Lecture 4: BFS, Graph Representations, DFS

Sam McCauley

February 17, 2025

Welcome Back!

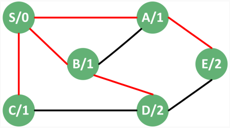
- Problem set 0 should be back tomorrow
- New homework out; current homework in at end of class
- TA hours and office hours new rooms! (Posted on website.)
- Tips on how to approach proofs handout posted tonight
    - Let me know if you have questions or comments

Breadth-First Search

Breadth-First Search (BFS)

- We’ll refer to as BFS
- Idea: start with some node s
- Slowly explore outwards from s
- “peeling one layer after another”

BFS Definition: Very High Level Intuition



- We start with some vertex s
- Then we explore the neighbors of s
- Then the neighbors of the neighbors of s
- Then the neighbors of the neighbors of the neighbors of s
- And so on until there are no new neighbors to explore

BFS Definition: Intuition

We define BFS using a sequence of layers

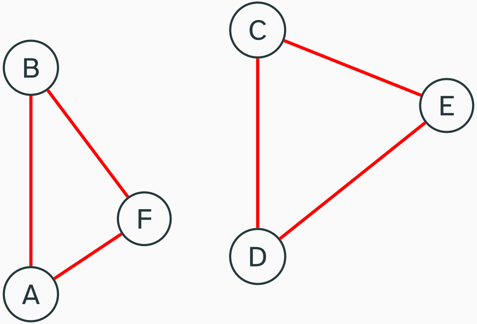
- Initialize $L_0 = \{s\}$, $i = 0$; mark s as visited
- if there exists a node in L_i with an unvisited neighbor:
  - Set L_{i+1} to be all unvisited neighbors of nodes in L_i; mark all nodes in L_{i+1} as visited; set $i = i + 1$

Let's do an example [On Board #1]

Any questions about this algorithm? We'll look at pseudocode for this algorithm later today

What does BFS Do?

- Keeps exploring until run out of nodes to explore
- Question: can you give an example of a graph (and a starting vertex s in the graph) where BFS does not visit all nodes?



$$V = \{A, B, C, D, E\}$$

$$E = \{(A, B), (A, F), (B, F),\\ (C, D), (C, E), (D, E)\}$$

If the graph is not connected, BFS will not visit all nodes.

Properties of BFS

- We saw with Gale Shapley: analyzing an algorithm can tell you something about the *problem itself*

- Let’s look at two properties of BFS

- You will need to use these on your problem sets and on midterm 1

- They are useful for:
  - analyzing BFS
  - creating new algorithms
  - analyzing the structure of graphs in general!

First key BFS Property

Idea: For any edge (x, y) in an *undirected* graph, x and y are stored in the same level, or adjacent levels.

Theorem

For any undirected graph G, if $(x, y) \in E$, and $x \in L_i$ and $y \in L_j$ for a BFS starting at some node s, then i and j differ by at most 1 (that is to say: $|i - j| \leq 1$).

Proof: Assume the contrary, that $i - j \geq 2$ or $j - i \geq 2$.

First, let's say $i \geq j + 2$. Since $y \in L_j$, all unvisited neighbors of y are added to L_{j+1}. Since x is not in level $L_{j'}$ for $j' \leq j$, x is unvisited, so x is added to L_{j+1}, a contradiction.

Second, let's say $j \geq i + 2$. (This case is basically identical.) Since $x \in L_i$, all unvisited neighbors of x are added to L_{i+1}. Since y is not in level $L_{i'}$ for $i' \leq i$, y is unvisited, so y is added to L_{i+1}, a contradiction.

BFS and Connected Graphs

Lemma

In any connected undirected graph G, BFS starting at vertex s will visit every vertex.

Can we prove this using the BFS property we showed?

Consider some vertex v; we show that BFS visits v. Since G is connected there is a path from s to v; call this path $p = s, v_1, v_2, \ldots, v_k, v$.

Idea: We have that s $\in L_0$. Since v_1 is a neighbor of s, $v_1 \in L_1$. Let's generalize to all v_i using an induction.

Proof by induction: v_i is in level L_j for some $j \leq i$. Base case: $i = 1$ by above. Assume true for some i. Since v_{i+1} is a neighbor of v_i, then v_{i+1} must be in level $L_{j'}$ where $|j - j'| \leq 1$. Since $j \leq i$, we must have $j' \leq i + 1$.

Running BFS

- On disconnected graphs: if we run out of vertices, start again from a new unvisited vertex
- Cost for BFS to explore a node v with d_v neighbors?
- Answer: $O(1 + d_v)$
- Total running time:

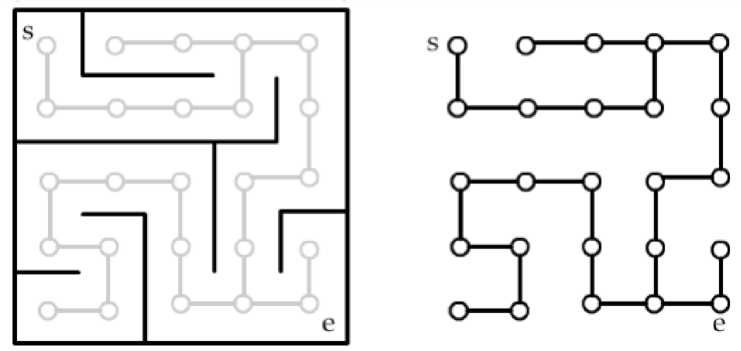
$$\sum_{v \in V} O(1 + d_v) = O\left(n + \sum_{v \in V} d_v\right) = O(n + 2m) = O(n + m)$$

Recall that since each edge is adjacent to two vertices, $\sum d_v = 2|E|$.

The BFS Tree

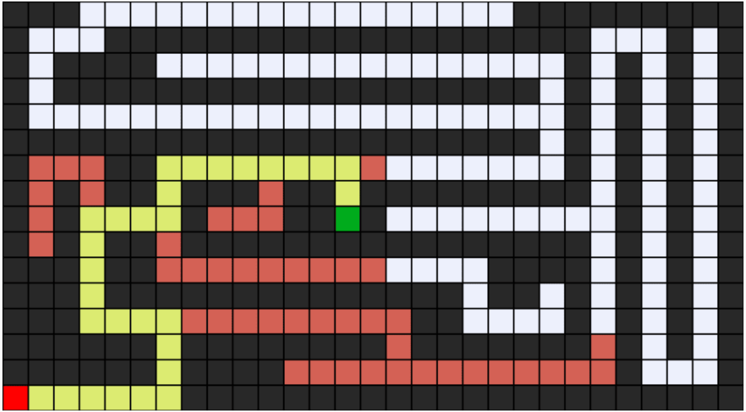
- The levels explored by the BFS are the levels of a tree (i.e. the nodes at a particular height)
- If v' is a neighbor of v that we add to some level, then v is the parent of v'.
- Let's do an example together [On Board #2]
- The vertices at level d of the BFS tree are exactly the vertices in layer L_d
- We can calculate the BFS tree while doing the BFS in $O(n+m)$ time
  - Useful for some applications!
  - And some Problem Sets

Application: Maze Solving



- BFS can find if a maze is solvable!
- Turn the maze into a graph: node for each square; edge if can get from one square to another
- How can we prove that BFS *always* solves the maze if possible?
- Animation: `https://youtu.be/zMy5MwQWwss?si=VRNW3sgRgMeK7aVd&t=129`

Application: Maze Solving



- How do we get the path from start to end of the maze?
- One answer: use the BFS tree!
- Path from s to e in the tree is a path from s to e in the maze

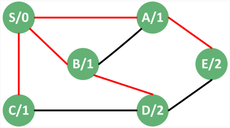
Second Key Property: BFS to find Shortest Path

- BFS gives the *shortest path* between the initial vertex s and any other vertex *v* in the graph
  - We call the length of the shortest path between two vertices *u* and *v* the *distance* betwen *u* and *v*
- How can we formalize?

Theorem

For any vertex v in any graph G (directed or undirected), if v is at height d of the BFS tree rooted at s (in other words, if v is in L_d), then the shortest path from s to v has length d.

Building Intuition on BFS for Shortest Path



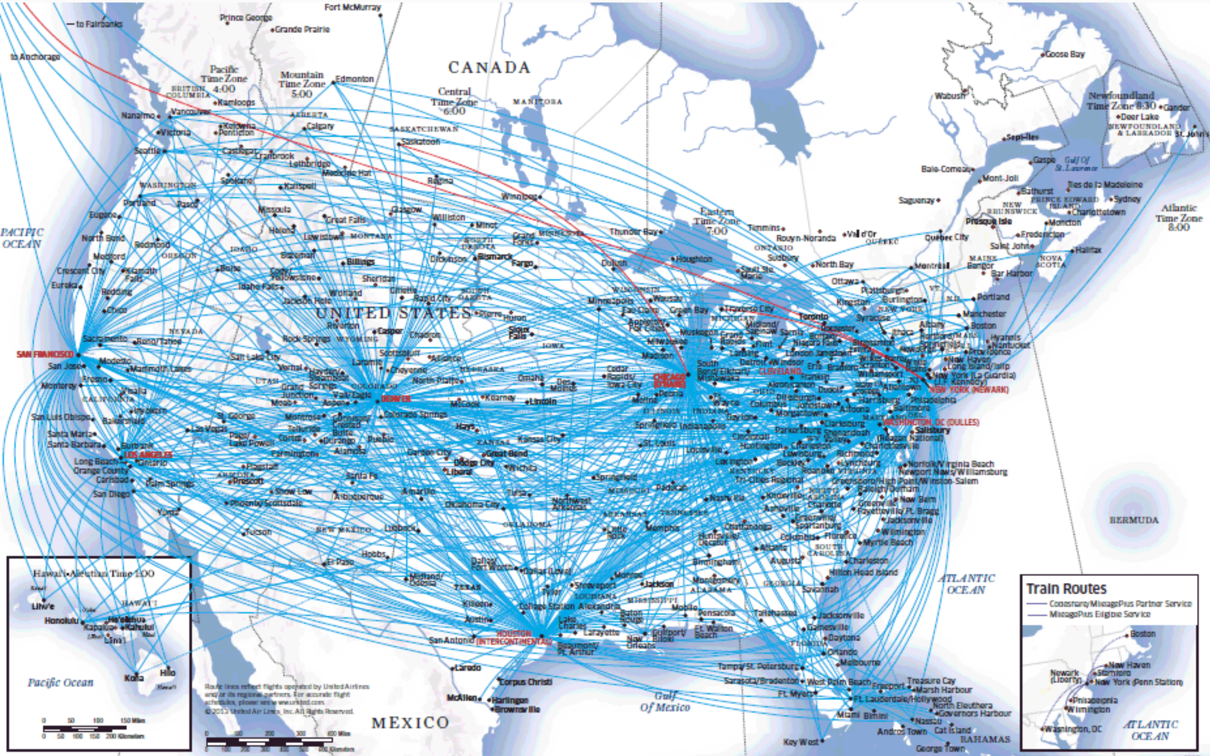
- We start with some vertex s

- Then we explore the neighbors of s (each has distance 1)

- Then any unexplored neighbors of the neighbors of s (each has distance 2)

- Then the unexplored neighbors of the neighbors of the neighbors of s (each has distance 3)

- And so on until there are no new neighbors to explore



CANADA
UNITED STATES
MEXICO
PACIFIC OCEAN
ATLANTIC OCEAN
Gulf Of Mexico
BERMUDA
BAHAMAS
Pacific Time Zone 4:00
Mountain Time Zone 5:00
Central Time Zone 6:00
Eastern Time Zone 7:00
Atlantic Time Zone 8:00
Newfoundland Time Zone 8:30
to Fairbanks
to Anchorage
Fort McMurray
Prince George
Grande Prairie
Edmonton
Calgary
Saskatoon
Regina
Winnipeg
Vancouver
Victoria
Nanaimo
Kamloops
Kelowna
Penticton
Castlegar
Cranbrook
Lethbridge
Medicine Hat
Thunder Bay
Timmins
Rouyn-Noranda
Val d'Or
Sudbury
North Bay
Ottawa
Montreal
Quebec City
Toronto
Kingston
Saguenay
Baie-Comeau
Mont-Joli
Sept-Iles
Gaspe
Wabush
Goose Bay
Gander
Deer Lake
St. John's
Bathurst
Iles de la Madeleine
Sydney
Charlottetown
Moncton
Fredericton
Saint John
Halifax
Presque Isle
Bangor
Bar Harbor
Portland
Seattle
Spokane
Pasco
Eugene
North Bend
Redmond
Medford
Klamath Falls
Crescent City
Eureka
Redding
Chico
Sacramento
Reno/Tahoe
SAN FRANCISCO
San Jose
Modesto
Mammoth Lakes
Fresno
Monterey
Visalia
Inyokern
San Luis Obispo
Bakersfield
Santa Maria
Santa Barbara
Burbank
LOS ANGELES
Long Beach
Orange County
Ontario
Carlsbad
Palm Springs
San Diego
Yuma
Las Vegas
Boise
Idaho Falls
Jackson Hole
Salt Lake City
Missoula
Kalispell
Great Falls
Helena
Bozeman
Billings
Glasgow
Williston
Minot
Bismarck
Dickinson
Fargo
Grand Forks
Sheridan
Gillette
Cody/Yellowstone
Worland
Riverton
Casper
Rock Springs
Rapid City
Pierre
Huron
Sioux Falls
Chadron
Scottsbluff
Alliance
Cheyenne
Laramie
Vernal
Hayden/Steamboat Springs
Vail/Eagle
Aspen
Grand Junction
Moab
DENVER
Colorado Springs
Montrose
Gunnison/Crested Butte
Telluride
Cortez
Durango
Pueblo
Alamosa
Farmington
St. George
Page/Lake Powell
Flagstaff
Prescott
Show Low
Phoenix/Scottsdale
Tucson
Santa Fe
Albuquerque
Amarillo
Lubbock
Hobbs
El Paso
Midland/Odessa
North Platte
Kearney
McCook
Lincoln
Omaha
Des Moines
Hays
Garden City
Dodge City
Liberal
Great Bend
Wichita
Kansas City
Tulsa
Oklahoma City
Springfield
Northwest Arkansas
Little Rock
Dallas/Fort Worth
Dallas (Love)
Tyler
Killeen
Austin
College Station
San Antonio
HOUSTON (INTERCONTINENTAL)
Beaumont/Pt. Arthur
Corpus Christi
Laredo
McAllen
Harlingen
Brownsville
Shreveport
Monroe
Alexandria
Lake Charles
Lafayette
Baton Rouge
New Orleans
Jackson
Gulfport/Biloxi
Mobile
Pensacola
Ft. Walton Beach
Memphis
Minneapolis
Duluth
Houghton
Wausau
Eau Claire
Green Bay
Appleton
Fox Cities
Milwaukee
Madison
Cedar Rapids/Iowa City
Moline
Peoria
CHICAGO (O'HARE)
Springfield
St. Louis
Traverse City
Muskegon
Grand Rapids
Lansing
Flint
Midland/Saginaw
Sarnia
South Bend/Elkhart/Mishawaka
Ft. Wayne
Indianapolis
Dayton
Columbus
Cincinnati
Louisville
Lexington
Paducah
Nashville
Knoxville
Chattanooga
Huntsville/Decatur
Birmingham
Montgomery
Atlanta
Augusta
Columbus
Savannah
Detroit/Windsor
London
Erie
CLEVELAND
Akron/Canton
Pittsburgh
Johnstown
Franklin
Bradford
DuBois
Morgantown
Clarksburg
Parkersburg
Huntington
Charleston
Lewisburg
Beckley
Roanoke
Lynchburg
Shenandoah Valley
Charlottesville
Richmond
Norfolk/Virginia Beach
Newport News/Williamsburg
Tri-Cities Regional
Greensboro/High Point/Winston-Salem
Raleigh/Durham
Asheville
Charlotte
Greenville/Spartanburg
Greenville
New Bern
Fayetteville/Ft. Bragg
Jacksonville
Wilmington
Myrtle Beach
Florence
Columbia
Charleston
Hilton Head Island
Tallahassee
Jacksonville
Gainesville
Daytona
Orlando
Melbourne
Tampa/St. Petersburg
Sarasota/Bradenton
Ft. Myers
West Palm Beach
Ft. Lauderdale/Hollywood
Miami
Key West
Bimini
Freeport
Treasure Cay
Marsh Harbour
North Eleuthera
Governors Harbour
Nassau
Andros Town
Cat Island
George Town
Niagara Falls
Buffalo
Rochester
Syracuse
Ithaca
Binghamton
Elmira
Jamestown
Plattsburgh
Burlington
Albany
Manchester
Boston
Hartford/Springfield
Providence
Hyannis
Nantucket
New Haven
Long Island/Islip
New York (La Guardia)
New York (J.F. Kennedy)
NEW YORK (NEWARK)
Wilkes-Barre/Scranton
Williamsport
State College
Altoona
Harrisburg
Allentown
Philadelphia
Baltimore
WASHINGTON, DC (DULLES)
(Reagan National)
Salisbury
Hawaii-Aleutian Time 2:00
Lihu'e
Honolulu
Kapalua
Ho'olehua
Kahului
Lana'i
Kona
Hilo
Pacific Ocean
0 50 100 150 Miles
0 50 100 150 200 Kilometers
Route lines reflect flights operated by United Airlines and/or its regional partners. For accurate flight schedules, please see www.united.com.
©2013 United Air Lines, Inc. All Rights Reserved.
Train Routes
Coastliner/MileagePlus Partner Service
MileagePlus Eligible Service
Boston
New Haven
Stamford
Newark (Liberty)
New York (Penn Station)
Philadelphia
Wilmington
Washington, DC
ATLANTIC OCEAN

BFS to find Shortest Path: Proof

Theorem

For any vertex v in any graph G, v is at depth d of the BFS tree rooted at s if and only if the shortest path from s to v has length d.

Let's discuss. What will this proof look like?

- We'll proceed by strong induction on d.
- I see "if and only if". That means we need to prove two directions:
  - If v is in L_d, its distance from s is d; and
  - if v has distance d from s, it is in L_d.

BFS to find Shortest Path: Proof

Theorem

For any vertex v in any graph G, v is in L_d if and only if the shortest path from s to v has length d.

Proof: By strong induction on d. Base case: for $d = 0$, the only vertex with a shortest path of length 0 from s is s; we have that $L_0 = \{s\}$ by definition of BFS.

Now, assume that for some d, for all $1 \leq k \leq d$, L_k consists of all vertices whose shortest path from s has length k. (Goal: show that L_{d+1} consists of all vertices w/ shortest path length $d + 1$.)

First, we show that if a vertex v is in L_{d+1}, its shortest path from s has length $d + 1$. We break this into two parts: first we show that there exists a path of length $d + 1$; then we show that no path has length $< d + 1$.

BFS to find Shortest Path

Proof: (Recall:) Now, assume that for some d, for all $1 \leq k \leq d$, L_k consists of all vertices whose shortest path from s has length k. (Goal: show that L_{d+1} consists of all vertices w/ shortest path length $d+1$.)

First, we show that if a vertex v is in L_{d+1}, its shortest path from s has length $d+1$. We break this into two parts: first we show that there exists a path of length $d+1$; then we show that no path has length $< d+1$.

Moving forward: Since $v \in L_{d+1}$, v has a neighbor $v' \in L_d$. By the I.H., the shortest path from s to v' has length d. Therefore, there is a path from s to v of length $d+1$, so the shortest path from s to v has length *at most* $d+1$.

Now, we show that no path from s to v has length $< d+1$. Consider a path of length k, $p = s, v_1, \ldots, v_{k-1}, v$ for $k < d+1$. By the I.H., v_{k-1} is in level L_{k-1}; but since there is an edge from v_{k-1} to v, v must be in L_k or earlier, contradicting our assumption that $v \in L_{d+1}$.

BFS to find Shortest Path

Proof: *Recall:* Proof by strong induction on d. Let's do the other direction.

Now, assume that for some d, for all $1 \leq k \leq d$, L_k consists of all vertices whose shortest path from s has length k. (Goal: show that L_{d+1} consists of all vertices w/ shortest path length $d + 1$.)

Now, we show that if the shortest path from s to v has length $d + 1$, then $v \in L_{d+1}$. By I.H., $v \notin L_j$ for $j < d + 1$.

Let $p = s, v_1, \ldots, v_d, v$ be a path of length $d + 1$ from s to v. By the I.H., $v_d \in L_d$. When we explore the neighbors of v_d, we cannot have already explored v since $v \notin L_j$ for $j < d + 1$; thus $v \in L_{d+1}$

BFS to find Shortest Path (wrapup)

Theorem

For any vertex v in any graph G, v is at depth d of the BFS tree rooted at s if and only if the shortest path from s to v has length d.

Proof: By strong induction on d. Base case: for $d = 0$, the only vertex with a shortest path of length 0 from s is s; we have that $L_0 = \{s\}$ by definition of BFS.

Summary: We have shown that assuming the I.H. for all $1 \leq k \leq d$, if $v \in L_{d+1}$, then the shortest path from s to v has length $d + 1$; furthermore, if the shortest path from s to v has length $d + 1$, then $v \in L_{d+1}$. Therefore the inductive step is complete.

BFS Properties (Review/reference)

Useful shorthand: if $x \in L_i$, we also write $i = L[x]$.

Lemma

For any undirected graph G, if $(x, y) \in E$, then for any BFS tree on G, $|L[x] - L[y]| \leq 1$.

Theorem

In a connected undirected graph G, BFS starting at any vertex s will visit every vertex.

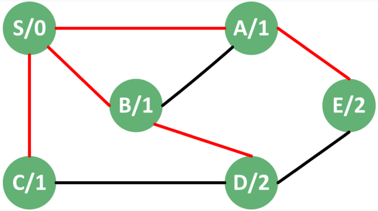
Theorem

In any graph G, for any vertex v explored using BFS, $L[v]$ is the distance from s to v.

Theorem

BFS runs in $O(n + m)$ time on any graph with n vertices and m edges.

BFS Properties Summary



- Starts at some node s
- Partitions vertices into levels $L_0, L_1, \ldots$
- Gives a BFS tree T; a vertex at height h in the tree is in L_h
- If $(x, y) \in E$, the level of x and y differ by ≤ 1
- A vertex is at height h in T if and only if its shortest path from s has distance h

Implementing BFS

Implementing BFS

- Can we be more specific about how BFS works?
- Maybe give pseudocode?
- Do we need to store the levels explicitly? How should we store them?
- Key insight: we can explore the nodes in level L_i in the same order they were added to L_i. (And note that each was added before any node in L_{i+1})
- So: explore nodes in the same order they were visited! Don't need to keep track of the level

BFS Pseudocode

```
BFS(G, s):
    Put s in a queue Q
    while Q is not empty:
        v = Q.dequeue() # take the first vertex from Q
        if v is not marked as visited:
            mark v as visited
            for each edge (v,w):
                Q.enqueue(w) # add w to Q
```

Note: this algorithm only works if at start all vertices in G are not marked as visited!

- Question: How can we calculate the BFS tree T?
- Can we *guarantee* that this is equivalent to the level-by-level version of BFS?

Proof that BFS Algorithms are Equivalent

Theorem

In $BFS(G, s)$, all nodes in level L_i are explored (removed from the queue) before any node in level L_{i+1}

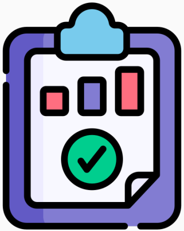
We'll use the following *invariant*: if at any time the first instance of the univisted nodes in the queue are in order $v_1, v_2, \ldots, v_k$, then

$$L[v_1] \leq L[v_2] \leq \cdots \leq L[v_k] \leq L[v_1] + 1.$$

If this invariant holds, then the theorem is true.

Some intuition: can we rephrase this equation in English?

Proof that BFS Algorithms are Equivalent



Inductive Hypothesis: if after x iterations of the while loop, the order of the first instance of univisted nodes in the queue $v_1, v_2, \ldots, v_k$, then $L[v_1] \leq L[v_2] \leq \cdots \leq L[v_k] \leq L[v_1] + 1$.

Base Case: For $x = 0$, the queue only contains s.

Inductive Step: Assume I.H. after some $x \geq 0$ iterations of the while loop. During $(x+1)$st iteration, v_1 is removed from the queue and its neighbors are added to the queue; let $u_1, \ldots, u_r$ be the unvisited neighbors that are not already in the queue. We have that $L[u_1] = L[u_2] = \cdots = L[u_r] = L[v_1] + 1$.

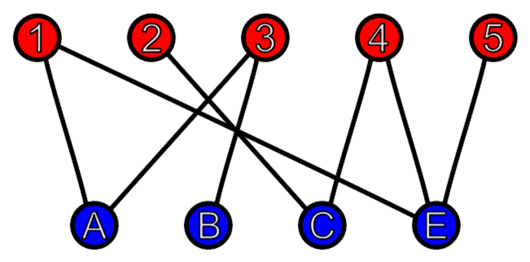
The queue now contains $v_2, v_3, \ldots, v_k, u_1, u_2, \ldots, u_r$. By I.H. and the above,

$$L[v_2] \leq L[v_3] \leq \cdots \leq L[v_k] \leq L[u_1] \leq \cdots \leq L[u_r] \leq L[v_1] + 1$$

Since we also had $L[v_1] \leq L[v_2]$ from I.H., we are done:

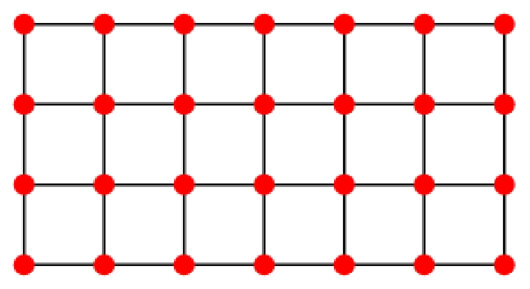
$$L[v_2] \leq L[v_3] \leq \cdots \leq L[v_k] \leq L[u_1] \leq \cdots \leq L[u_r] \leq L[v_2] + 1$$

Last BFS Application: Bipartite Testing



- Bipartite graph: graph G whose vertices can be partitioned into V_1, V_2 where every edge e has one endpoint in V_1 and one endpoint in V_2.

Last BFS Application: Bipartite Testing



- How can we test if a given undirected graph is bipartite?
  - Maybe greedily assign vertices to one set or the other? Does this always work?
  - Today: use BFS
  - Run BFS from any start vertex. If there is an edge between two vertices at the same level, return "not bipartite." Otherwise, return "bipartite."

Bipartite Testing

Theorem

The BFS bipartite testing algorithm is correct.

Proof (part 1: correct if returns "bipartite"): If the algorithm returns "bipartite," then G is bipartite.

Let V_1 be all vertices at even levels, and V_2 be all vertices at odd levels. We must show that every edge is between a vertex in V_1 and a vertex in V_2.

Consider an edge $e = (u, v)$. We must have that $|L[u] - L[v]| \leq 1$ by BFS property. We cannot have $L[u] = L[v]$, so $|L[u] - L[v]| = 1$. But then $u \in V_1$ and $v \in V_2$ (or vice versa). $\square$

Bipartite Testing

Theorem

The BFS bipartite testing algorithm is correct.

Proof (part 2: correct if returns “not bipartite”): If the algorithm returns “not bipartite,” there is an edge e between two vertices v_1 and v_2 at the same level k (for some k). Assume by contradiction that G is bipartite. Then v_1 and v_2 are in different partitions; let’s say $v_1 \in V_1$ and $v_2 \in V_2$.

Let p_1 be the path from s to v_1 in the BFS tree T, and let p_2 be the path from v_2 to s in T. Both p_1 and p_2 have length k.

Let $p_1 = (s = u_0, u_1, u_2, \ldots, u_k = v_1)$. We know that $u_k \in V_1$, so $u_{k-1} \in V_2$; and so on. So if k is odd, $s \in V_2$; if k is even then $s \in V_1$.

Let $p_2 = (v_2 = w_0, w_2, \ldots, w_k = s)$. We know that $w_1 \in V_2$, so $w_2 \in V_1$; and so on. So if k is odd, $s \in V_1$; if k is even then $s \in V_2$. In either case (k odd or even) we have a contradiction.

BFS is a simple algorithm, but—with careful analysis—it can accomplish quite a lot!