Lecture 10
Graph algorithms: testing graph properties

COMP 523: *Advanced Algorithmic Techniques*

Lecturer: *Dariusz Kowalski*

Overview

Previous lectures:

- Representation of graphs (adjacency matrix or list)
- Breadth-First Search (BFS)
- Depth-First Search (DFS)

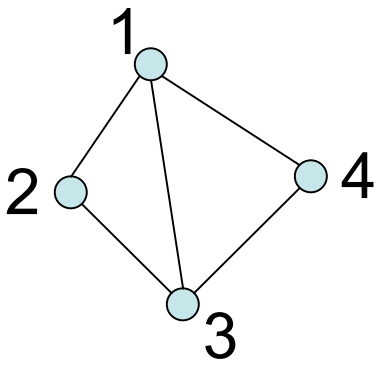
This lecture:

- Testing bipartiteness
- Testing for (directed) cycles

How to represent graphs?

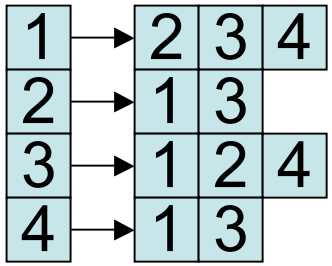
Given graph G = (V,E) , how to represent it?

1. **Adjacency matrix**: i^{th} node is represented as i^{th} row and i^{th} column, edge between i^{th} and j^{th} nodes is represented as 1 in row i and column j, and vice versa (0 if there is no edge between i and j)
2. **Adjacency list**: nodes are arranged as array/list, each node record has the list of its neighbours attached



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Adjacency matrix



Adjacency list

Adjacency list

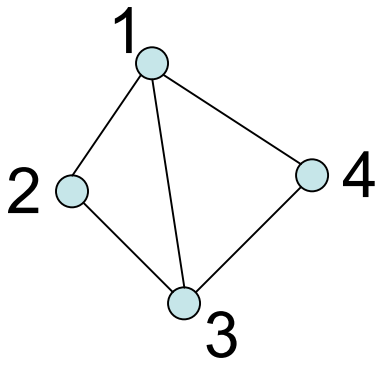
Advantages:

- Representation takes memory $O(m+n)$ - versus $O(n^2)$
- Examining all neighbours of a given node requires time $O(m/n)$ in average - versus $O(n)$

Disadvantages:

- Check if an edge belongs to the graph requires time $O(m/n)$ in average - versus $O(1)$

Advantages especially for sparse graphs!



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Adjacency matrix

1	2	3	4
2	1	3	
3	1	2	4
4	1	3	

Adjacency list

Bipartiteness

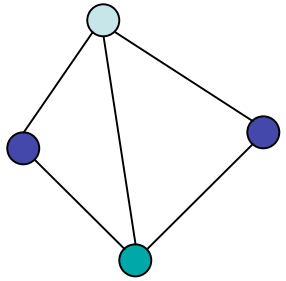
Graph G = (V,E) is **bipartite** iff it can be partitioned into two sets of nodes A and B such that each edge has one end in A and the other end in B.

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.

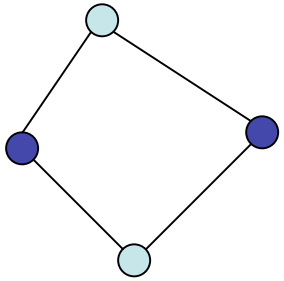
Question: given a graph represented as the adjacency list, how to test if graph is bipartite?

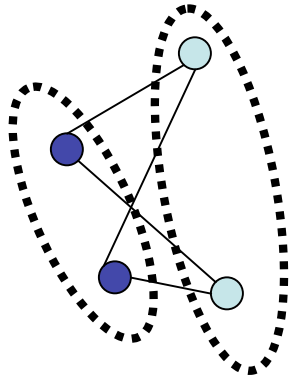
Note: graphs without cycles (trees) are bipartite.



not bipartite

bipartite:





Testing bipartiteness

We use following definition:

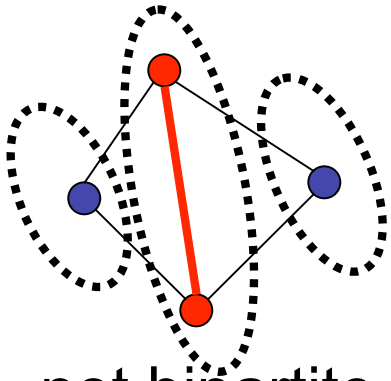
- Graph G = (V,E) is bipartite iff all its cycles have even length, or
- Graph G = (V,E) is bipartite iff it has no odd cycle.

Method: use BFS search tree.

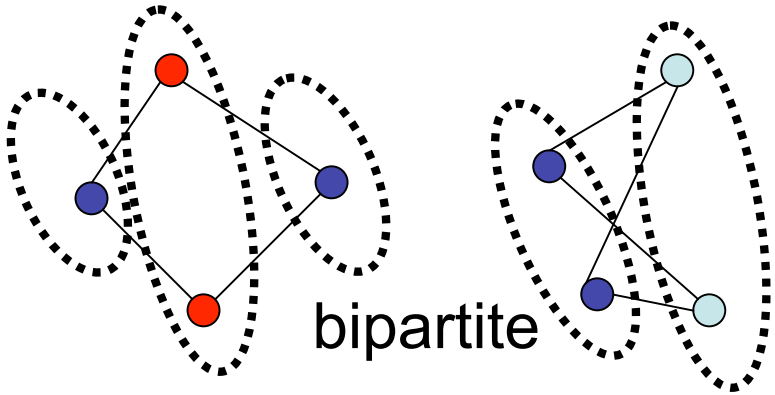
Recall: BFS is a rooted spanning tree having the same layers as the original graph G (each node has the same distance from the root in BFS tree and in graph G)

Algorithm:

- Perform BFS search and colour all nodes in odd layers red, others blue
- Go through all edges in adjacency list and check if each of them has two different colours at its ends - if so then G is bipartite



not bipartite



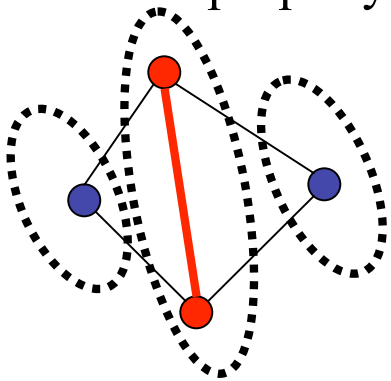
bipartite

Testing bipartiteness - why it works

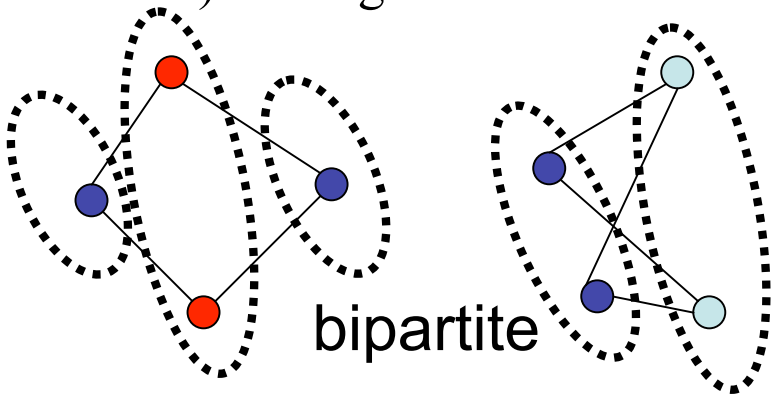
Property of layers:

- Every edge is either between two consecutive layers or within one layer (two ends are in the same layer) - it follows from BFS property

1. Suppose that graph G is not bipartite. Hence there is an odd cycle. This cycle must have an edge in one layer, and so the ends of this edge have the same colour, so the algorithm answers correctly.
2. Suppose that graph G is bipartite. Hence all its cycles have even length. Suppose, to the contrary, that there is uni-coloured edge. This edge must be in one layer. Consider one of such edges which is in the smallest possible layer. Consider a cycle containing this edge and the BFS path between its two ends. The length of this path is odd, which is a contradiction. Thus all edges are coloured properly (in sense of their ends) and algorithm is correct.



not bipartite



bipartite

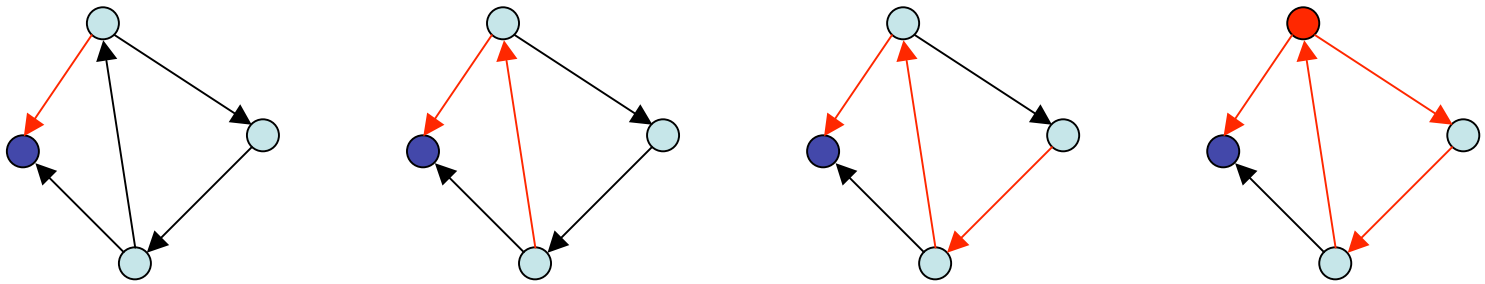
Testing cycles

Directed graph G (edges have direction - one end is the beginning, the other one is the end of the edge).

Reversed graph G^r has the same nodes but each edge is a reversed edge from graph G.

Representing directed graph: adjacency list - each node has two lists: of in-coming and out-coming edges/neighbours

Problem: does the graph has a directed cycle?



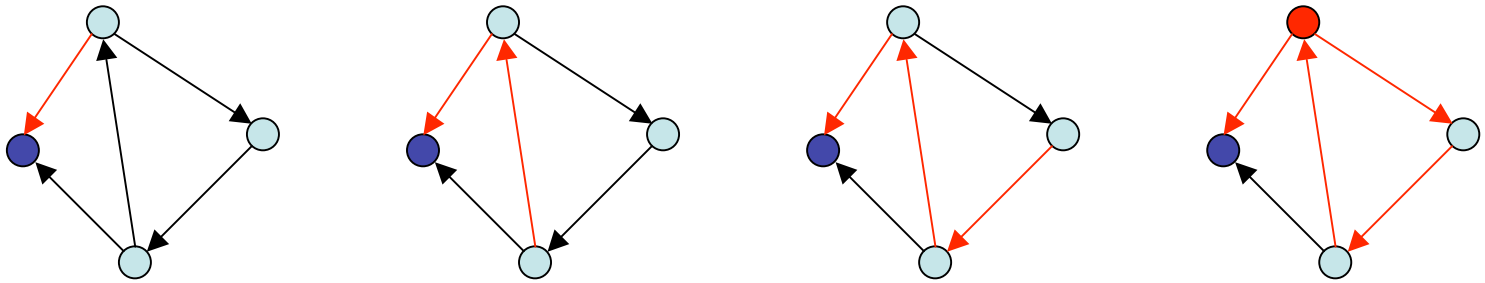
Testing cycles

Technique: use directed DFS tree for graph G.

Algorithm:

- Find a node with no incoming edges - if not found answer *cycled*
- Build a DFS directed tree - consider only edges in proper direction. If during building DFS two nodes which are already in the stack are considered then answer *cycled*, otherwise answer *acyclic*.

General remark: if graph G is not connected then do the same until no free node remains.

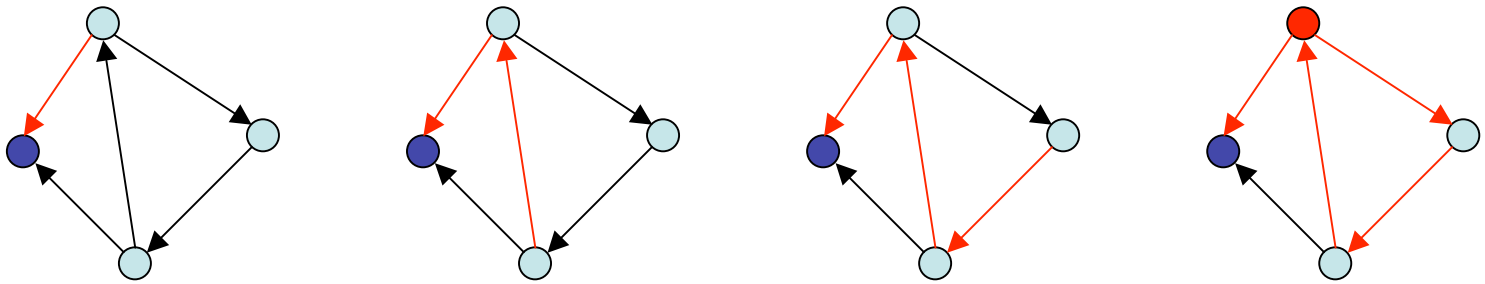


Testing cycles - analysis

Property of acyclic graph:

- There is a node with no incoming edges

1. Suppose that graph G is acyclic. Hence there is no directed cycle and then while building DFS directed tree we never try to explore the already visited node. Answer *acyclic* is then proper.
2. Suppose that graph G is not acyclic. It follows that there is a cycle in it. Let *v* be the first node in a cycle visited by DFS search. By the property of DFS, all nodes from this cycle will be reached during the search rooted in *v*, and so the out-neighbour of *v* from this cycle will attempt to visit *v* also, which causes that the algorithm stops with the correct answer *cycled*.



Conclusions

- Testing graph properties
  - bipartiteness (based on BFS),
  - if a directed graph is acyclic (based on DFS),

  in time $O(m + n)$.

Textbook and Exercises

- Chapter 3, Sections 3.4, 3.5 and 3.6.
- Prove that if a DFS and BFS trees in graph G are the same, than G is also the same like they (does not contain any other edge).
- Prove that if G has n nodes, where n is even, and each node has at least $n/2$ neighbours then G is connected.
- Prove the property of layers: every edge is either between two consecutive layers or within one layer (two ends are in the same layer).
- Prove the property of acyclic graphs: there is a node with no incoming edges. Is this property true for out-coming edges?
- Design and analyze time and memory taken by BFS for directed graphs.
- What happens if you try to use a different method of searching to check bipartiteness or acyclicity?