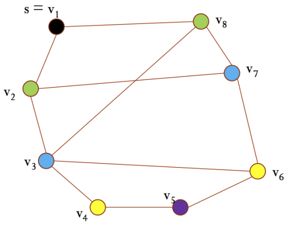
CSE101: Design and Analysis of Algorithms

Ragesh Jaiswal, CSE, UCSD

Graph Algorithms

Breadth First Search (BFS)

BFS(G, s)
- $Layer(0) = \{s\}$
- $i \leftarrow 1$
- While(true)
  - Visit all new nodes that have an edge to a vertex in $Layer(i-1)$
  - Put these nodes in the set $Layer(i)$
  - If $Layer(i)$ is empty, then end
  - $i \leftarrow i+1$



- Theorem 1: The shortest path from s to any vertex in $Layer(i)$ is equal to i.

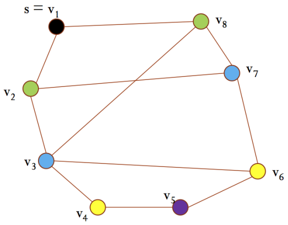
- Theorem 1: The shortest path from s to any vertex in $Layer(i)$ is equal to i.

Proof sketch

- We will prove by induction. Let $P(i)$ denote the statement: *The shortest path from s to any vertex in $Layer(i)$ is equal to i.*
- We will prove that $P(i)$ is true for all i using induction.
- Base case: $P(0)$ is true since $Layer(0)$ contains s.
- Inductive step: Assume $P(0), ..., P(k)$ are true. We will show that $P(k+1)$ is true.
  - Assume for the sake of contradiction that $P(k+1)$ is not true.
  - This implies that there is a vertex v in $Layer(k+1)$ such that the shortest path length from s to v is $< k+1$ (the case $> k+1$ is skipped for class discussion)
  - Consider such a path from s to v. Let u be the vertex in this path just before v.
  - Claim 1: u is contained in $Layer(k)$.
  - This gives us a contradiction since by induction hypothesis, the shortest path length from s to u is k.

Breadth First Search (BFS)

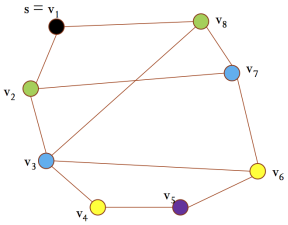
BFS(G, s)
- $Layer(0) = \{s\}$
- $i \leftarrow 1$
- While(true)
  - Visit all new nodes that have an edge to a vertex in $Layer(i-1)$
  - Put these nodes in the set $Layer(i)$
  - If $Layer(i)$ is empty, then end
  - $i \leftarrow i+1$



- What is the running time of BFS given that the graph is given in adjacency list representation?

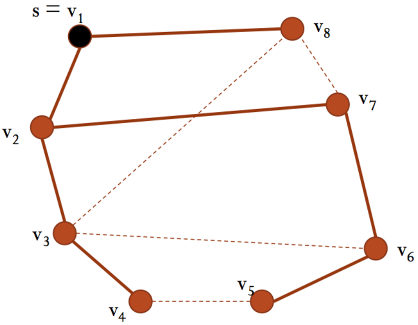
Breadth First Search (BFS)

BFS(G, s)
- $Layer(0) = \{s\}$
- $i \leftarrow 1$
- While(true)
  - Visit all new nodes that have an edge to a vertex in $Layer(i-1)$
  - Put these nodes in the set $Layer(i)$
  - If $Layer(i)$ is empty, then end
  - $i \leftarrow i+1$



- What is the running time of BFS given that the graph is given in adjacency list representation? $O(n+m)$

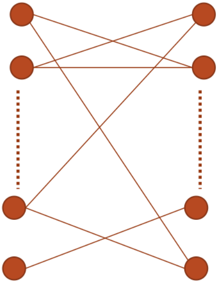
- The BFS algorithm defines the following BFS tree rooted at s
  - Vertex u is the parent of vertex v if u caused the immediate discovery of v.



- Bipartite graph: A graph is *bipartite* iff the vertices can be partitioned into two sets such that there is no edge between any pair of vertices in the same set.

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.



Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

- Consider BFS below
- Is it possible that there is an edge between vertices which belong to sets $Layer(i)$ and $Layer(j)$ such that $j - 1 > i$?

Breadth First Search (BFS)

BFS(G, s)
- $Layer(0) = \{s\}$
- $i \leftarrow 1$
- While(true)
  - Visit all new nodes that have an edge to a vertex in $Layer(i - 1)$
  - Put these nodes in the set $Layer(i)$
  - If $Layer(i)$ is empty, then end
  - $i \leftarrow i + 1$

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

- Consider BFS below
- Is it possible that there is an edge between vertices which belong to sets $Layer(i)$ and $Layer(j)$ such that $j - 1 > i$? No.

Breadth First Search (BFS)

BFS(G, s)
- $Layer(0) = \{s\}$
- $i \leftarrow 1$
- While(true)
  - Visit all new nodes that have an edge to a vertex in $Layer(i - 1)$
  - Put these nodes in the set $Layer(i)$
  - If $Layer(i)$ is empty, then end
  - $i \leftarrow i + 1$

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

- Is it possible that there is an edge between vertices which belong to sets $Layer(i)$ and $Layer(j)$ such that $j - 1 > i$? No.
- Suppose the given graph contains a cycle of odd length. Can this graph be bipartite?

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

- Is it possible that there is an edge between vertices which belong to sets $Layer(i)$ and $Layer(j)$ such that $j - 1 > i$? No.
- Suppose the given graph contains a cycle of odd length. Can this graph be bipartite? No.
  - For sake of contradiction assume that the graph is bipartite.
  - Consider a cycle of odd length with nodes numbered $v_1, v_2, ..., v_{2k+1}$.
  - Since the graph is bipartite the nodes may be partitioned into two sets X and Y s.t. there does not exist en edge between nodes in the same partition.
  - If node v_1 is in X, then v_2 has to be in Y, and node v_3 has to be in X and so on. So, node v_{2k+1} has to be in X. But then there is a edge between v_1 and v_{2k+1}.

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

- Is it possible that there is an edge between vertices which belong to sets $Layer(i)$ and $Layer(j)$ such that $j - 1 > i$? No.
- Suppose the given graph contains a cycle of odd length. Can this graph be bipartite? No.
- Can you now use BFS to check if the graph is bipartite?

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

- Is it possible that there is an edge between vertices which belong to sets $Layer(i)$ and $Layer(j)$ such that $j - 1 > i$? No.
- Suppose the given graph contains a cycle of odd length. Can this graph be bipartite? No.
- Can you now use BFS to check if the graph is bipartite?

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- <u>Claim 1</u>: Any given graph G is bipartite if and only if `IsBipartite(`G`)` outputs "yes".

Algorithm

`IsBipartite(`G`)`
- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- Claim 1: Any given graph G is bipartite if and only if `IsBipartite(`G`)` outputs "yes".

Proof sketch of Claim 1

- Claim 1.1: If `IsBipartite(`G`)` outputs "no", then G is not bipartite.

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- Claim 1: Any given graph G is bipartite if and only if `IsBipartite(`G`)` outputs "yes".

Proof sketch of Claim 1

- Claim 1.1: If `IsBipartite(`G`)` outputs "no", then G is not bipartite.
  - Since there is an odd cycle in G.

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- Claim 1: Any given graph G is bipartite if and only if `IsBipartite(`G`)` outputs "yes".

Proof sketch of Claim 1

- Claim 1.1: If `IsBipartite(`G`)` outputs "no", then G is not bipartite.
  - Since there is an odd cycle in G.
- Claim 1.2: If `IsBipartite(`G`)` outputs "yes", then G is bipartite.

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- Claim 1: Any given graph G is bipartite if and only if `IsBipartite(`G`)` outputs "yes".

Proof sketch of Claim 1

- Claim 1.1: If `IsBipartite(`G`)` outputs "no", then G is not bipartite.
  - Since there is an odd cycle in G.
- Claim 1.2: If `IsBipartite(`G`)` outputs "yes", then G is bipartite.
  - Since the odd and the even layers forms the two partitions of a bipartite graph.

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- What is the running time of the above algorithm?

Algorithm

`IsBipartite(G)`
- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- What is the running time of the above algorithm? $O(n+m)$
  - While running the BFS algorithm, we maintain an array A such that the i^{th} entry of the array stores the layer to which the i^{th} vertex belongs to as per the BFS execution. Note that maintaining such an array while running BFS will only cost $O(1)$ time per vertex. So the total time of running BFS and constructing the array A would be $O(n+m)$.
  - Now, we need to go thorough all edges in the graph and for an edge (i, j), check if $A[i] = A[j]$. This would take a total of $O(m)$ time.
  - So the total running time of the algorithm will be $O(n+m)$.

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

Algorithm

`IsBipartite(`G`)`

- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

- What if G is not a strongly connected graph?

Problem

Given a graph $G = (V, E)$, check if the graph is bipartite.

Algorithm (for strongly connected graphs)

`IsBipartite(`G`)`
- Run BFS and check if two vertices in the same layer has an edge between them
- If yes then output("no") else output("yes")

Algorithm (for any graph)

`IsBipartite(`G`)`
- Let R contain all vertices of G
- While R is not empty
  - Let s be an arbitrary vertex in R
  - Run $BFS(G, s)$ and check if two vertices in the same layer have an edge between them
  - If yes then output("no")
  - Remove all vertices from R that were explored while running $BFS(G, s)$
- Output("yes")

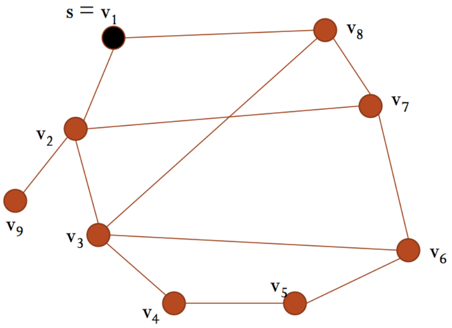
Depth First Search (DFS)

`DFS(`s`)`

- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(`v`)`

Depth First Search (DFS)

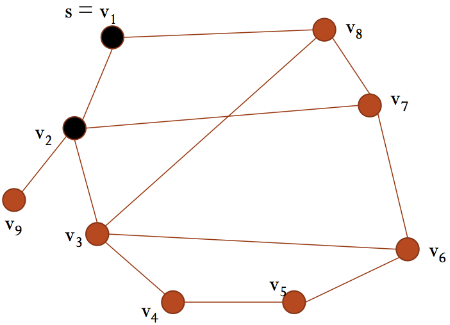
`DFS(s)`
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(v)`



Depth First Search (DFS)

`DFS(`s`)`

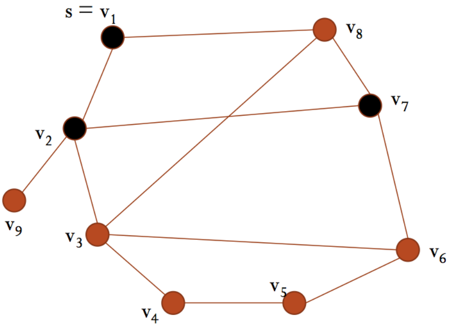
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(`v`)`



Depth First Search (DFS)

`DFS(`s`)`

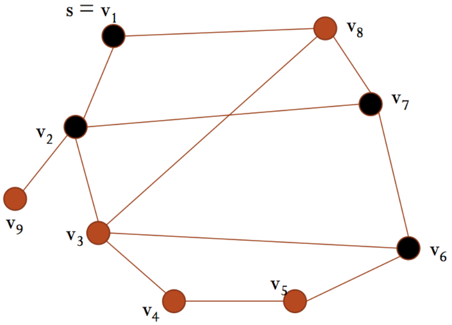
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(`v`)`



Depth First Search (DFS)

DFS(s)

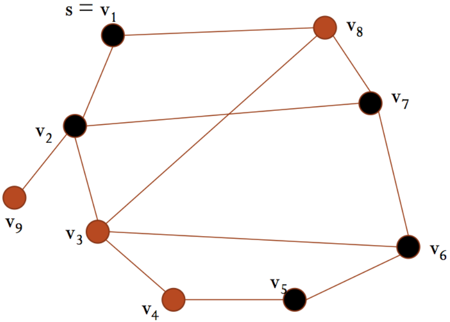
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call DFS(v)



Depth First Search (DFS)

`DFS(s)`

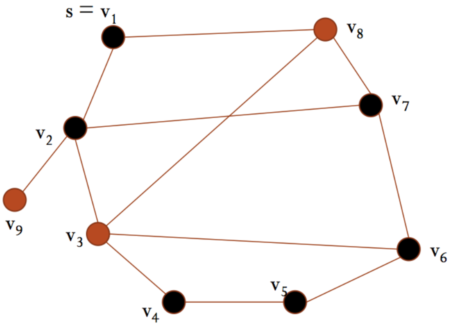
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(v)`



Depth First Search (DFS)

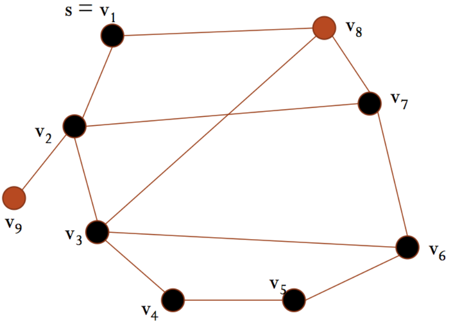
`DFS(s)`

- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(v)`



Depth First Search (DFS)

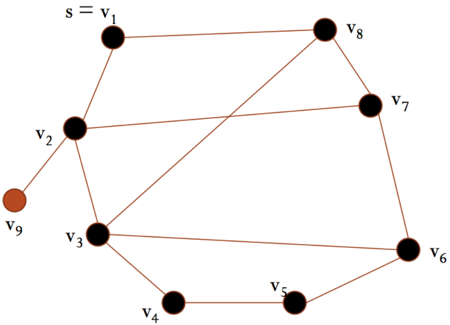
`DFS(s)`
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(v)`



Depth First Search (DFS)

`DFS(s)`

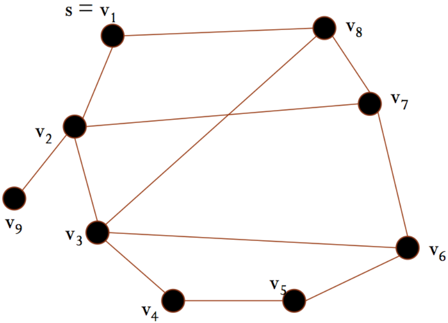
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(v)`



Depth First Search (DFS)

`DFS(s)`

- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(v)`



Depth First Search (DFS)

`DFS`(s)
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS`(v)

- What is the running time of DFS?

Depth First Search (DFS)

`DFS(`s`)`

- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS(`v`)`

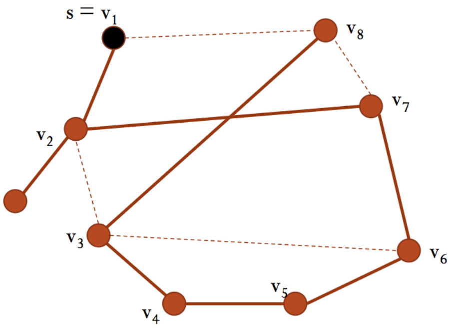
- What is the running time of DFS? $O(n + m)$

Depth First Search (DFS)

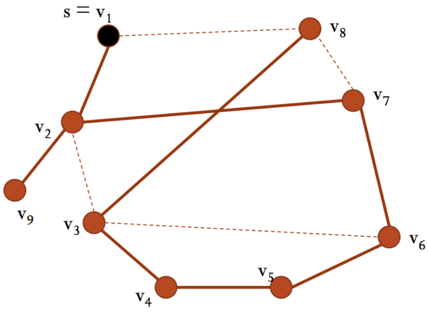
`DFS`(s)
- Mark s as explored
- For each unexplored neighbour v of s
  - Recursively call `DFS`(v)

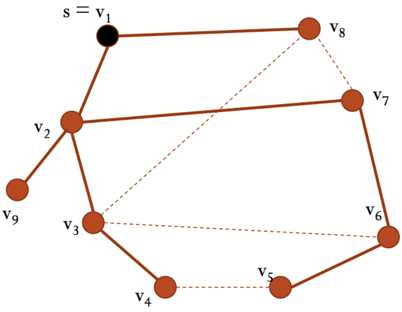
- The DFS algorithm defined the following "DFS tree" rooted at s
  - Vertex u is the parent of vertex v if u caused the immediate discovery of v.

- The DFS algorithm defined the following "DFS tree" rooted at s
  - Vertex u is the parent of vertex v if u caused the immediate discovery of v.

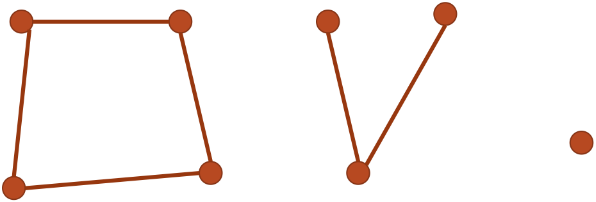


- DFS tree Vs BFS tree

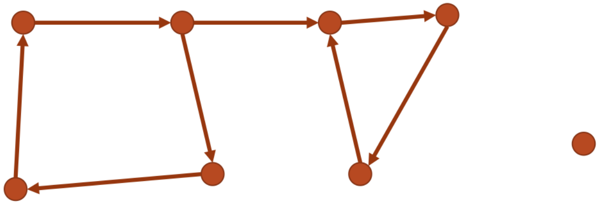




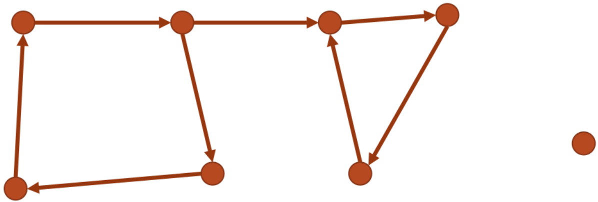
- A graph may not always be "connected".
- A connected component in an undirected graph is a maximal subgraph (maximal subset of vertices along with respective edges) such that there is a path between any pair of vertices in the subset.



- In a directed graph, a strongly connected component is a maximal subgraph such that for each pair of vertices (u, v) in the subset, there is a path from u to v and there is a path from v to u.



- Question: Given a directed graph, can a vertex be in two strongly connected components?

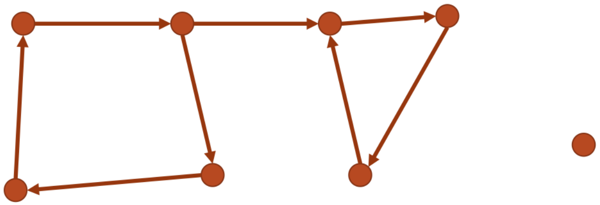


- Question: Given a directed graph, can a vertex be in two strongly connected components? No
  - For sake of contradiction, assume that there is a vertex v and vertex sets A, B in two strongly connected components s.t. $v \in A$, $v \in B$ and $A \neq B$.
  - Claim: For ever pair of vertices $p, q \in A \cup B$, there is a path from p to q and there is a path from q to p.
  - This implies that either A or B is not a *maximal* subset.

- Question: Given a directed graph, can a vertex be in two strongly connected components? No

Problem

Given a directed graph and a vertex s. Give an algorithm to find the vertices in the strongly connected component containing s. What is the running time?



End