CS213/293 Data Structure and Algorithms 2025

Lecture 14: Graphs - Breadth-first search

Instructor: Ashutosh Gupta

IITB India

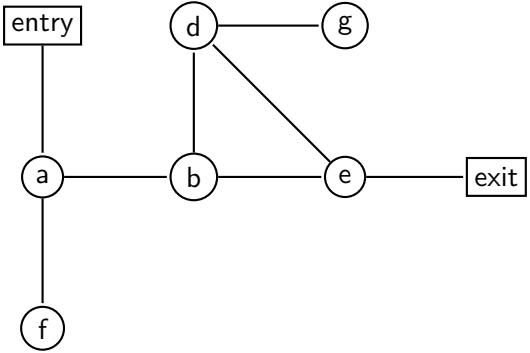
Compile date: October 6, 2025

Topic 14.1

Breadth-first search (BFS)

Solving a maze

What is a good way of solving a maze?



- Every choice point is a vertex
- Paths connecting the points are edges
- Problem: find the exit node

Breadth-first search

Definition 14.1

A breadth-first search(BFS) traverses a connected component in the following order.

- BFS starts at a vertex, which is at level 0.
- BFS traverses the unvisited adjacent vertices of level $n-1$ vertices, which are the vertices at level n.

The above traversal defines a spanning tree of the graph.

In the algorithm, we need to keep track of the already visited vertices and visit vertices at the lower level first.

Algorithm: BFS for search

Algorithm 14.1: BFS(Graph $G = (V, E)$, vertex r, Value x)

```
1  Queue Q;
2  set visited := {r};
3  Q.enqueue(r);
4  while not Q.empty() do
5      v := Q.dequeue();
6      if v.label == x then
7          return v
8      for w ∈ G.adjacent(v) do
9          if w ∉ visited then
10             visited := visited ∪ {w};
11             Q.enqueue(w)
```

A vertex can be in three possible states.

- Not visited
- Visited and in queue
- Visited and not in queue

Exercise 14.1

How do we maintain the visited set?

Commentary: We are only inserting and checking membership. There is no delete. We may mix ideas from hashing and BST. Bloom filter is a classic technique to store such objects. https://en.wikipedia.org/wiki/Bloom_filter

Example : BFS

Initially: $Q = [entry]$

After dequeuing entry: $Q = [a]$

After dequeuing a: $Q = [f, b]$

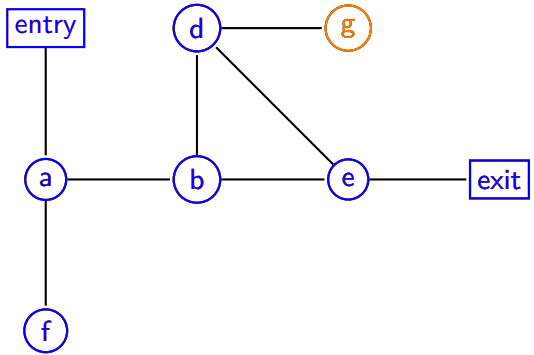
After dequeuing f: $Q = [b]$

After dequeuing b: $Q = [e, d]$

After dequeuing e: $Q = [d, exit]$

After dequeuing d: $Q = [exit, g]$

After dequeuing exit: the return is triggered.



We may not only want to find the exit node but also a path to the exit node.

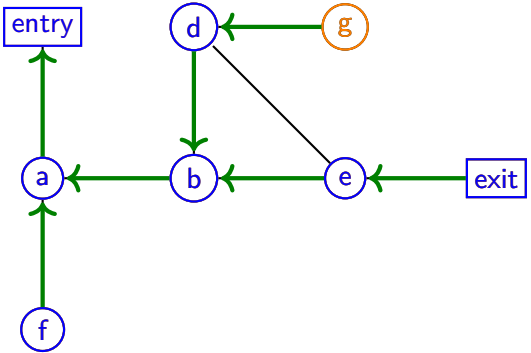
Algorithm: BFS for a path to the found node

Algorithm 14.2: BFS(Graph $G = (V, E)$, vertex r, Value x)

```
1  Queue Q;
2  visited := {r};
3  Q.enqueue(r);
4  while not Q.empty() do
5      v := Q.dequeue();
6      if v.label == x then
7          return v
8      for w ∈ G.adjacent(v) do
9          if w ∉ visited then
10             visited := visited ∪ {w};
11             Q.enqueue(w);
12             w.parent = v
```

Example : BFS with parent relation

Green edges point at the parents.



Topic 14.2

Spanning tree from BFS

Finding a spanning tree

Let us suppose if we want to find a path to every vertex from the source vertex.

We can compute a rooted spanning tree using BFS.

We do not stop at some node and continue to traverse the entire graph.

We also keep track of levels, which allows us to record auxiliary information about the algorithm.

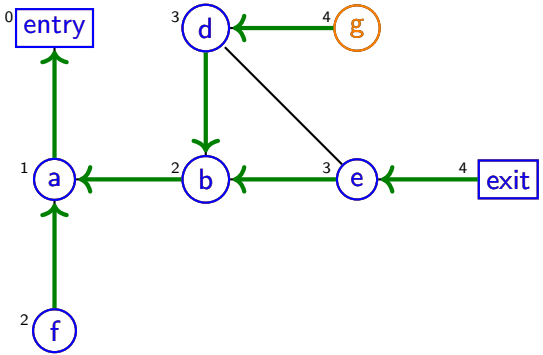
Algorithm: BFS for rooted spanning tree

Algorithm 14.3: BFSSPANNING(Graph $G = (V, E)$, vertex r)

```
1  Queue Q;
2  visited := {r};
3  Q.enqueue(r);
4  r.level := 0;
5  while not Q.empty() do
6      v := Q.dequeue();
7      for w ∈ G.adjacent(v) do
8          if w ∉ visited then
9              visited := visited ∪ {w};
10             Q.enqueue(w);
11             w.parent := v;
12             w.level := v.level + 1
```

Example : Spanning tree from BFS

Superscripts are the level of the vertices.



Topic 14.3

Analysis of BFS

Running time of BFS

- For Each node there is an enqueue and a dequeue. Therefore, $O(|V|)$ queue operations.
- For each node adjacent nodes are enumerated. Therefore, the inner loop will have $O(|E|)$ iterations.

Therefore, the running time is $O(|V| + |E|)$

The pattern in the content of Q

Definition 14.2

Let $Q = v_1, ...v_n$. The level sequence of Q is $v_1.level, ..., v_n.level$.

Theorem 14.1

The level sequence of Q is $k...k\underbrace{(k+1)...(k+1)}_{\text{possibly empty}}$ for some k.

Proof.

We prove it by induction.

Base case:

Initially, Q has level sequence 0.

Induction step:

Let us suppose at a given time the level sequence of Q is $k...k(k+1)...(k+1)$.

We will dequeue the front k-level vertex and possibly enqueue several level-$k+1$ vertices. □

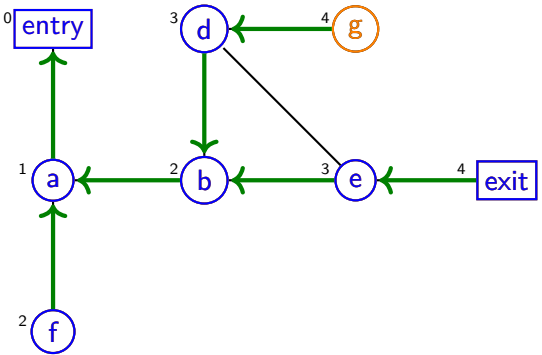
Example: contents of Q

Example 14.1

Let us look at the content of Q in our running example.

The states of Q.

Q	Level sequence of Q
$[entry]$	0
$[a]$	1
$[f, b]$	2,2
$[b]$	2
$[e, d]$	3,3
$[d, exit]$	3,4
$[exit, g]$	4,4



Level difference of adjacent nodes.

Theorem 14.2

For any edge $\{v, v'\} \in E$, $|v.level - v'.level| \leq 1$.

Proof.

Let us suppose v was added to Q before v'.

v' must enter the queue before the end of the iteration that dequeues v. (Why?)

We have two possible cases.

- v' entered Q at the iteration for the dequeue of v: Therefore, $v'.level = v.level + 1$.
- v' entered Q before dequeue of v: Due to the previous theorem, $v'.level$ must be either $v.level + 1$ or $v.level$. (Why?)

□

Commentary: In the last case, v is in Q when v' is entering. Let us suppose the level sequence of $Q = k..k(k+1)..(k+1)$. $v.level$ is either k or $k+1$. $v'.level$ will be set to $k+1$. Therefore, $v'.level$ is either $v.level + 1$ or $v.level$.

BFS finds the shortest path

Theorem 14.3

For each $v \in V$, the path from v to r via the parent field is a path with the shortest length.

Proof.

Since $r.level = 0$, the path from v to r has $v.level$ edges.

Due to the previous theorem, no edge can reduce *level* more than one, the lengths of all paths to r from v cannot be smaller than $v.level$. □

Topic 14.4

Finding connected components

Detect a component

Algorithm 14.4: BFSConnected(Graph $G = (V, E)$, Vertex r, int id)

```
1  Queue Q;
2  visited := {r};
3  Q.enqueue(r);
4  r.component := id;
5  while not Q.empty() do
6      v := Q.dequeue();
7      for w ∈ G.adjacent(v) do
8          if w ∉ visited then
9              visited := visited ∪ {w};
10             Q.enqueue(w);
11             w.component := id
```

Find all connected components

Algorithm 14.5: CC(Graph $G = (V, E)$)

```
1 for v ∈ V do
2     v.component := 0
3 componentId := 1;
4 while r ∈ V such that r.component == 0 do
5     BFSConnected(G, r, componentId);
6     componentId := componentId + 1;
```

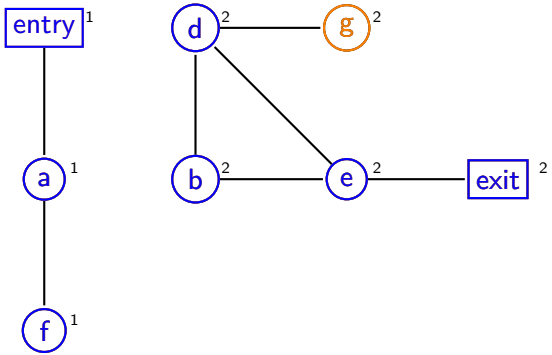
Exercise 14.2

a. What is the cost of evaluating the condition at line 4?

b. What is the running time of the above procedure?

Commentary: We should not evaluate the condition at line 4 from the start. We should try to reuse the previous runs of the condition.

Example: find all connected components



Topic 14.5

Checking bipartite graph

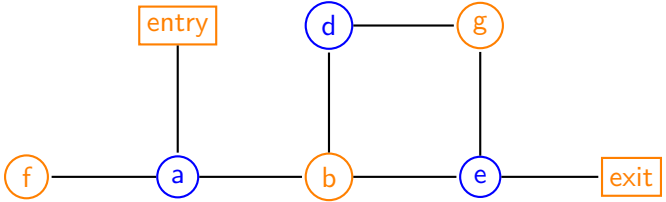
Bipartite graphs

Definition 14.3

A graph $G = (V, E)$ is bipartite if there are V_1 and V_2 such that $V = V_1 \uplus V_2$ and for all $e \in E$, $e \not\subseteq V_1$ and $e \not\subseteq V_2$.

Example 14.2

The following is a bipartite graph, where $V_1 = \{entry, f, b, g, exit\}$ and $V_2 = \{a, d, e\}$.



Theorem 14.4

A bipartite graph does not contain cycles of odd length. Done in tutorial.

Checking Bipartite graph

Algorithm 14.6: IsBipartite(Graph $G = (V, E)$)

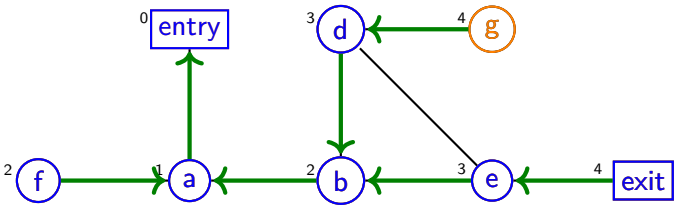
```
1 Assume(graph is connected);
2 Choose r ∈ V;
3 BFSSpanning(G, r);                          // Constructs spanning tree with levels
4 if for each {v, v'} ∈ E  v.level ≠ v'.level then
5     return True;
6 return False;
```

Exercise 14.3

a. Modify the above algorithm to support a not-connected graph.
b. What is the cost of evaluating the condition at line 4?
c. What is the running time of the above procedure?

Example : IsBipartite

We ran BFS on the following graph.

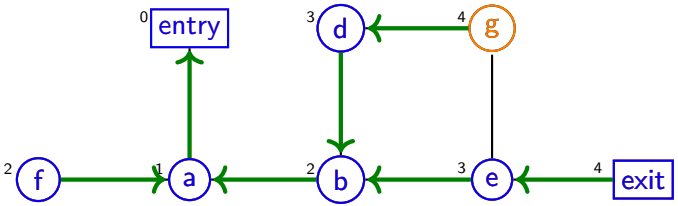


Since $d.level = b.level$, $\{d, e\}$ edge causes the **if** condition to fail.

Therefore, the graph is not bipartite.

Example : another example for IsBipartite

We ran BFS on the following graph.



BFS spanning tree edges will naturally satisfy the **if** condition.

Since $g.level \neq e.level$, the extra edge $\{g, e\}$ also does not cause the **if** condition to fail.

Therefore, the graph is bipartite.

Correctness of IsBipartite

Theorem 14.5
If IsBipartite($G = (V, E)$) returns true, G is bipartite.

Proof.
Let $V_1 = \{v | v.level\%2 = 0\}$ and $V_2 = \{v | v.level\%2 = 1\}$.

Due to the **if** condition in IsBipartite, there are no edges connecting the same level.

Due to theorem 14.2, we only have edges connecting neighboring levels.

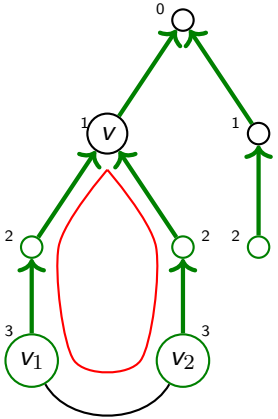
There are no edges that are inside V_1 or V_2. □

Correctness of IsBipartite

Theorem 14.6

If IsBipartite($G = (V, E)$) returns false, G is not bipartite.

Proof.



Since false is returned, there is $\{v_1, v_2\} \in E$ such that $v_1.level = v_2.level$.

Let v be the least common ancestor of v_1 and v_2 in the spanning tree induced by the run of BFS.

The lengths of paths $v_1, ..., v$ and $v_2, ..., v$ are the same. (Why?)

Therefore, path $v_1..v...v_2v_1$ is an odd cycle.

Therefore, G is not bipartite. □

Topic 14.6

Diameter of a graph

Diameter of a graph G

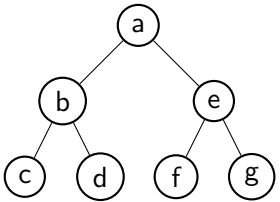
Definition 14.4

For a graph G, let the distance(v,v') be the length of one of the shortest paths between v and v'.

Definition 14.5

For a connected graph $G = (V, R')$, $diameter(G) = max\{distance(v, v') | \{v, v'\} \in E\}$.

Example 14.3



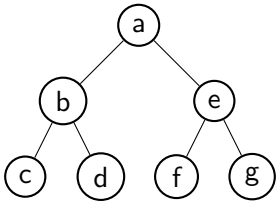
The diameter of the graph is 4.

Can we use BFS for diameter?

Let us run $\mathrm{BFSSPANNING}(G = (V, E), r)$ for some $r \in V$.

Let *maxlevel* be the maximum level assigned to a node in the above graph.

Example 14.4



In the above graph, let $r = a$. The maxlevel is 2.

BFS diameter relation.

Theorem 14.7

Let *maxlevel* be the maximum level assigned to a node in $G = (V, E)$ after running BFS from node r. $maxlevel \leq diameter(G) \leq 2 * maxlevel$

Proof.

Since there are nodes *maxlevel* distance away from r, $maxlevel \leq diameter(G)$.

Let $v_1, v_2 \in V$. $distance(r, v_1) \leq maxlevel$ and $distance(r, v_2) \leq maxlevel$.

Therefore, $distance(r, v_1) + distance(r, v_2) \leq 2 * maxlevel$.

Therefore, $distance(v_1, v_2) \leq 2 * maxlevel$.

Therefore, $diameter(G) \leq 2 * maxlevel$.

□

All runs of BFS

Algorithm 14.7: DIAMETER(Graph $G = (V, E)$)

```
1 ASSUME(graph is connected);
2 maxlevel := 0;
3 for r ∈ V do
4 |   BFSSPANNING(G, r);
5 |   maxlevel' := maximum level assigned to a node;
6 |   maxlevel := max(maxlevel, maxlevel')
7 return maxlevel;
```

Exercise 14.4

Is this the best algorithm for the diameter computation? Ask Search engines, LLMs, etc.

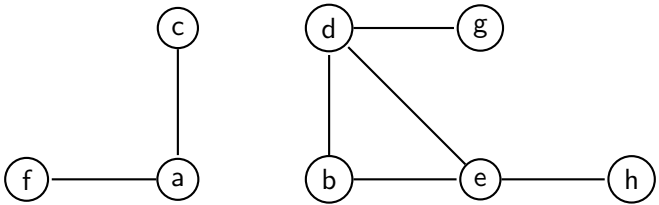
Topic 14.7

Union find for connected components

Example: connected components are disjoint sets

Example 14.5

Consider the following graph.



The connected components are disjoint sets of nodes $\{c, a, f\}$ and $\{d, g, b, e, h\}$.

Connected components via UnionFind

Instead of BFS, we may use UnionFind to find the connected components.

Algorithm 14.8: ConnectedComponents(Graph $G = (V, E)$)

```
UnionFind s;
while v ∈ V do
    s.makeSet(v)
while {v, v'} ∈ E do
    if s.findSet(v) ≠ s.findSet(v') then
        s.union(v.v')
return s;
```

Exercise 14.5

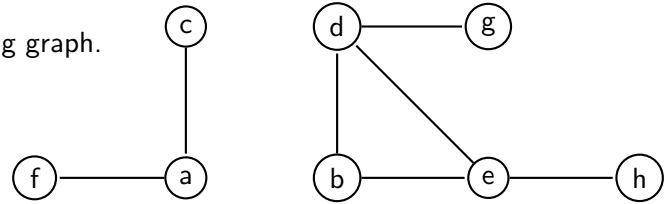
Given the returned s,

- can we efficiently check the number of connected components?
- can we efficiently check if two vertices belong to the same component or not?
- can we efficiently enumerate the vertices of a component?

Example: run UnionFind for connected components

Example 14.6

Consider the following graph.



After processing all nodes: we have sets $\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{c\}$
After processing edge $\{d, b\}$: we have sets $\{a\}$ $\{b, d\}$ $\{c\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$
After processing edge $\{a, f\}$: we have sets $\{a, f\}$ $\{b, d\}$ $\{c\}$ $\{e\}$ $\{g\}$ $\{h\}$
After processing edge $\{e, h\}$: we have sets $\{a, f\}$ $\{b, d\}$ $\{c\}$ $\{e, h\}$ $\{g\}$
After processing edge $\{d, g\}$: we have sets $\{a, f\}$ $\{b, d, g\}$ $\{c\}$ $\{e, h\}$
After processing edge $\{d, g\}$: we have sets $\{a, f\}$ $\{b, d, g\}$ $\{c\}$ $\{e, h\}$
After processing edge $\{a, c\}$: we have sets $\{a, f, c\}$ $\{b, d, g\}$ $\{e, h\}$
After processing edge $\{d, e\}$: we have sets $\{a, f, c\}$ $\{b, d, g, e, h\}$
After processing edge $\{b, e\}$: we have no change.

Recall: breadth-first based connected components

Algorithm 14.9: $\mathrm{CC}(\text{ Graph } G = (V, E)\ )$

for $v \in V$ **do**
    $v.component := 0$

$componentId := 1$;
while $r \in V$ *such that* $r.component == 0$ **do**
    $\mathrm{BFSConnected}(G, r, componentId)$;
    $componentId := componentId + 1$;

In BFS, there is a step that needs to find the next unvisited node after finishing a component.

Therefore, we needed to maintain a set of unvisited nodes.

Exercise 14.6

a. What is the needed interface for the set of unvisited nodes?
b. Can all the needed operations be done within unit cost?

Topic 14.8

Tutorial problems

Exercise: shortest path

Exercise 14.7

There are many variations of BFS to solve various needs. For example, suppose that every edge e=(u,v) also has a weight w(e) (say the width of the road from u to v). Assume that the set of values that w(e) can take is small. For a path p= (v1,v2,...,vk), let the weight w(p) be the minimum of the weights of the edges in the path. We would like to find one of the shortest paths from a vertex s to all vertices v. Can we adapt BFS to detect this path?

Exercise: correctness of BFS

Exercise 14.8

Write an induction proof to show that if vertex r and v are connected in a graph G, then v will be visited in call BFSConnected(Graph $G = (V, E)$, Vertex r, int id).

Exercise: Graph with two kinds of edges

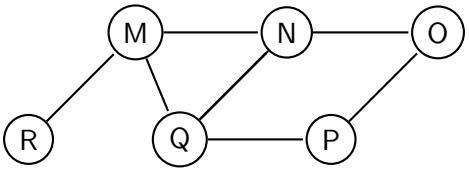
Exercise 14.9

Suppose that there is an undirected graph G(V,E) where the edges are colored either red or blue. Given two vertices u and v. It is desired to (i) find the shortest path irrespective of color, (ii) find the shortest path, and of these paths, the one with the fewest red edges, (iii) a path with the fewest red edges. Draw an example where the above three paths are distinct. Clearly, to solve (i), BFS is the answer. How will you design algorithms for (ii) and (iii)?

Exercise: order of traversal (quiz 23)

Exercise 14.10

Is MNRQPO a possible BFS traversal for the following graph?



Topic 14.9

Problems

True or False

Exercise 14.11

Mark the following statements True / False and also provide justification.

1. Variable Q in our BFS always contains nodes from same level.

Exercise: representation and BFS (quiz 23)

Exercise 14.12

Compute the running time of breadth-first search on a tree with n edges in the cases when the tree is represented by

- an adjacency list or
- an adjacency matrix.

End of Lecture 14