CS213/293 Data Structure and Algorithms 2024

Lecture 13: Graphs - Breadth-first search

Instructor: Ashutosh Gupta

IITB India

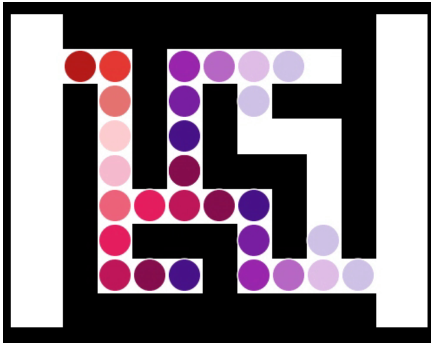
Compile date: 2024-10-13

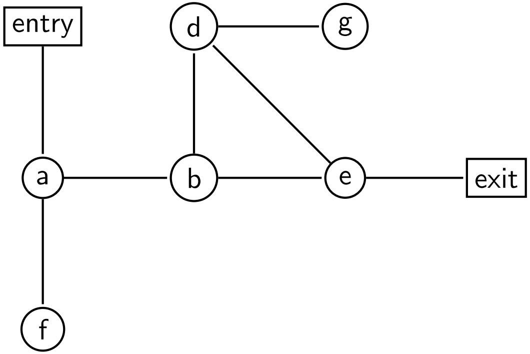
Topic 13.1

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 13.1

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 lower level first.

Algorithm: BFS for search

Algorithm 13.1: BFS(Graph $G = (V, E)$, vertex r, Value x)

```
Queue Q;
set $visited := \{r\}$;
$Q.enqueue(r)$;
while not $Q.empty()$ do
    $v := Q.dequeue()$;
    if $v.label == x$ then
        return $v$
    for $w \in G.adjacent(v)$ do
        if $w \notin visited$ then
            $visited := visited \cup \{w\}$;
            $Q.enqueue(w)$
```

A vertex can be in three possible states.

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

Exercise 13.1

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 visiting entry: $Q = [a]$

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

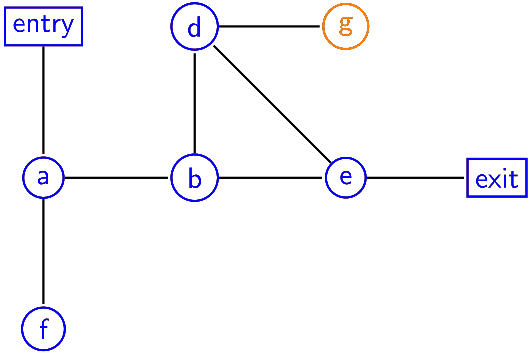
After visiting f: $Q = [b]$

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

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

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

After visiting exit: the return is triggered.



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

Algorithm: BFS for a path to the found node

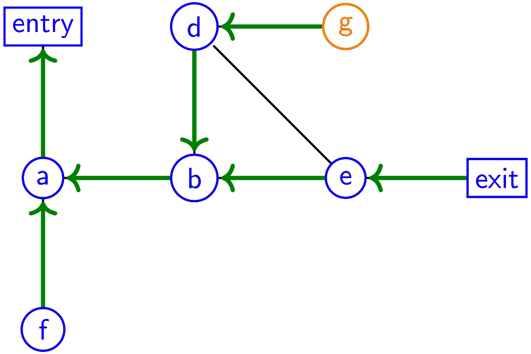
Algorithm 13.2: BFS(Graph $G = (V, E)$, vertex r, Value x)

```
Queue Q;
$visited := \{r\}$;
$Q.enqueue(r)$;
while not $Q.empty()$ do
    $v := Q.dequeue()$;
    if $v.label == x$ then
        return $v$
    for $w \in G.adjacent(v)$ do
        if $w \notin visited$ then
            $visited := visited \cup \{w\}$;
            $Q.enqueue(w)$;
            $w.parent = v$
```

To get the path, keep track of parent of each node and finally after return using these parents in recursion you'll have the path to the target node.

Example : BFS with parent relation

Green edges point at the parents.



Algorithm: BFS for rooted spanning tree

We do not stop at some node but traverse the entire graph.

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

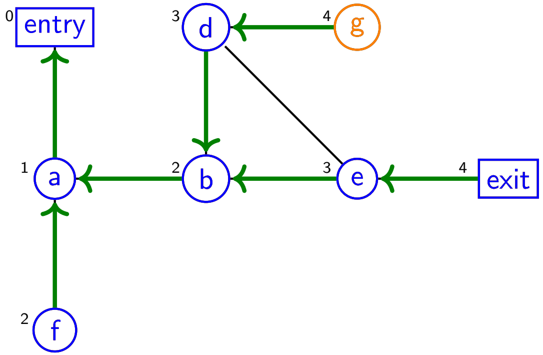
Algorithm 13.3: BFSSPANNING(Graph $G = (V, E)$, vertex r)

```
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 $w \in G.adjacent(v)$ do
8          if $w \notin visited$ then
9              $visited := visited \cup \{w\}$;
10             $Q.enqueue(w)$;
11             $w.parent := v$;
12             $w.level := v.level + 1$
```

make a struct and store all the information like level, label and parent in the node.

Example : Spanning tree from BFS

Superscripts are the level of the vertices.



Topic 13.2

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

Theorem 13.1

Q has vertices with level sequence $k...k\underbrace{(k+1)...(k+1)}_{\textit{possibly empty}}$ for some k.

Proof.

We prove it by induction.

definitely it will have elements in increasing order as at an instance if it has only k-level elements then we'll dequeue it and enqueue k+1 level elements only.

Base case:

Initially, Q has level sequence 0.

Induction step:

Let us suppose at a given time the level sequence is $k...k\underbrace{(k+1)...(k+1)}$.

We will dequeue a k-level vertex and possibly enqueue several level-$k+1$ vertices. Hence proved. □

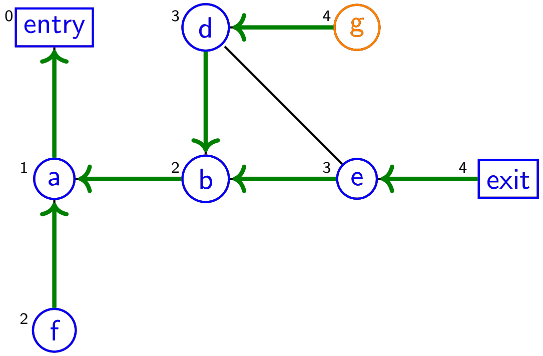
Example: contents of Q

Example 13.1

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

The states of Q.

Q	Level sequence in 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 13.2

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

It is super trivial and intuitive.

Proof.

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

Due to the previous theorem, the vertices will enter Q with increasing levels.

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. $v'.level$ must be either $v.level + 1$ or $v.level$. (Why?)

$\square$

Commentary: In the last case, v is in Q when v' is entering, therefore $v'.level$ cannot enter Q with a value greater than $v.level + 1$.

BFS finds the shortest path

v to r is the shortest path in bfs using parent field

Theorem 13.3

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 $v.level$. $\square$

Topic 13.3

Finding connected components

Detect a component

Algorithm 13.4: BFSCONNECTED(Graph $G = (V, E)$, Vertex r, int id)

```
Queue Q;
$visited := \{r\}$;
$Q.enqueue(r)$;
$r.component := id$;
while not $Q.empty()$ do
    $v := Q.dequeue()$;
    for $w \in G.adjacent(v)$ do
        if $w \notin visited$ then
            $visited := visited \cup \{w\}$;
            $Q.enqueue(w)$;
            $w.component := id$
```

Find all connected components

```
Algorithm 13.5: CC( Graph $G = (V, E)$ )
1 for $v \in V$ do
2     $v.component := 0$
3 $componentId := 1$;
4 while $r \in V$ such that $r.component == 0$ do
5     BFSConnected($G, r, componentId$);
6     $componentId := componentId + 1$;
```

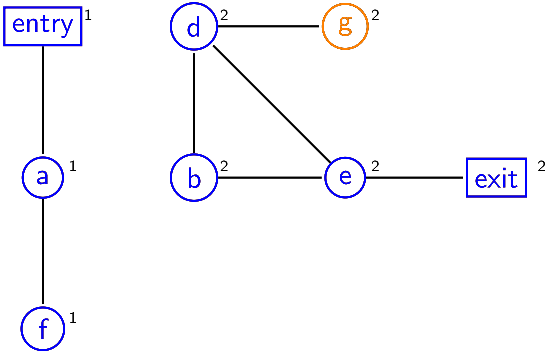
Exercise 13.2

a. *What is the cost of evaluating the condition at line 4?* O(V)

b. *What is the running time of the above procedure?* O(V+E)

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 13.4

Checking bipartite graph

Bipartite graphs

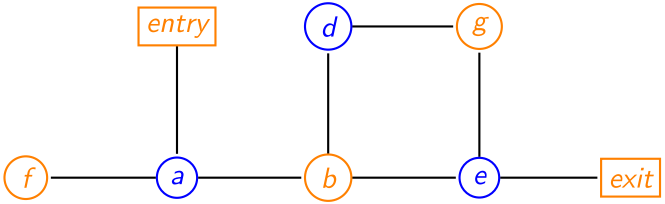
Definition 13.2

A graph is bipartite iff you can partition the graph node's into two parts such that all nodes are covered and two parts make a bipartite graph.

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 13.2

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



Theorem 13.4

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

Checking Bipartite graph

```
Algorithm 13.6: IsBipartite( Graph $G = (V, E)$ )
1 Assume(graph is connected);
2 Choose $r \in V$;
3 BFSSpanning($G, r$);
4 if  for each $\{v, v'\} \in E$  $v.level \neq v'.level$ then
5     return True;
6 return False;
```

Exercise 13.3

Run a loop checking if r is visited or not, based call Span

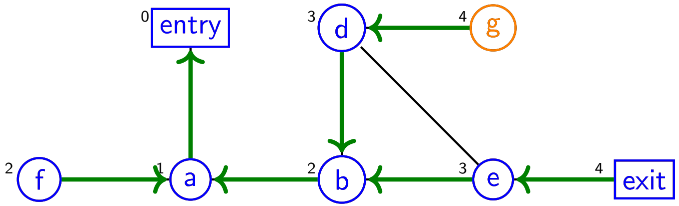
a. *Modify the above algorithm to support a not-connected graph.*

b. *What is the cost of evaluating the condition at line 4?* O(|E|)

c. *What is the running time of the above procedure?* Overall, it would converge to O(V+E)

Example : IsBipartite

We ran BFS on the following graph.

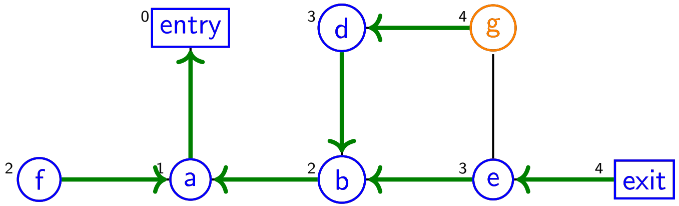


Since $d.level = b.level$, $\{d, e\}$ edge causes the if condition to fail.

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 13.5

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 condition in IsBipartite, there are no edges connecting the same level.

Due to theorem 13.2, we only have edges connecting neighboring levels.

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

Correctness of IsBipartite

Theorem 13.6
If IsBipartite$(G = (V, E))$ *returns false, G is not bipartite.*

Proof.
Since false is returned, there exists $\{v_1, v_2\} \in E$ 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?)

There is an edge between same level hence from ancestor two paths of length 'l' each and one edge in same level makes it (2*l + 1)

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

Therefore, G is not bipartite. □

Topic 13.5

Diameter of a graph

Diameter of a graph G

Definition 13.3

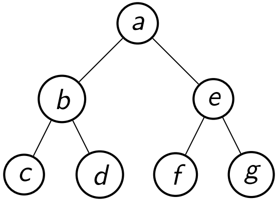
For a graph G, let the distance(v,v') be the length of one of the shortest paths between v and v'.

Definition 13.4

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

Diameter is only defined for a connected graph.

Example 13.3



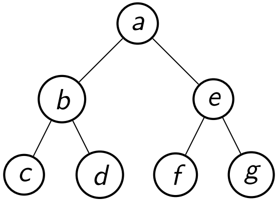
The diameter of the above graph is 4.

Can we use BFS for diameter?

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

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

Example 13.4



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

BFS diameter relation.

Theorem 13.7

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

We just proved that distance between any two nodes is lesser than 2*maxlevel.

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

□

All runs of BFS

Algorithm 13.7: DIAMETER(Graph $G = (V, E)$)

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

Exercise 13.4

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

We can use two bfs for this, because first bfs run find out the max distant node for any given node, then from that given node find the maximum distant node and the distance between them is diameter. This works because the farthest distance from any arbitrary node lies on the longest path in the graph, and starting BFS from this farthest node yields the diameter.

Topic 13.6

Tutorial problems

Exercise: shortest path

Exercise 13.5

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 13.6

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 $\textsc{BFSConnected}$*(Graph $G = (V, E)$, Vertex r, int id).*

Exercise: Graph with two kinds of edges

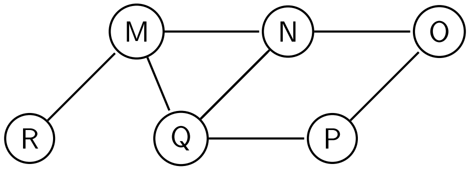
Exercise 13.7

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 13.8

Is MNRQPO a possible BFS traversal for the following graph?



Topic 13.7

Problems

Exercise: representation and BFS (quiz 23)

Exercise 13.9

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 13