Linear-Time Graph Algorithms

T. M. Murali

February 7, 2024

All Connected Components

- We have discussed the component containing a particular node s.
- Each node belongs to a component.
- What is the relationship between all these components?

All Connected Components

- We have discussed the component containing a particular node s.
- Each node belongs to a component.
- What is the relationship between all these components?
  - If v is in u's component, is u in v's component?
  - If v is not in u's component, can u be in v's component?

All Connected Components

- We have discussed the component containing a particular node s.
- Each node belongs to a component.
- What is the relationship between all these components?
  - If v is in u's component, is u in v's component?
  - If v is not in u's component, can u be in v's component?
- Claim: For any two nodes s and t in a graph, their connected components are either equal or disjoint. Read proof in page 86 of your textbook.

Computing All Connected Components

1. Pick an arbitrary node s in G.
2. Compute its connected component using BFS (or DFS).
3. Find a node (say v, not already visited) and repeat the BFS from v.
4. Repeat this process until all nodes are visited.

- Time spent to compute each component is

Computing All Connected Components

1. Pick an arbitrary node s in G.
2. Compute its connected component using BFS (or DFS).
3. Find a node (say v, not already visited) and repeat the BFS from v.
4. Repeat this process until all nodes are visited.

- Time spent to compute each component is linear in the size of the *component*.
- Running time of the algorithm is

Computing All Connected Components

1. Pick an arbitrary node s in G.
2. Compute its connected component using BFS (or DFS).
3. Find a node (say v, not already visited) and repeat the BFS from v.
4. Repeat this process until all nodes are visited.

- Time spent to compute each component is linear in the size of the *component*.
- Running time of the algorithm is linear in the total sizes of the components, i.e., $O(m+n)$.

Computing All Connected Components

1. Pick an arbitrary node s in G.
2. Compute its connected component using BFS (or DFS).
3. Find a node (say v, not already visited) and repeat the BFS from v.
4. Repeat this process until all nodes are visited.

- Time spent to compute each component is linear in the size of the *component*.
- Running time of the algorithm is linear in the total sizes of the components, i.e., $O(m + n)$.
- Connectivity in directed graphs: Read Chapter 3.5 of your textbook.

Bipartite Graphs

- A graph $G = (V, E)$ is *bipartite* if V can be partitioned into two subsets X and Y such that every edge in E has one endpoint in X and one endpoint in Y.
  - $(X \times X) \cap E = \emptyset$ and $(Y \times Y) \cap E = \emptyset$.
  - Colour the nodes in X red and the nodes in Y blue. Then no edge in E connects nodes of the same colour.
- Examples of bipartite graphs:

Bipartite Graphs

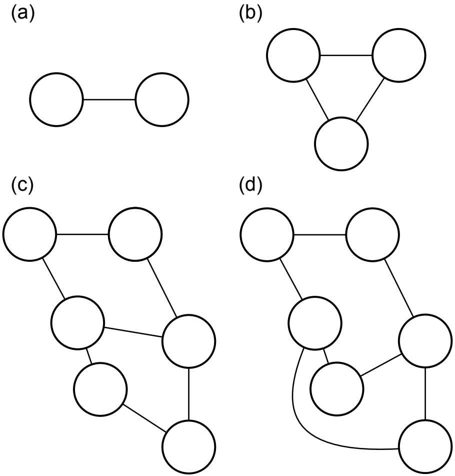
- A graph $G = (V, E)$ is *bipartite* if V can be partitioned into two subsets X and Y such that every edge in E has one endpoint in X and one endpoint in Y.
  - $(X \times X) \cap E = \emptyset$ and $(Y \times Y) \cap E = \emptyset$.
  - Colour the nodes in X red and the nodes in Y blue. Then no edge in E connects nodes of the same colour.
- Examples of bipartite graphs: medical residents and hospitals, COVID-19 vaccines and countries in which they are being adminsitered, jobs and processors they can be scheduled on, professors and courses they can teach.

TestBipartiteness

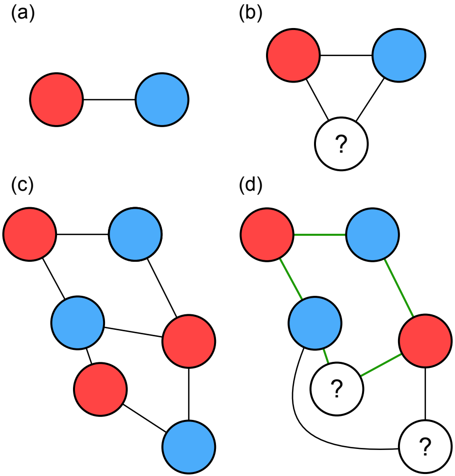
INSTANCE: An undirected graph $G = (V, E)$

QUESTION: Is G bipartite?

Examples



Examples



- A triangle is not bipartite.
- Generalisation: No cycle of odd length is bipartite.
- Claim: If a graph is bipartite, then it cannot contain a cycle of odd length.

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red.

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue.

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue. Colour the uncoloured neighours of *these* nodes red, and so on till all nodes are coloured.

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue. Colour the uncoloured neighours of *these* nodes red, and so on till all nodes are coloured. Check if very edge has endpoints of different colours.

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue. Colour the uncoloured neighours of *these* nodes red, and so on till all nodes are coloured. Check if very edge has endpoints of different colours. Algorithm is just like BFS!

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue. Colour the uncoloured neighbours of *these* nodes red, and so on till all nodes are coloured. Check if very edge has endpoints of different colours. Algorithm is just like BFS!
- Algorithm:
  1. Run BFS on G. Maintain an additional array `Colour`.
  2. When we add a node v to a layer i, set `Colour`$[v]$ to red if i is even, otherwise to blue.
  3. At the end of BFS, scan all the edges to check if there is any edge both of whose endpoints received the same colour.

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue. Colour the uncoloured neighbours of *these* nodes red, and so on till all nodes are coloured. Check if very edge has endpoints of different colours. Algorithm is just like BFS!
- Algorithm:
  1. Run BFS on G. Maintain an additional array `Colour`.
  2. When we add a node v to a layer i, set `Colour`[v] to red if i is even, otherwise to blue.
  3. At the end of BFS, scan all the edges to check if there is any edge both of whose endpoints received the same colour.
- Running time of this algorithm is

Algorithm for Testing Bipartiteness

- Assume G is connected. Otherwise, apply the algorithm to each connected component separately.
- Idea: Pick an arbitrary node s and colour it red. Colour all its neighbours blue. Colour the uncoloured neighbours of *these* nodes red, and so on till all nodes are coloured. Check if very edge has endpoints of different colours. Algorithm is just like BFS!
- Algorithm:
  1. Run BFS on G. Maintain an additional array `Colour`.
  2. When we add a node v to a layer i, set `Colour`$[v]$ to red if i is even, otherwise to blue.
  3. At the end of BFS, scan all the edges to check if there is any edge both of whose endpoints received the same colour.
- Running time of this algorithm is $O(n + m)$, since we do a constant amount of work per node in addition to the time spent by BFS.

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.
2. If G is not bipartite, can we determine the reason from the output of the algorithm?

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.
2. If G is not bipartite, can we determine the reason from the output of the algorithm? The algorithm has effectively found a cycle of odd length in G.

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.
2. If G is not bipartite, can we determine the reason from the output of the algorithm? The algorithm has effectively found a cycle of odd length in G.

- Let G be a graph and let $L_0, L_1, L_2, \ldots L_k$ be the layers produced by BFS, starting at node s. Then exactly one of the following statements is true:
    1. No edge of G joins two nodes in the same layer:
    2. There is an edge of G that joins two nodes in the same layer:

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.
2. If G is not bipartite, can we determine the reason from the output of the algorithm? The algorithm has effectively found a cycle of odd length in G.

- Let G be a graph and let $L_0, L_1, L_2, \dots L_k$ be the layers produced by BFS, starting at node s. Then exactly one of the following statements is true:
    1. No edge of G joins two nodes in the same layer: then G is bipartite and nodes in even layers can be coloured red and nodes in odd layers can be coloured blue.
    2. There is an edge of G that joins two nodes in the same layer:

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.
2. If G is not bipartite, can we determine the reason from the output of the algorithm? The algorithm has effectively found a cycle of odd length in G.

- Let G be a graph and let $L_0, L_1, L_2, \ldots L_k$ be the layers produced by BFS, starting at node s. Then exactly one of the following statements is true:
   1. No edge of G joins two nodes in the same layer: then G is bipartite and nodes in even layers can be coloured red and nodes in odd layers can be coloured blue.
   2. There is an edge of G that joins two nodes in the same layer: then G contains a cycle of odd length and cannot be bipartite.

Correctness of the Algorithm

- Need to prove two separate claims.

1. If the algorithm says that G is bipartite, then G is indeed bipartite.
2. If G is not bipartite, can we determine the reason from the output of the algorithm? The algorithm has effectively found a cycle of odd length in G.

- Let G be a graph and let $L_0, L_1, L_2, \dots L_k$ be the layers produced by BFS, starting at node s. Then exactly one of the following statements is true:
    1. No edge of G joins two nodes in the same layer: then G is bipartite and nodes in even layers can be coloured red and nodes in odd layers can be coloured blue.
    2. There is an edge of G that joins two nodes in the same layer: then G contains a cycle of odd length and cannot be bipartite.

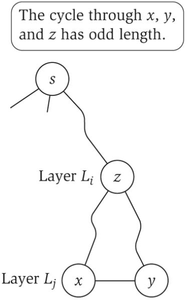


Figure 3.6 If two nodes x and y in the same layer are joined by an edge, then the cycle through x, y, and their lowest common ancestor z has odd length, demonstrating that the graph cannot be bipartite.