The Vertex coloring problem and bipartite graphs

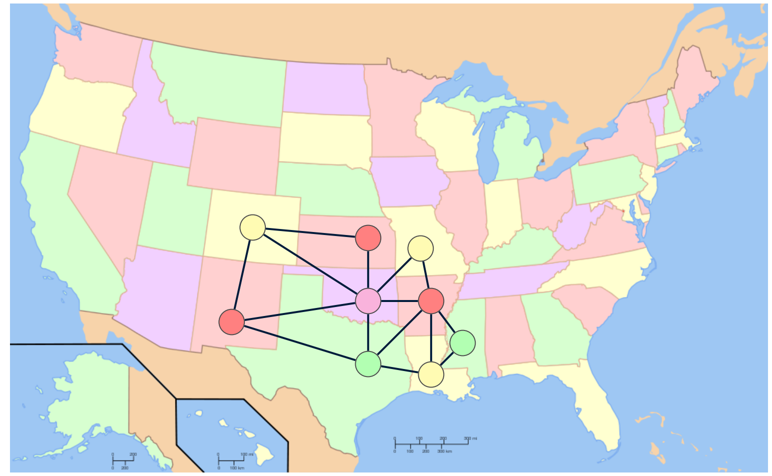
Tyler Moore

CSE 3353, SMU, Dallas, TX

Lecture 8

Some slides created by or adapted from Dr. Kevin Wayne. For more information see `http://www.cs.princeton.edu/~wayne/kleinberg-tardos`. Graph-coloring of registers adapted from Stanford's CS143 (Aiken, Treichler).

Vertex-coloring problem



Vertex-coloring problem

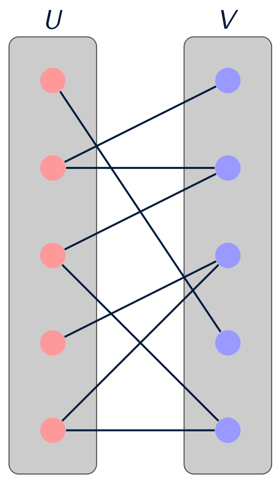
- The vertex-coloring problem seeks to assign a label (aka color) to each vertex of a graph such that no edge links any two vertices of the same color
- Trivial solution: assign each vertex a different color
- However, goal is usually to use as few colors as possible

Applications of the vertex-coloring problem

- Apart from working at National Geographic, when might you encounter a vertex-coloring problem?
- Vertex-coloring problems arise in scheduling problems, where access to shared resources must be coordinated
- Example: register allocation by compilers
  - Variables are used for fixed timespan (after initialization, before final use)
  - Two variables with intersecting lifespans can't be put in the same register
  - We can build a graph with variables assigned to vertices and edges drawn between vertices if the variables' lifespan intersects
  - Color the graph, and assign variables to the same register if their vertices have the same color

Vertex-coloring problem special case: two colors

- A bipartite graph is an undirected graph whose vertices can be divided into disjoint sets U and V such that every edge connects a vertex in U to one in V.
- Bipartite graphs arise in matching problems: matching workers to jobs, matching kidney donors with recipients, finding heterosexual mates
- If we can color a graph's vertices using just two colors, then we have a bipartite graph
- Problem: given a graph, find its two-coloring or report that a two-coloring is not possible

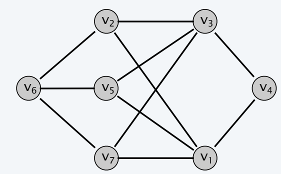


Testing bipartiteness

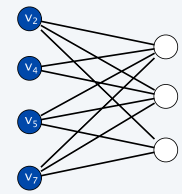
Many graph problems become:

- Easier if the underlying graph is bipartite (matching).
- Tractable if the underlying graph is bipartite (independent set).

Before attempting to design an algorithm, we need to understand structure of bipartite graphs.



a bipartite graph G

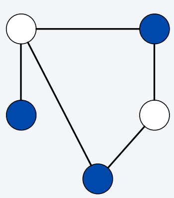


another drawing of G

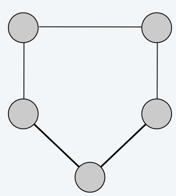
An obstruction to bipartiteness

Lemma. If a graph G is bipartite, it cannot contain an odd length cycle.

Pf. Not possible to 2-color the odd cycle, let alone G.



bipartite
(2-colorable)



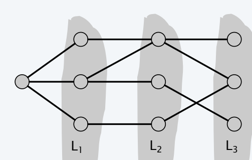
not bipartite
(not 2-colorable)

Bipartite graphs

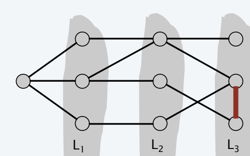
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).



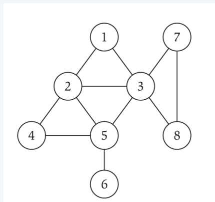
Case (i)

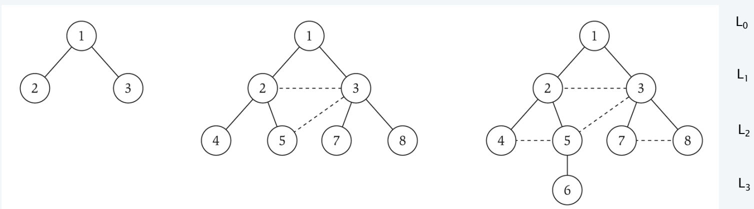


Case (ii)

Breadth-first search

Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then, the level of x and y differ by at most 1.





(a) (b) (c)

Bipartite graphs

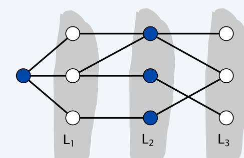
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

Pf. (i)

- Suppose no edge joins two nodes in adjacent layers.
- By previous lemma, this implies all edges join nodes on same level.
- Bipartition: red = nodes on odd levels, blue = nodes on even levels.



Case (i)

Bipartite graphs

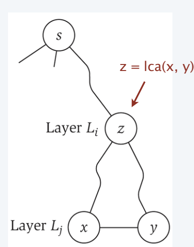
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

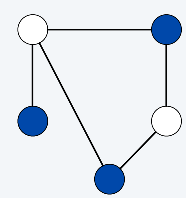
Pf. (ii)

- Suppose (x, y) is an edge with x, y in same level L_j.
- Let $z = lca(x, y)$ = lowest common ancestor.
- Let L_i be level containing z.
- Consider cycle that takes edge from x to y, then path from y to z, then path from z to x.
- Its length is $\underbrace{1}_{(x,\,y)} + \underbrace{(j-i)}_{\text{path from } y \text{ to } z} + \underbrace{(j-i)}_{\text{path from } z \text{ to } x}$, which is odd. ▪

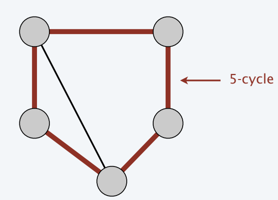


The only obstruction to bipartiteness

Corollary. A graph G is bipartite iff it contain no odd length cycle.



bipartite (2-colorable)



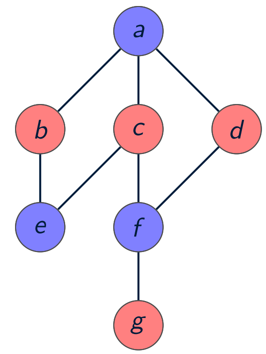
not bipartite (not 2-colorable)

Two-coloring algorithm

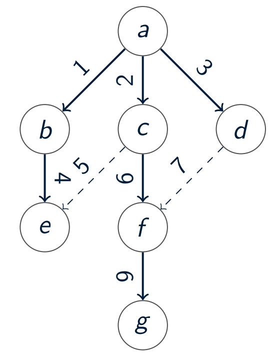
1. Suppose there are two colors: blue and red.
2. Color the first vertex blue.
3. Do a breadth-first traversal. For each newly-discovered node, color it the opposite of the parent (i.e., red if parent is blue)
4. If the child node has already been discovered, check if the colors are the same as the parent. If so, then the graph isn't bipartite.
5. If the traversal completes without any conflicting colors, then the graph is bipartite.

Two-coloring algorithm example 1

Undirected Graph

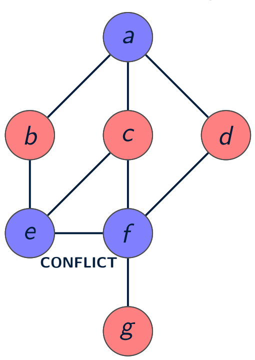


Breadth-First Search Tree

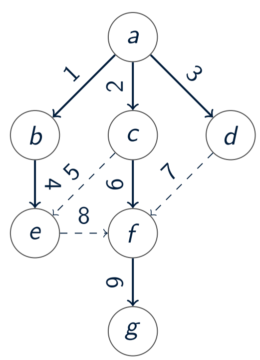


Two-coloring algorithm example 2

Undirected Graph



Breadth-First Search Tree



Exercise: Check for bipartness

Global Optimization

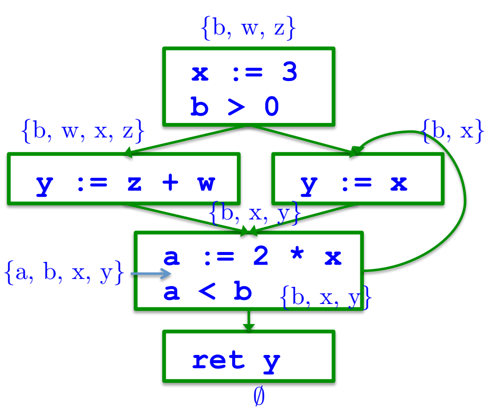
- Consider the following the following IL:

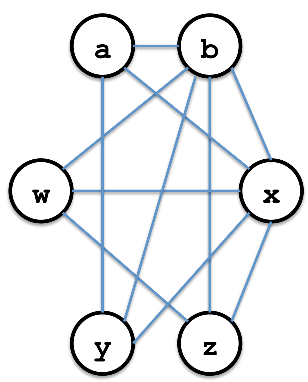
```
  x := 3
  if b > 0 goto L1
  y := z + w
  goto L2
L1:
  y := x
L2:
  a := 2 * x
  if a < b goto L1
  ret y
```


Register Interference Graph

- Use liveness analysis to compute a *register interference graph* (RIG)
- Each variable is a node in the RIG
- An edge exists between two nodes (variables) if:
  - at ANY point in program, both variables are live
- Directly connected nodes are variables that cannot share a register

RIG Example





Graph Coloring

- A *coloring* of a graph is a assignment of colors to nodes:
  - such that node that share an edge have different colors
- A *k-coloring* is a coloring that uses at most *k* different colors
- A *k*-colorable graph is a graph for which there exists at least one *k*-coloring

Register Allocation via Coloring

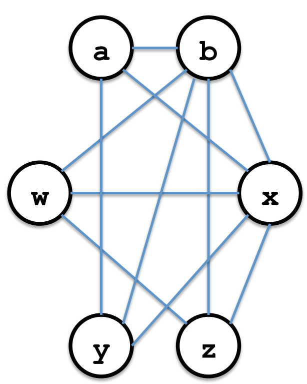
- A *k*-coloring of a RIG is a valid register allocation for *k* registers:
  - Each color is a register
  - Variables with the same color are never live at the same time
- Graph coloring is a hard problem (NP-hard)
  - Have to use heuristics

Our Heuristic

- Start with full RIG
- While graph is not empty:
  - Select a node with minimum number of edges
  - Remove node from graph, place on stack
- While stack is not empty:
  - Pop node from stack, put back in graph
  - Add back any edges to other nodes in graph
  - Pick a color for the node that doesn't match any neighbor
    - Pick a new color if necessary

Example Coloring

Graph:



Stack:

y

Example Coloring

Graph:

Stack:

b

w

x

z

a

y

Example Coloring

Graph:

Stack:

b

w

x

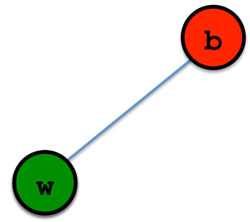
z

a

y

Example Coloring

Graph:



Stack:

w

x

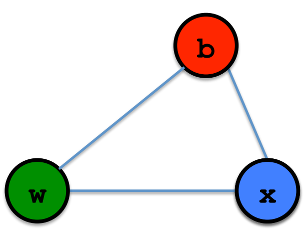
z

a

y

Example Coloring

Graph:



Stack:

x

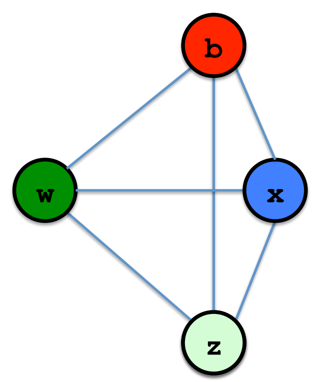
z

a

y

Example Coloring

Graph:



Stack:

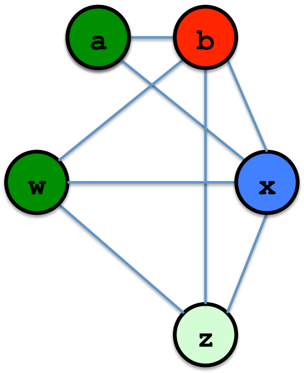
z

a

y

Example Coloring

Graph:



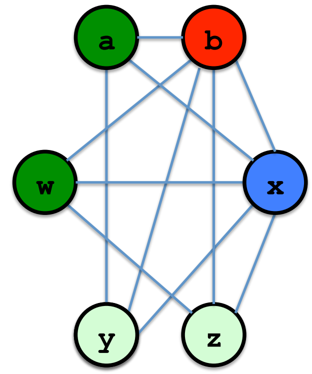
Stack:

a

y

Example Coloring

Graph:

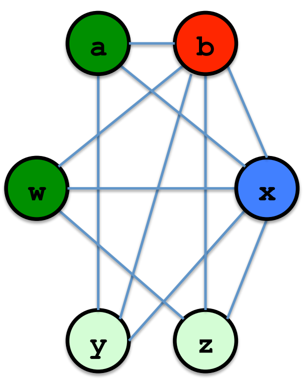


Stack:

y

Example Coloring

Graph:

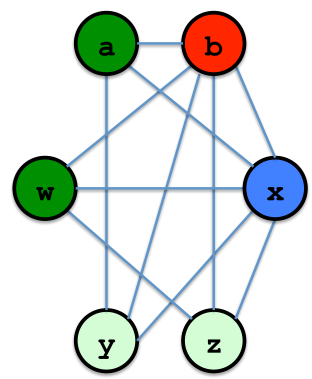


Code:

```
  x := 3
  if b > 0 goto L1
  y := z + w
  goto L2
L1:
  y := x
L2:
  a := 2 * x
  if a < b goto L1
  ret y
```


Example Coloring

Graph:



Code:

```
  r3 := 3
  if r1 > 0 goto L1
  r4 := r4 + r2
  goto L2
L1:
  r4 := r3
L2:
  r2 := 2 * r3
  if r2 < r1 goto L1
  ret r4
```