Last Time

- Introduced Graphs
Today

- Traversing a Graph
- A shortest path algorithm
Example Graph

We will use this graph as an example throughout today’s class.
Graph Operations

- for $v \in V$ – iterate over all vertices
- for $e \in E$ – iterate over all edges
- for $e \in \text{edges}(v)$ – iterate over all edges with $v$ as an endpoint
- for $v_j \in \text{adjacent}(v_i)$ – iterate over all vertices that are adjacent to $v_i$
Traversal

Reachability

Input: a directed or undirected graph $G = (V, E)$, and a source node $s$.
Output: the set of nodes in $G$ reachable from $s$

Search($G, s$)

1. $R = \{s\}$
2. while there is an edge $e$ from $R$ to $V-R$ do
   1. let $e = (u, v)$
   2. $R = R \cup \{v\}$
   3. $\text{parent}[v] = u$
3. end while
4. return $R$
Given a Graph $G = (V,E)$.
At any time during a search algorithm we can partition the set of vertices into three sets.

- **$R$** – the set of visited nodes
- **$V-R$** – the set of nodes that haven’t been visited yet
- **active** or **fringe** vertices – those nodes that have edges from $R$ to $V-R$

Choosing which active node to expand, and which edge to follow, differentiates different search algorithms.

The book colors these sets of nodes do differentiate them.

- processed nodes - black
- active nodes - grey
- unreached nodes - white
Generic Search Algorithms
Depth-First Search – select the active node for processing by selecting the “most-recent” node first. Use a Stack.
**Depth-First Search** – select the active node for processing by selecting the “most-recent” node first. Use a Stack.

**DFS(G,s)**

```plaintext
for u ∈ V do
    mark[u] = 0; parent[u] = ∅
end for
Recursive-DFS(s)
```

**Recursive-DFS(u)**

```plaintext
mark[u] = 1
for v ∈ adjacent(u) do
    if mark[v] = 1 then
        parent[v] = u;
        DFS(v)
    end if
end for
```
Depth-First Search runtime
Depth-First Search runtime

DFS runtime

Initialization Runtime?
Recursion Runtime?
Depth-First Search runtime

Initialization Runtime?
Recursion Runtime?

Initialization = O(V)
Recursion = O(E)
Total DFS Runtime = O(V+E)
Breadth-First Search – select the active node for processing by selecting the “earliest” node first. Use a Queue.
Breadth-First Search – select the active node for processing by selecting the “earliest” node first. Use a Queue.

\[
\text{BFS}(G, s)
\]

\[
\text{for } v \in V - \{s\} \text{ do} \\
\quad \text{parent}[v] = \emptyset; \text{mark}[v] = 0; \text{d}[v] = 0 \\
\text{end for} \\
\text{parent}[s] = \emptyset; \text{mark}[s] = 1; \text{d}[s] = 0 \\
\text{Q} = \{s\} \\
\text{while } Q \neq \emptyset \text{ do} \\
\quad u = \text{Dequeue}(Q) \\
\quad \text{for } v \in \text{adjacent}(u) \text{ do} \\
\quad\quad \text{if } \text{mark}[v] = 0 \text{ then} \\
\quad\quad\quad \text{mark}[v] = 1; \text{parent}[v] = u; \text{d}[v] = \text{d}[u] + 1; \\
\quad\quad\quad \text{Enqueue}(Q, v) \\
\quad\quad \text{end if} \\
\quad \text{end for} \\
\text{end while}
Runtime of Breadth-First-Search

BFS(G,s)

\[\textbf{for } v \in V-\{s\} \textbf{ do}\]
\[\text{parent}[v] = \emptyset; \text{mark}[v] = 0; d[v] = 0\]
\[\textbf{end for}\]
\[\text{parent}[s] = \emptyset; \text{mark}[s] = 1; d[s] = 0\]
\[Q = \{s\}\]
\[\textbf{while } Q \neq \emptyset \textbf{ do}\]
\[u = \text{Dequeue}(Q)\]
\[\textbf{for } v \in \text{adjacent}(u) \textbf{ do}\]
\[\textbf{if } \text{mark}[v] = 0 \textbf{ then}\]
\[\text{mark}[v] = 1; \text{parent}[v] = u; d[v] = d[u] + 1;\]
\[\text{Enqueue}(Q,v)\]
\[\textbf{end if}\]
\[\textbf{end for}\]
\[\textbf{end while}\]
Runtime of Breadth-First-Search

BFS(G,s)

\[
\begin{align*}
\text{for } v \in V-\{s\} \text{ do} \\
\quad \text{parent}[v] = \emptyset; \text{mark}[v] = 0; \text{d}[v] = 0 \\
\text{end for} \\
\text{parent}[s] = \emptyset; \text{mark}[s] = 1; \text{d}[s] = 0 \\
Q = \{s\} \\
\text{while } Q \neq \emptyset \text{ do} \\
\quad u = \text{Dequeue}(Q) \\
\quad \text{for } v \in \text{adjacent}(u) \text{ do} \\
\quad \quad \text{if } \text{mark}[v] = 0 \text{ then} \\
\quad \quad \quad \text{mark}[v] = 1; \text{parent}[v] = u; \text{d}[v] = \text{d}[u] + 1; \\
\quad \quad \quad \text{Enqueue}(Q,v) \\
\quad \quad \text{end if} \\
\quad \text{end for} \\
\text{end while} \\
\end{align*}
\]

\(O(V+E)\)
DFS and BFS create **Search Trees**

Prove that DFS and BFS construct **Directed Acyclic Graphs**. Show that

- The resulting graph is a directed Graph
- The search traversal cannot have cycles

These are **Spanning Trees** or **Spanning Forests** if there are multiple connected components.
Define distance\((u,v)\) as the minimum length (in edges) of a path from \(u\) to \(v\).

Claim: \(d[v] = \text{distance}(s,v)\) for all nodes \(v \in V\)

Proof: Need to show that

- \(d[v] \geq \text{distance}(s,v)\)
- \(d[v] \leq \text{distance}(s,v)\)
**Proof:** Case 1) \( d[v] \geq \text{distance}(s,v) \)

A path with length \( d[v] \) can be reconstructed by traversing the parents of \( v \) until \( s \) is reached. Since \( \text{distance}(s,v) \) is the length of the minimum path, \( d[v] \) must be at least as large as \( \text{distance}(s,v) \). This logic can be demonstrated with greater rigor using induction.

- Base Case: \( d[s] \geq \text{distance}(s,s) \).
- Inductive step: for some vertex \( u \) adjacent to a fringe node \( v \).
  - \( d[v] = d[u] + 1 \)
  - \( d[v] \geq \text{distance}(s,u) + 1 \)
  - \( d[v] \geq \text{distance}(s,v) \)
**Proof:** Case 2) \(d[v] \leq \text{distance}(s,v)\)

Assume not. Assume \(d(v) > \text{distance}(s,v)\). Show that this can never be true. Let \(u\) be the node immediately preceding \(v\) on the shortest path from \(s\) to \(v\)

\[
d[v] > \text{distance}(s,v) = \text{distance}(s,u) + 1 = d[u] + 1
\]

Consider what happens when \(u\) is dequeued from \(Q\).

1. \(v\) was **unvisited** – Then \(d[v]\) is set to \(d[u] + 1\). Contradiction
2. \(v\) was visited – Then \(v\) was removed from \(Q\) earlier, and \(d[v] < d[u]\). Contradiction.
3. \(v\) was a fringe node, in the queue – Then it was enqueued by \(w\), **before** \(u\) where \(d[w] \leq d[u]\), and \(d[v]\) was set to \(d[w] + 1\).

Thus \(d[v] = d[w] + 1 \leq d[u] + 1\). Contradiction

Therefore, \(d[v] \leq \text{distance}(s,v)\)

- Since \(d[v] \leq \text{distance}(s,v)\) and \(d[v] \geq \text{distance}(s,v)\), \(d[v] = \text{distance}(s,v)\)
We can use the same structure of the BFS to calculate the shortest path between two points in an undirected graph. Now distance\((u,v)\) is \(\sum\) weights of edges on the shortest path from \(u\) to \(v\). Rather than expanding the nodes in order in a queue. We will expand the closest node first.
Example of Dijkstra’s Algorithm

Find the minimum length from source $s$ to any node $v \in V$. 

![Diagram of Dijkstra's Algorithm](image)
Find the minimum length from source $s$ to any node $v \in V$. 

Example of Dijkstra’s Algorithm
Example of Dijkstra’s Algorithm

Find the minimum length from source s to any node \( v \in V \).
Example of Dijkstra’s Algorithm

Find the minimum length from source $s$ to any node $v \in V$. 

```
a:1  
```

```
s:0  
```

```
c:2  
```

```
d:1  
```

```
e:3  
```

```
f:2  
```

```
h:∞  
```

```
i:3  
```

```
j:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```

```
m:∞  
```

```
f:2  
```

```
h:∞  
```

```
l:∞  
```
Find the minimum length from source $s$ to any node $v \in V$. 
Find the minimum length from source $s$ to any node $v \in V$. 
Find the minimum length from source $s$ to any node $v \in V$. 
Find the minimum length from source $s$ to any node $v \in V$. 

\[
\begin{array}{cccccc}
s : 0 & & & & & \\
a : 1 & 2 & e : 3 & 3 & h : 4 & \\
c : 2 & & & & & \\
d : 1 & 1 & f : 2 & 3 & j : 5 & \\
i : 3 & 1 & & 3 & & \\
\end{array}
\]
Example of Dijkstra’s Algorithm

Find the minimum length from source $s$ to any node $v \in V$. 

![Graph diagram with labels and edges]
Example of Dijkstra’s Algorithm

Find the minimum length from source $s$ to any node $v \in V$. 

```
a:1  s:0  c:2  d:1  e:3  f:2  h:4  i:3  j:5  
1     2  1    2    2  2    2  1
1     3    1  3    3    3
1     2    3  1
```
Dijkstra's Algorithm pseudocode

\[
\text{Dijkstra}(G,s)
\]

\[
\begin{align*}
\text{for } \forall v \in V \setminus \{s\} & \text{ do} \\
& \quad \text{mark}[v] = 0; \text{d}[v] = \infty \\
\text{end for} \\
\text{d}[s] = 0 \\
\text{MinHeap } H = \{s\} \\
\text{while } H \neq \emptyset \text{ do} \\
& \quad u = \text{ExtractMin}(H) \\
& \quad \text{mark}[u] = 1 \\
& \quad \text{for } e \in \text{edges}(u) \text{ do} \\
& \quad \quad (u,v) = e \\
& \quad \quad \text{if } \text{mark}[v] = 0 \text{ then} \\
& \quad \quad \quad \text{if } \text{d}[u] + \text{weight}[e] < \text{d}[v] \text{ then} \\
& \quad \quad \quad \quad \text{d}[v] = \text{d}[u] + \text{weight}[e]; \\
& \quad \quad \quad \quad \text{parent}[v] = u \\
& \quad \quad \quad \text{end if} \\
& \quad \quad \text{Insert}(H,v) \\
& \quad \text{end if} \\
& \quad \text{end for} \\
\text{end while}
\]
The Proof of Dijkstra’s algorithm has the same structure as the Proof that $d[v] = \text{distance}(s,v)$. Need to Show

- $d[v] \geq \text{ShortestPath}(s, v)$
- $d[v] \leq \text{ShortestPath}(s, v)$

Rather than incrementing by single values, you increment by edge weights.
Define Connected Components

All of the nodes within a connected component are **reachable** from every other node in the connected component.

Connected\((u,v)\) if there exists a path from \(u\) to \(v\)

- Reflexive: \(\text{Connected}(u,u)\)
- Symmetric: \(\text{Connected}(u,v) = \text{Connected}(v,u)\)
- Transitive: \(\text{Connected}(u,v)\) and \(\text{Connected}(v,w)\) then \(\text{Connected}(u,w)\)
Identify Connected Components

Components(G)

numComps = 0
for v ∈ V do
    mark[v] = 0; parent[v] = ∅
end for
for v ∈ V do
    if mark[v] = 0 then
        numComps = numComps+1
        DFSC(v,numComps)
    end if
end for

DFSC(v, c)

mark[v] = 1; comp[v] = c
for u ∈ adjacent(v) do
    if mark[u] = 0 then
        parent[u] = v
        DFSC(u,c)
    end if
end for
To detect cycles using DFS.

- Construct a DFS spanning Tree.
- If there are any **back edges** in the Graph, it contains a cycle
  - A **back edge** connects a node at some depth $d$ (in the DFS tree), to a node at some depth $d' < d$. 
Cycle detection using DFS
Cycle detection using DFS
Cycle detection using DFS

New Graph
Cycle detection using DFS
Cycle detection using DFS
Back edge from $f$ to $b$ indicates the presence of a cycle.
Homework 9 is Posted.
- Due in 1 week 11/19

Next time (11/17)
- Strongly Connected Components.
- Greedy Algorithms for finding minimum spanning trees
  - Kruskal’s
  - Prim’s