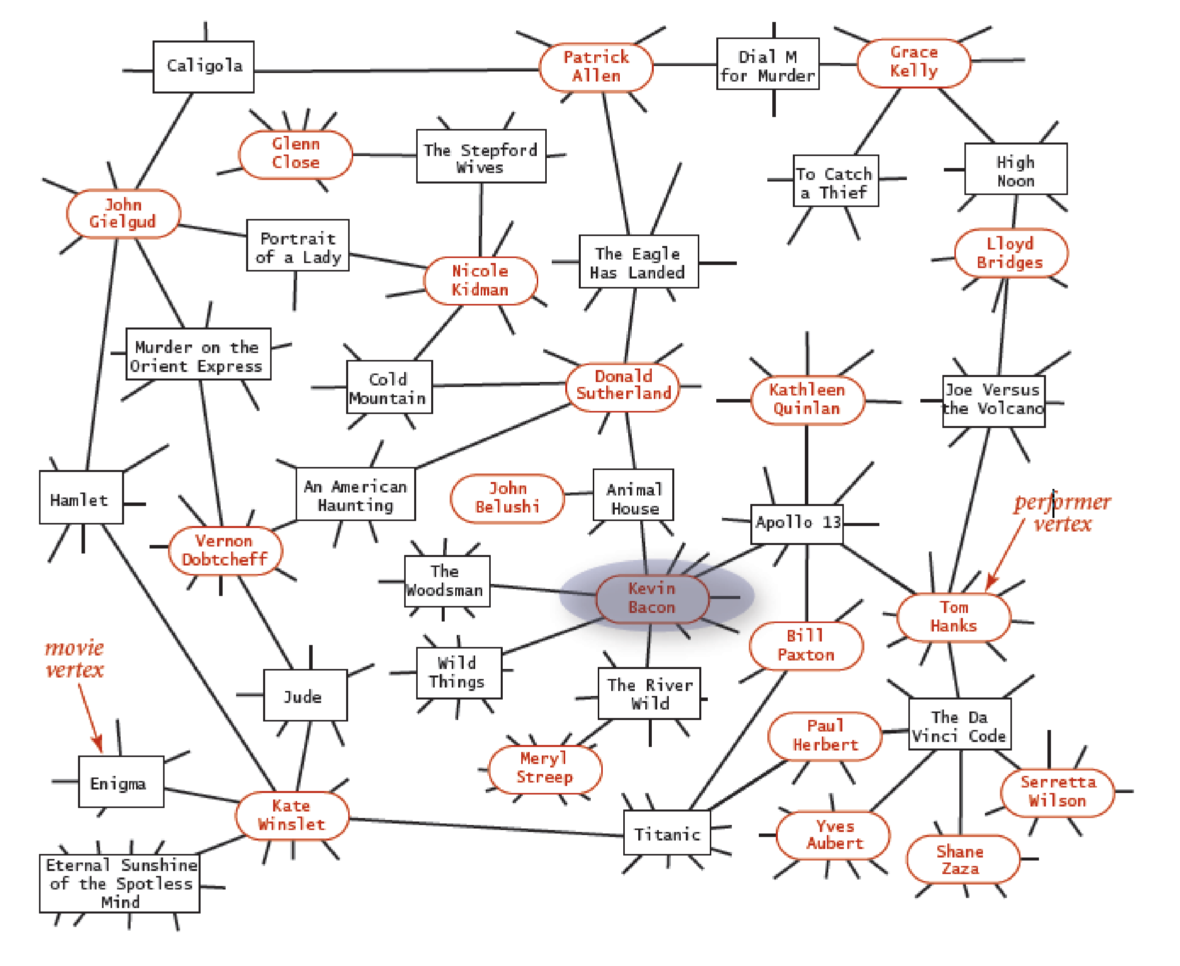
Breadth-first search application: Kevin Bacon numbers



six degrees of separation?

Breadth-first search application: Kevin Bacon numbers

- 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 demo

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

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.

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 demo

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

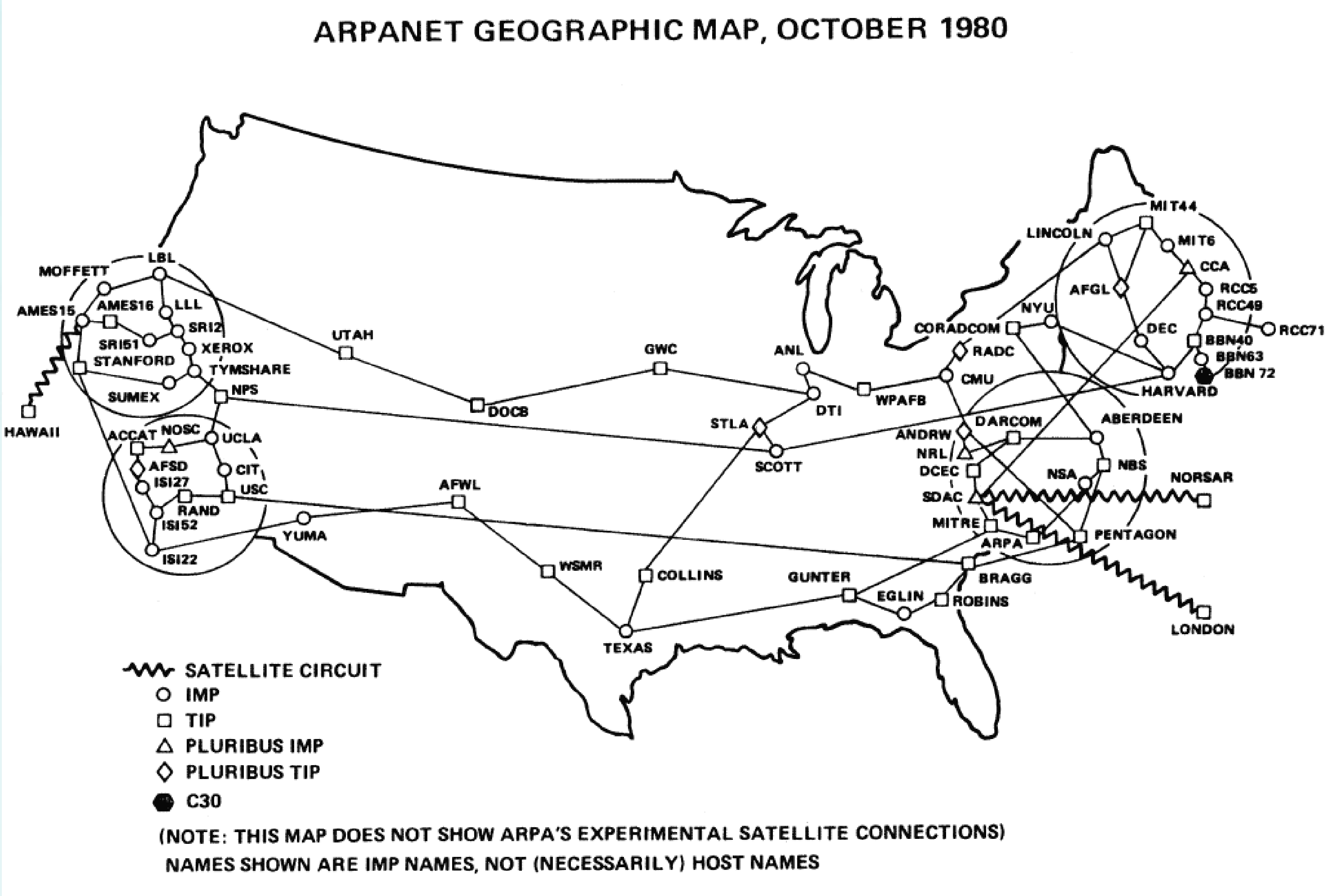
Facebook users are getting more connected to each other as the service grows and matures, according to a new study by the company's data team and the University of Milan. Instead of the traditional "six degrees of separation" that researchers have historically observed between all people in the world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

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.



Data Structures
Chapter 7: Graph

1. Introduction
   - Terminology, Representation, ADT
2. Basic Operations
   - DFS, CC, **BFS**, Processing
3. Digraph and Applications
4. Minimum Spanning Tree(MST)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University