Graph $G = (V, E)$ is bipartite iff it can be partitioned into two sets of nodes $A$ and $B$ such that each edge has one end in $A$ and the other end in $B$.

**Alternatively:**

- Graph $G = (V, E)$ is bipartite iff all its cycles have even length.
- Graph $G = (V, E)$ is bipartite iff nodes can be coloured using two colours.

**Question:** Given a graph $G$, how to test if the graph is bipartite?

**Note:** Graphs without cycles (trees) are bipartite.

Testing bipartiteness

**Method:** Use BFS search tree.

**Recall:** BFS is a rooted spanning tree.

**Algorithm:**

- Run BFS search and colour all nodes in odd layers red, others blue.
- Go through all edges in the adjacency list and check if each of them has two different colours at its ends. If so, then $G$ is bipartite, otherwise it is not.

We use the following alternative definitions in the analysis:

- Graph $G = (V, E)$ is bipartite iff all its cycles have even length, or.
- Graph $G = (V, E)$ is bipartite iff it has no odd cycle.

Want to "sort" or linearize a directed acyclic graph (DAG).
**Topological Sort**

- Performed on a DAG.
- Linear ordering of the vertices of $G$ such that if $(u, v) \in E$, then $u$ appears before $v$.

Topological-Sort ($G$)
1. call DFS($G$) to compute finishing times $f[v]$ for all $v \in V$
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices

Time: $\Theta(V + E)$.

**Example**

Linked List:

Example

Linked List:
Example

Linked List:

A ➔ B ➔ D ➔ C ➔ E

Linked List:

A ➔ B ➔ D ➔ C ➔ E

Example

Linked List:

A ➔ B ➔ D ➔ C ➔ E

Linked List:

A ➔ B ➔ D ➔ C ➔ E

Precedence Example

1. Tasks that have to be done to eat breakfast:
   - get glass, pour juice, get bowl, pour cereal, pour milk, get spoon, eat.
2. Certain events must happen in a certain order (ex: get bowl before pouring milk)
3. For other events, it doesn't matter (ex: get bowl and get spoon)
Precedence Example

Order: glass, juice, bowl, cereal, milk, spoon, eat.

Precedence Example

Topological Sort

consider reverse order of finishing times: spoon, bowl, cereal, milk, glass, juice, eat

Correctness Proof

Show if \((u, v) \in E\), then \(f[v] < f[u]\).

When we explore \((u, v)\), what are their colors?

- Note, \(u\) is gray – we are exploring it
- Is \(v\) gray?
  - No, because then \(v\) would be an ancestor of \(u\).
  - \(\Rightarrow (u, v)\) is a back edge.
  - \(\Rightarrow\) a cycle (dag has no back edges).
- Is \(v\) white?
  - Then \(v\) becomes descendant of \(u\).
  - By parenthesis theorem, \(d[u] < d[v] < f[v] < f[u]\).
- Is \(v\) black?
  - Then \(v\) is already finished.
  - Since we're exploring \((u, v)\), we have not yet finished \(u\).
  - Therefore, \(f[v] < f[u]\).
Strongly Connected Components

- Consider a directed graph.
- A strongly connected component (SCC) of the graph is a maximal set of nodes with a (directed) path between every pair of nodes.
  - If a path from $u$ to $v$ exists in the SCC, then a path from $v$ to $u$ also exists.
- Problem: Find all the SCCs of the graph.

Uses of SCC's

- Packaging software modules
  - Construct directed graph of which modules call which other modules
  - A SCC is a set of mutually interacting modules
  - Pack together those in the same SCC

SCC Example

- Diagram showing four SCCs

Main Idea of SCC Algorithm

- DFS tells us which nodes are reachable from the roots of the individual trees
- Also need information in the other direction: is the root reachable from its descendants?
- Run DFS again on the transpose graph (reverse the directions of the edges)
SCC Algorithm

Input: directed graph $G = (V, E)$
1. call DFS($G$) to compute finishing times
2. compute $G^T$ // transpose graph
3. call DFS($G^T$), considering nodes in decreasing order of finishing times
4. each tree from Step 3 is a separate SCC of $G$

SCC Algorithm Example

After Step 1

After Step 2

Order of nodes for Step 3: f, g, a, e, b, d, c

transposed input graph - run DFS with specified order of nodes
**After Step 3**

SCCs are \{f,h,g\} and \{a,e\} and \{b,c\} and \{d\}.

**Run Time of SCC Algorithm**

- Step 1: O(V+E) to run DFS
- Step 2: O(V+E) to construct transpose graph, assuming adjacency list rep.
  * Adjacency matrix is O(1) time w/ wrapper.
- Step 3: O(V+E) to run DFS again
- Step 4: O(V) to output result
- Total: O(V+E)

**Component Graph**

- \(G^{SCC} = (V^{SCC}, E^{SCC})\).
- \(V^{SCC}\) has one vertex for each SCC in G.
- \(E^{SCC}\) has an edge if there's an edge between the corresponding SCC's in G.

**Component Graph Facts**

- **Claim:** \(G^{SCC}\) is a directed acyclic graph.
  * Suppose there is a cycle in \(G^{SCC}\) such that component \(C_i\) is reachable from component \(C_j\) and vice versa.
  * Then \(C_i\) and \(C_j\) would not be separate SCCs.
- **Lemma:** If there is an edge in \(G^{SCC}\) from component \(C'\) to component \(C\), then \(f(C') > f(C)\).
  * Consider any component \(C\) during Step 1 (running DFS on G)
  * Let \(d(C)\) be *earliest* discovery time of any node in \(C\)
  * Let \(f(C)\) be *latest* finishing time of any node in \(C\)