# CS 341: Algorithms
# Lec 05: Breadth First Search

Armin Jamshidpey    Collin Roberts

Based on lecture notes by Éric Schost

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2025

# Goals

**This module:**

- basics on **undirected graphs**
- undirected BFS and applications (shortest paths, bipartite graphs, connected components)
- undirected DFS and applications (cut vertices)
- basics on **directed graphs**
- directed DFS and applications (testing for cycles, topological sort, strongly connected components)

# Undirected graphs

**Definition, notation:** a graph $\boldsymbol{G}$ is pair $\boldsymbol{(V, E)}$:

- $V$ is a finite set, whose elements are called vertices
- $E$ is a finite set, whose elements are unordered pairs of distinct vertices, and are called edges.

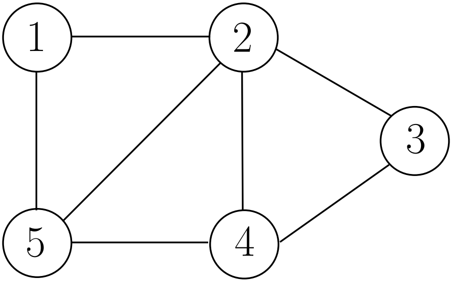
Convention: $\boldsymbol{n}$ is the number of vertices, $\boldsymbol{m}$ is the number of edges.

**Data structures:**

- **adjacency list:** an array $A[1..n]$ s.t. $A[v]$ is the linked list of all edges connected to $v$.
  $\boldsymbol{2m}$ list cells, total size $\boldsymbol{\Theta(n+m)}$, but testing if an edge exists is not $O(1)$

- **adjacency matrix:** a $(0, 1)$ matrix $M$ of size $n \times n$, with $M[v, w] = 1$ iff $\{v, w\}$ is an edge.
  size $\boldsymbol{\Theta(n^2)}$, but testing if an edge exists is $O(1)$

# Representations of graphs




[CLRS]

# Connected graphs, path, cycles, trees

**Definition:**

- path: a sequence $v_1, \ldots, v_k$ of vertices, with $\{v_i, v_{i+1}\}$ in $E$ for all $i$.
  $k = 1$ is OK.
- connected graph: $G = (V, E)$ such that for all $v, w$ in $V$, there is a path $v \rightsquigarrow w$
- cycle: a path $\boldsymbol{v_1, \ldots, v_k, v_1}$ with $\boldsymbol{k \geq 3}$ and $v_i$'s pairwise distinct
- tree: a connected graph without any cycle
- rooted tree: a tree with a special vertex called **root**

# Subgraphs, connected components

**Definition:**

- subgraph of $G = (V, E)$: a graph $G' = (V', E')$, where
  - $V' \subset V$
  - $E' \subset E$, with all edges $E'$ joining vertices from $V'$
- connected component of $G = (V, E)$
  - a connected subgraph of $G$
  - that is not contained in a larger connected subgraph of $G$

Let $G_i = (V_i, E_i)$, $i = 1, \ldots, s$ be the connected components of $G = (V, E)$.

- the $V_i$'s are a partition of $V$, with $\sum_i n_i = n$ $\qquad \boldsymbol{n_i = |V_i|}$
- the $E_i$'s are a partition of $E$, with $\sum_i m_i = m$ $\qquad \boldsymbol{m_i = |E_i|}$

# Breadth-first search

# Breadth-first exploration Idea

**Activity**

Assume we are looking for a person in a social network and we don't want to use the usual search. What is a possible strategy?

https://padlet.com/arminjamshidpey/CS341

# Breadth-first exploration of a graph

```
BFS(G, s)
G: a graph with n vertices, given by adjacency lists
s: a vertex from G
    let Q be an empty queue
    let visited be an array of size n, with all entries set to false
    enqueue(s, Q)
    visited[s] ← true
    while Q not empty do
        v ← dequeue(Q)
        for all w neighbours of v do
            if visited[w] is false
                enqueue(w, Q)
                visited[w] ← true
```

# Complexity

**Anaysis:**

- each vertex is enqueued at most once
- so each vertex is dequeued at most once
- so each adjacency list is read at most once

For all $v$, write $d_v$ = number of neighbours of $v$ = length of $A[v]$ = **degree** of $v$.

Then total cost at step 7 is

$$O\left(\sum_v d_v\right) = O(m)$$

cf. the adjacency array $A$ has $2m$ cells (handshaking lemma)

**Total:** $\boldsymbol{O(n+m)}$

### True/False

For all vertices $v$, there is a path $s \rightsquigarrow v$ in $G$ **if and only if** visited$[v]$ is true at the end.

`https://padlet.com/arminjamshidpey/CS341`

# Correctness

**Claim**

For all vertices $v$, if $\mathsf{visited}[v]$ is true at the end, there is a path $s \rightsquigarrow v$ in $G$

**Proof.** Let $s = v_0, \ldots, v_K$ be the vertices for which visited is set to true, in this order. We prove: **for all $\boldsymbol{i}$, there is a path $\boldsymbol{s \rightsquigarrow v_i}$**, by induction.

- OK for $i = 0$
- suppose true for $v_0, \ldots, v_{i-1}$.

  when $\mathsf{visited}[v_i]$ is set to true, we are examining the neighbours of a certain $v_j$, $\boldsymbol{j < i}$.

  by assumption, there is a path $s \rightsquigarrow v_j$

  because $\{v_j, v_i\}$ is in $E$, there is a path $s \rightsquigarrow v_i$

# Correctness

**Claim**

For all vertices $v$, if there is a path $s \rightsquigarrow v$ in $G$, $\mathsf{visited}[v]$ is true at the end

**Proof.** Let $v_0 = s, \ldots, v_k = v$ be a path $s \rightsquigarrow v$. We prove $\mathsf{visited}[v_i]$ is true for all $i$, by induction.

- $\mathsf{visited}[v_0]$ is true
- if $\mathsf{visited}[v_i]$ is true, we will examine all neighbours $u$ of $v_i$

  so at the end of Step 7, all $\mathsf{visited}[u]$ will be true, for $u$ neighbour of $v_i$

  in particular, $\mathsf{visited}[v_{i+1}]$ will be true

# Correctness

**Lemma**

For all vertices $v$, there is a path $s \rightsquigarrow v$ in $G$ **if and only if** visited$[v]$ is true at the end

**Applications**

- testing if there is a path $s \rightsquigarrow v$
- testing if $G$ is connected

in $\boldsymbol{O(n+m)}$.

**Exercise**

For a connected graph, $m \geq n - 1$.

# Keeping track of parents and levels

**BFS**$(G, s)$

```
let Q be an empty queue
let parent be an array of size n, with all entries set to NIL
let level be an array of size n, with all entries set to ∞
enqueue(s, Q)
parent[s] ← s
level[s] ← 0
while Q not empty do
    v ← dequeue(Q)
    for all w neighbours of v do
        if parent[w] is NIL
            enqueue(w, Q)
            parent[w] ← v
            level[w] ← level[v] + 1
```

# BFS tree

**Definition:** the **BFS tree** $T$ is the subgraph made of:

- all $w$ such that $\mathsf{parent}[w] \neq \mathbf{NIL}$.
- all edges $\{w, \mathsf{parent}[w]\}$, for $w$ as above (except $w = s$)

**Claim**

The BFS tree $T$ is a tree

**Proof:** by induction on the vertices for which $\mathsf{parent}[v]$ is not **NIL**

- when we set $\mathsf{parent}[s] \leftarrow s$, only one vertex, no edge.
- suppose true before we set $\mathsf{parent}[w] \leftarrow v$

  $v$ was in $T$ before, $w$ was not, so we add one vertex $w$ and one edge $\{v, w\}$ to $T$

  so $T$ remains a tree

**Remark:** we make it a **rooted** tree by choosing $s$ as root

# Shortest paths from the BFS tree

**Sub-claim 1**

The levels in the queue are non-decreasing

**Proof:** Exercise.

**Sub-claim 2**

For all vertices $u, v$, if there is an edge $\{u, v\}$, then $\mathsf{level}[v] \leq \mathsf{level}[u] + 1$.

**Proof:**

- if we dequeue $v$ before $u$, $\mathsf{level}[v] \leq \mathsf{level}[u]$ sub-claim 1
- if we dequeue $u$ before $v$, the parent of $v$ is either $u$, or was dequeued before $u$

  in any case, $\mathsf{level}[\mathsf{parent}[v]] \leq \mathsf{level}[u]$ sub-claim 1

  but $\mathsf{level}[\mathsf{parent}[v]] = \mathsf{level}[v] - 1$, so OK

# Shortest paths from the BFS tree

**Claim**

For all $v$ in $G$:

- there is a path $s \rightsquigarrow v$ in $G$ iff there is a path $s \rightsquigarrow v$ in $T$
- if so, the path in $T$ is a shortest path and $\mathsf{level}[v] = \mathrm{dist}(s, v)$

**Proof.** First item: Exercise.

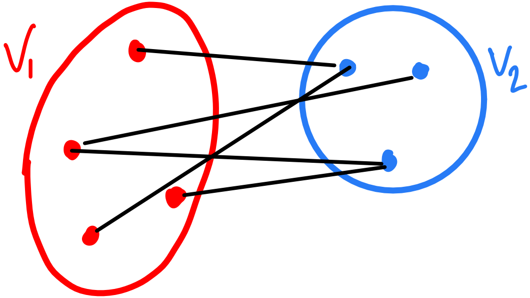
Second item:

- $\mathrm{dist}(s, v) \le \mathsf{level}[v]$ (follow the path on $T$)
- for all $i$, for all $v$, if there is a path $s \rightsquigarrow v$ of length $i$, then $\mathsf{level}[v] \le i$.
  - true for $i = 0$
  - suppose true for $i - 1$ and take $v$, a path $s \rightsquigarrow v$ of length $i$ and let $u$ be the vertex before $v$.
    induction assumption: $\mathsf{level}[u] \le i - 1$, so $\mathsf{level}[v] \le i$
    sub-claim 2
- so $\mathsf{level}[v] \le \mathrm{dist}(s, v)$.

# Bipartite graphs

**Definition**

- a graph $G = (V, E)$ is **bipartite** if there is a partition $V = \boldsymbol{V_1} \cup \boldsymbol{V_2}$ such that all edges have one end in $V_1$ and one end in $V_2$.



# Using BFS to test bipartite-ness

**Claim.**

Suppose $G$ connected, run BFS from any $s$, and set

- $V_1$ = vertices with odd level
- $V_2$ = vertices with even level.

Then $G$ is bipartite if and only if all edges have one end in $V_1$ and one end in $V_2$ (testable in $\boldsymbol{O(m)}$)

**Proof.** $\Longleftarrow$ obvious.

For $\Longrightarrow$, let $W_1, W_2$ be a bipartition. Because paths alternate between $W_1, W_2$:

- $V_1$ (= vertices with odd level) is included in $W_1$ (say)
- $V_2$ (= vertices with even level) is included in $W_2$

So $\boldsymbol{V_1 = W_1}$ and $\boldsymbol{V_2 = W_2}$.

# Computing the connected components

**Idea:** add an outer loop that runs BFS on successive vertices

**Exercise**

Fill in the details.

**Complexity:**

- each pass of BFS $\boldsymbol{O(n_i + m_i)}$, if $G_i(V_i, E_i)$ is the $i$th connected component
- total $\boldsymbol{O(n + m)}$