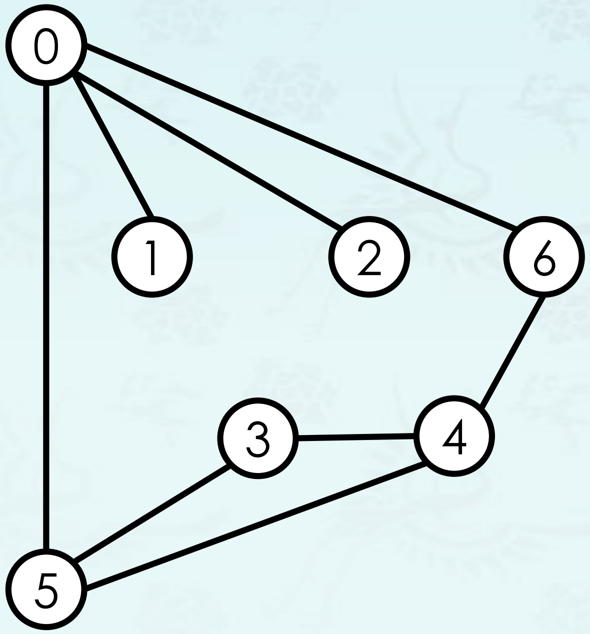
Graph

- Adjacency list processing
- Graph API - Implementation
  - **Cycle**
  - Bipartite

Major references:
1. Fundamentals of Data Structures by Horowitz, Sahni, Anderson-Freed,
2. Algorithms 4^{th} edition - Part 1 & Part 2 by Robert Sedgewick and Kevin Wayne
3. Wikipedia and many resources available from internet

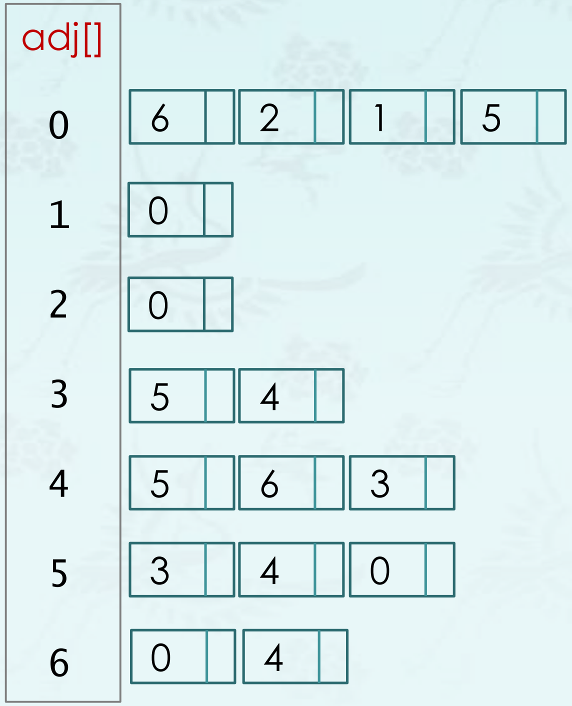
Adjacency list processing

Challenge: How to process adj[v] and its vertices:



Graph g

Adjacency lists



V-E lists

```
graph3.txt
13  ← V
13  ← E
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

Adjacency list processing

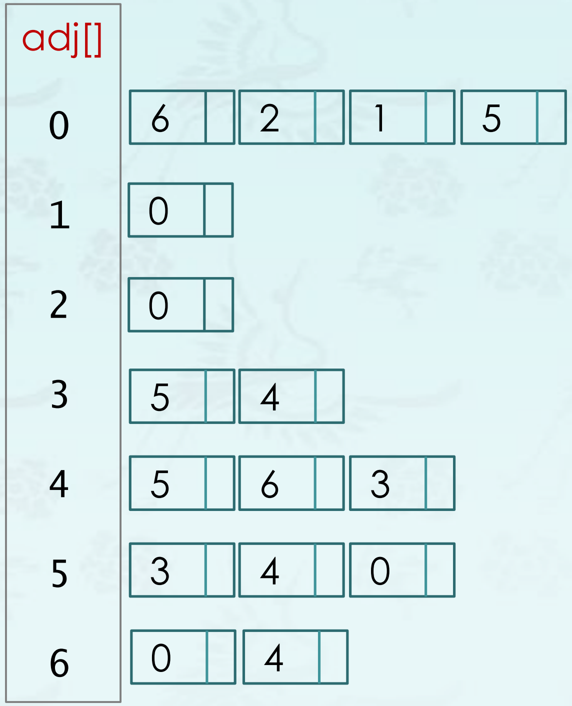
Challenge: How to process adj[v] and its vertices:

```
// print the adjacency list of graph
void print_adjlist(graph g) {

  cout << "\n\tAdjacency-list: \n";
  for (int v = 0; v < V(g); ++v) {
    cout << "\tV[" << v << "]: ";
    gnode w = g->adj[v].next;
    while (w) {
        ~~

    }
    cout << endl;
  }
}
```

Adjacency lists



Adjacency list processing

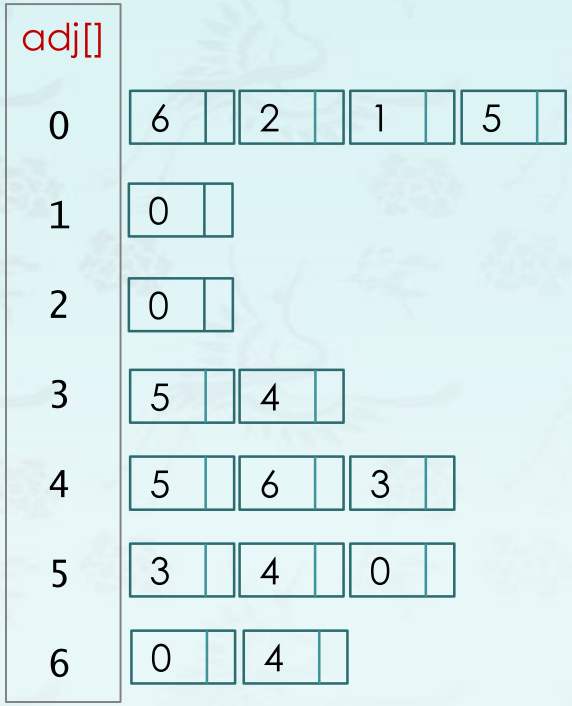
Challenge: How to process adj[v] and its vertices:

```
// print the adjacency list of graph
void print_adjlist(graph g) {

  cout << "\n\tAdjacency-list: \n";
  for (int v = 0; v < V(g); v++) {
    cout << "\tV[" << v << "]: ";
    for (gnode w = g->adj[v].next; w; w = w->next) {

       ~~
    }
}
```

Adjacency lists



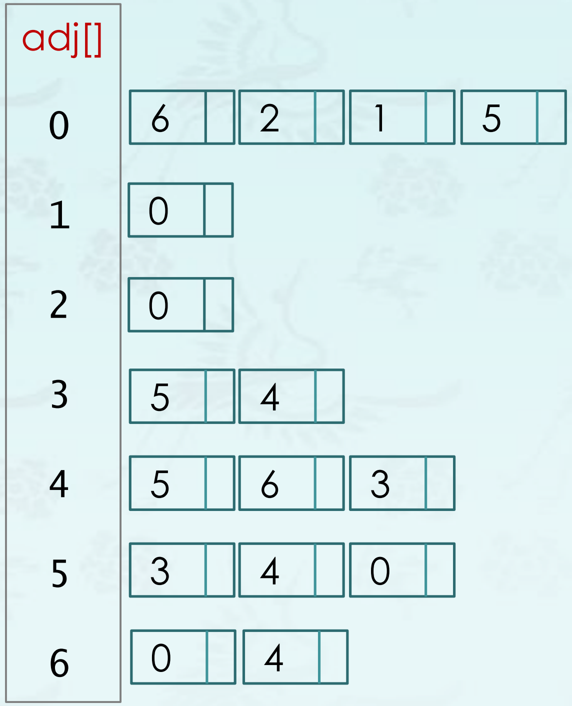
Adjacency list processing

Challenge: How to process adj[v] and its vertices:

```
// print the adjacency list of graph
void print_adjlist(graph g) {

  cout << "\n\tAdjacency-list: \n";
  for (int v = 0; v < V(g); ++v) {
    cout << "\tV[" << v << "]: ";
    for (gnode w = g->adj[v].next; w; w = w->next) {
      cout << w->item << " ";
      if (w->next == nullptr)
        cout << endl);
      else
        cout << "-> ";
    }
  }
}
```

Adjacency lists



Graph-processing challenge 1 – Review

Problem: Is a graph bipartite (or bigraph)?

a set of graph vertices decomposed into two disjoint sets
such that no two graph vertices within the same set are adjacent.

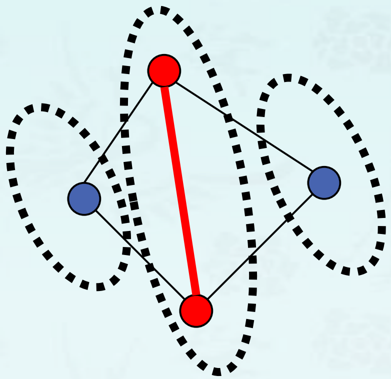
How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

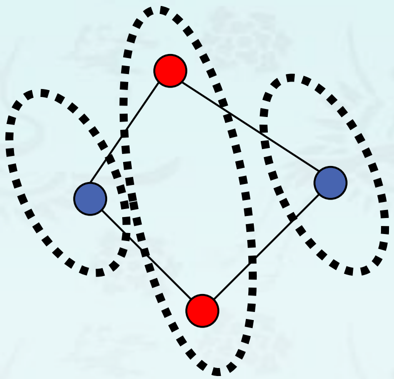
Graph-processing challenge 1 – Review

Problem: Is a graph bipartite (or bigraph)?

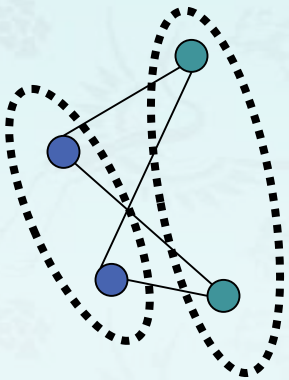
a set of graph vertices decomposed into **two disjoint sets**
such that no two graph vertices within the same set are adjacent.



non bipartite



bipartite



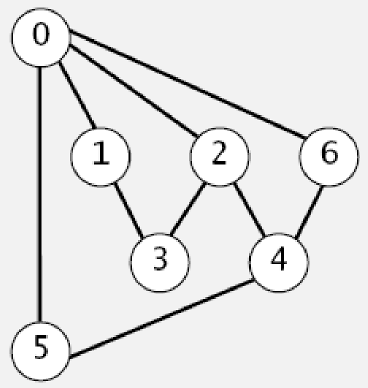
bipartite

Graph-processing challenge 1 – Review

Problem: Is a graph bipartite (or bigraph)?

a set of graph vertices decomposed into two disjoint sets
such that no two graph vertices within the same set are adja

a bigraph ?



How difficult?

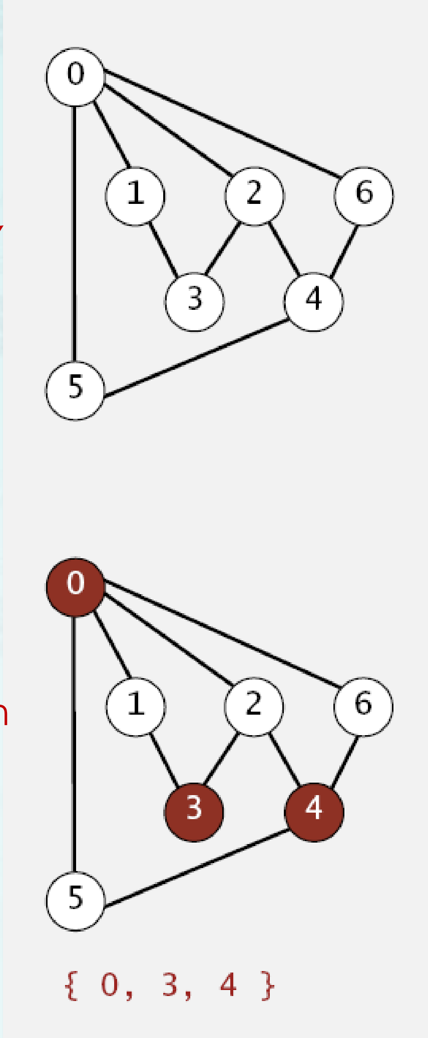
- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Graph-processing challenge 1 – Review

Problem: Is a graph bipartite (or bigraph)?

a set of graph vertices decomposed into two disjoint sets
such that no two graph vertices within the same set are adja

a bigraph ?



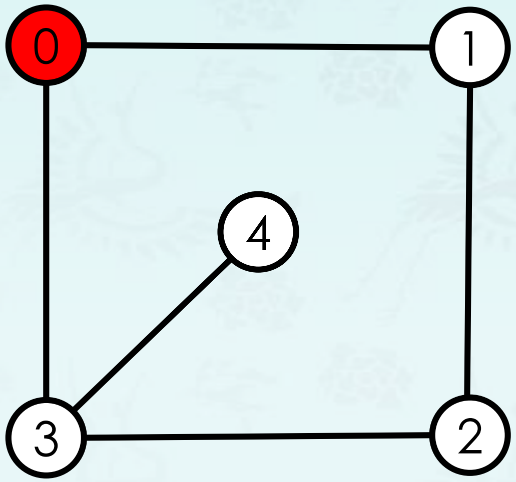
How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

simple DFS or BFS-based solution

Graph-processing challenge 1 – bigraph

Problem: Is a graph bipartite (or bigraph)?



Adjacency lists

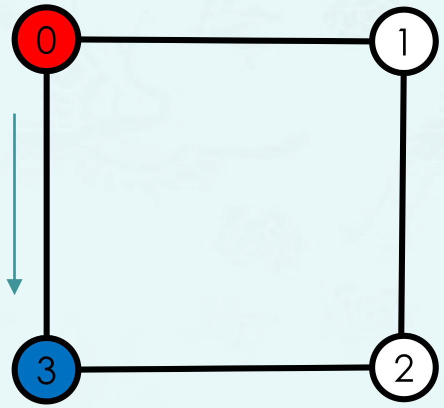
visit 0: check 3, **check 1**

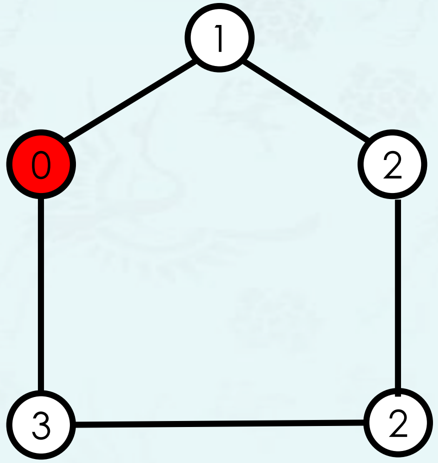
Graph-processing challenge 1 – bigraph

Problem: Is a graph bipartite (or bigraph)?

Solution: Two-colorability
The vertices of a given graph can be assigned one of two colors in such a way that no edge connects vertices of the same color.

Solution: It is called two-colorability. graphBipartite() uses depth-first search to determine whether or not a graph has a bipartition; if so, return one; if not, return an odd-length cycle. It takes time proportional to V + E in the worst case.



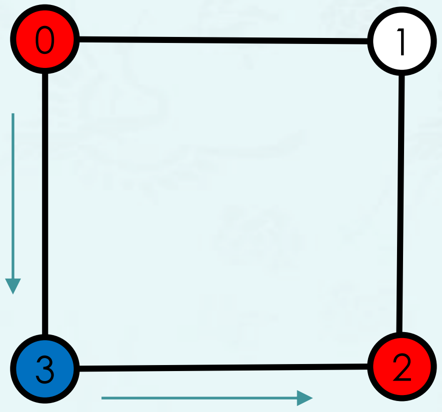


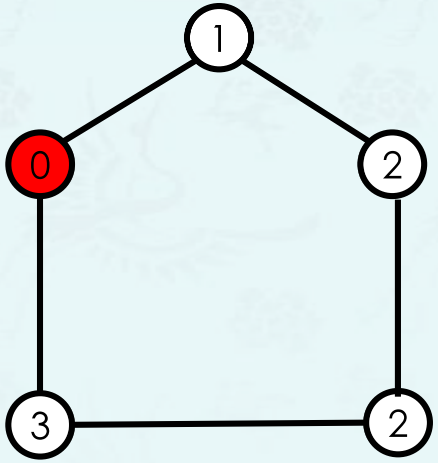
Graph-processing challenge 1 – bigraph

Problem: Is a graph bipartite (or bigraph)?

Solution: Two-colorability
The vertices of a given graph can be assigned one of two colors in such a way that no edge connects vertices of the same color.

Solution: It is called two-colorability. graphBipartite() uses depth-first search to determine whether or not a graph has a bipartition; if so, return one; if not, return an odd-length cycle. It takes time proportional to V + E in the worst case.



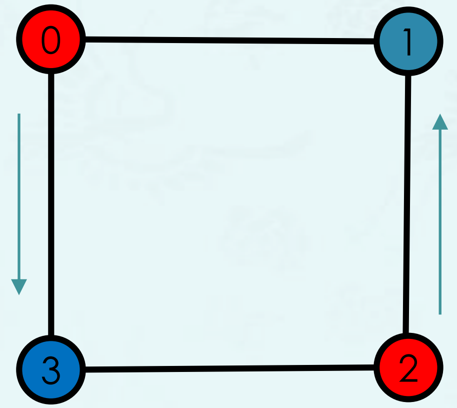


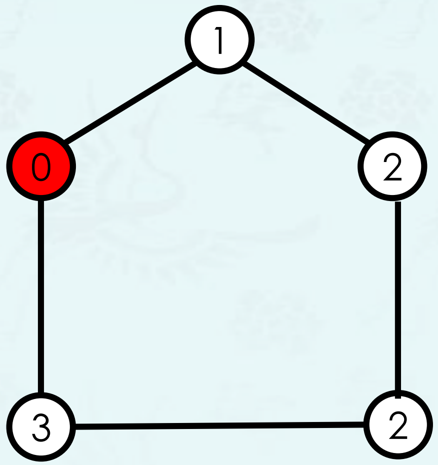
Graph-processing challenge 1 – bigraph

Problem: Is a graph bipartite (or bigraph)?

Solution: Two-colorability
The vertices of a given graph can be assigned one of two colors in such a way that no edge connects vertices of the same color.

Solution: It is called two-colorability. graphBipartite() uses depth-first search to determine whether or not a graph has a bipartition; if so, return one; if not, return an odd-length cycle. It takes time proportional to V + E in the worst case.



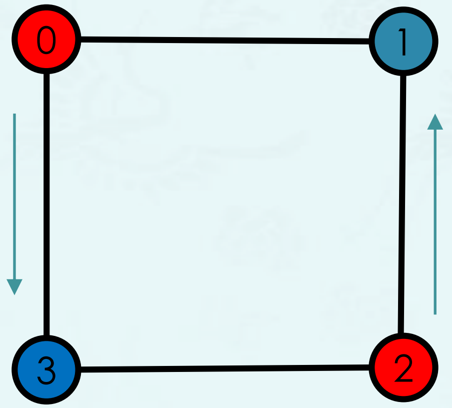


Graph-processing challenge 1 – bigraph

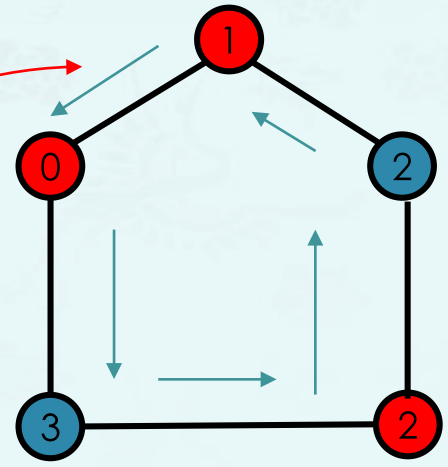
Problem: Is a graph bipartite (or bigraph)?

Solution: Two-colorability
The vertices of a given graph can be assigned one of two colors in such a way that no edge connects vertices of the same color.

Solution: bipartite() uses depth-first search to determine whether a graph has a bipartition or not; if not, return an odd-length cycle.
It takes time proportional to V + E in the worst case.



Once an odd-length cycle is found, push vertices.



Graph-processing challenge 1 – bigraph coding

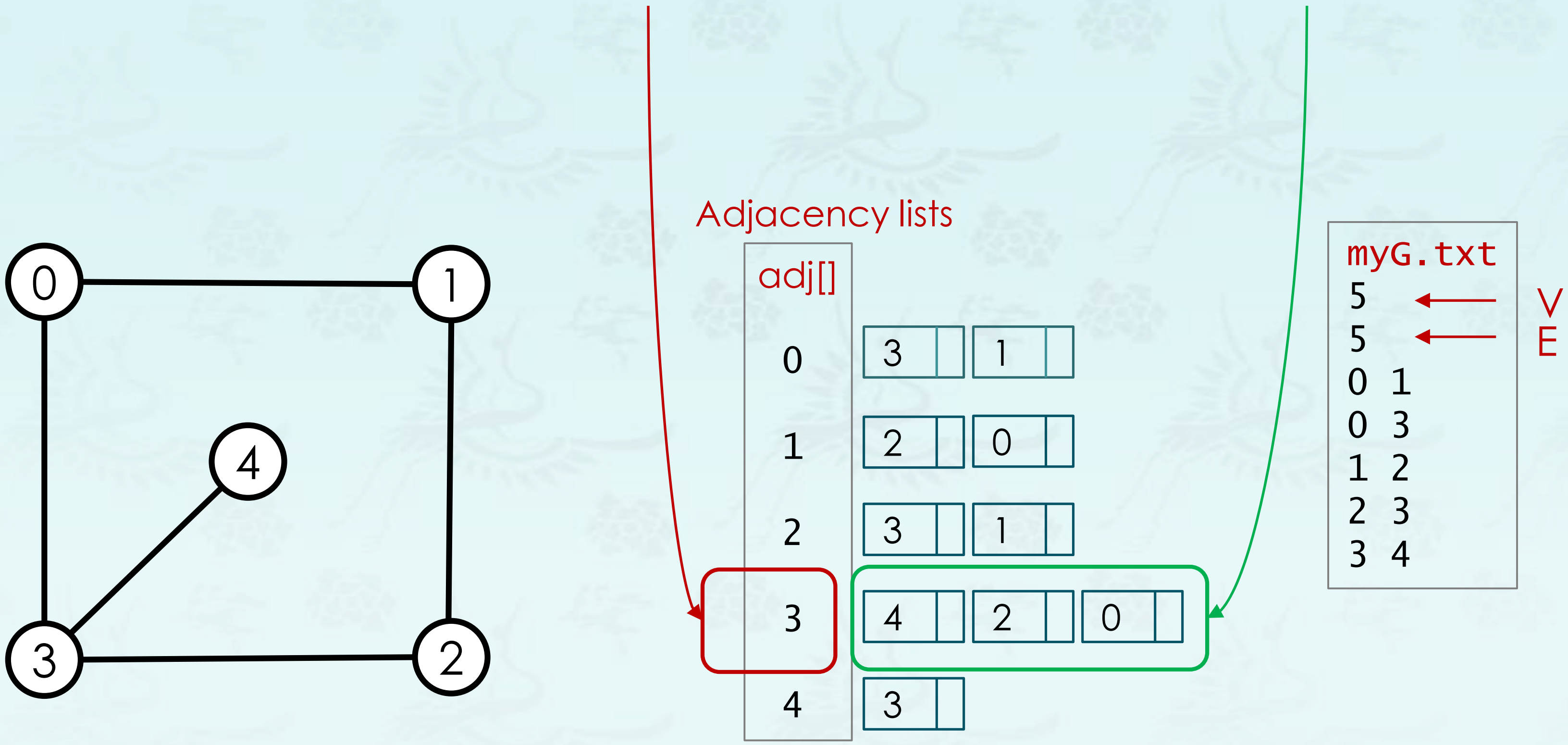
```
// determines whether or not an undirected graph is bigraph and
// finds either a bipartition or an odd length cycle.
// returns a stack with cyclic vertices pushed.
bool bigraph(graph g, stack<int>& cy) {
       if (empty(g)) return false;
       for (int i = 0; i < V(g); i++) {
              g->marked[i] = false;
              g->color[i] = BLACK;  // BLACK=0, WHITE=1
              g->parentDFS[i] = -1;   // needs info when backtrack the cycle.
       }
       cy = {};                                  // clear stack
       for (int v = 0; v < V(g); v++) {
              if (!g->marked[v]) {
                     if (!DFSbigraph(g, v, cy))
                            return false;  // found an odd-length cycle
              }
       }
       return true;
}
```

Graph-processing challenge 1 – bigraph coding

```
// Recursive DFS does the work
bool DFSbigraph(graph g, int v, stack<int>& cy) {
  g->marked[v] = true;
  for (gnode w = g->adj[v].next; w; w = w->next) {// short circuit if odd-length cycle found
    if (cy.size() > 0) return false;                // found 1st cycle
    if (!g->marked[w->item]) {                      // found uncolored vertex, so recur
        g->parentDFS[w->item] = v;                  // keep it to backtrack the cycle.
        g->color[w->item] = !g->color[v];           // flip the color
        DPRINT(cout << " " << v << " Color:" << g->color[v] << ",";);
        DPRINT(cout << " " << w->item << " Color:" << g->color[w->item] << endl;);
        DFSbigraph(g, w->item, cy);
    }   // if v-w create an odd-length cycle, find it (push vertices and push them)
    else if (g->color[w->item] == g->color[v]) {    // bipartite = false;
        // 1. instantiate a new stack and set it to g->cycle
        // 2. push w->item since first v = last v, duplicated
        // 3.  retrace g->parent[x] from v to w->item
        //       and push them to stack – need a for loop here.
        //  4. push w->item (to form a cycle)
        return false
    }
  }
  return true;
}
```

Graph-processing challenge 1 – bigraph two-colorability coding

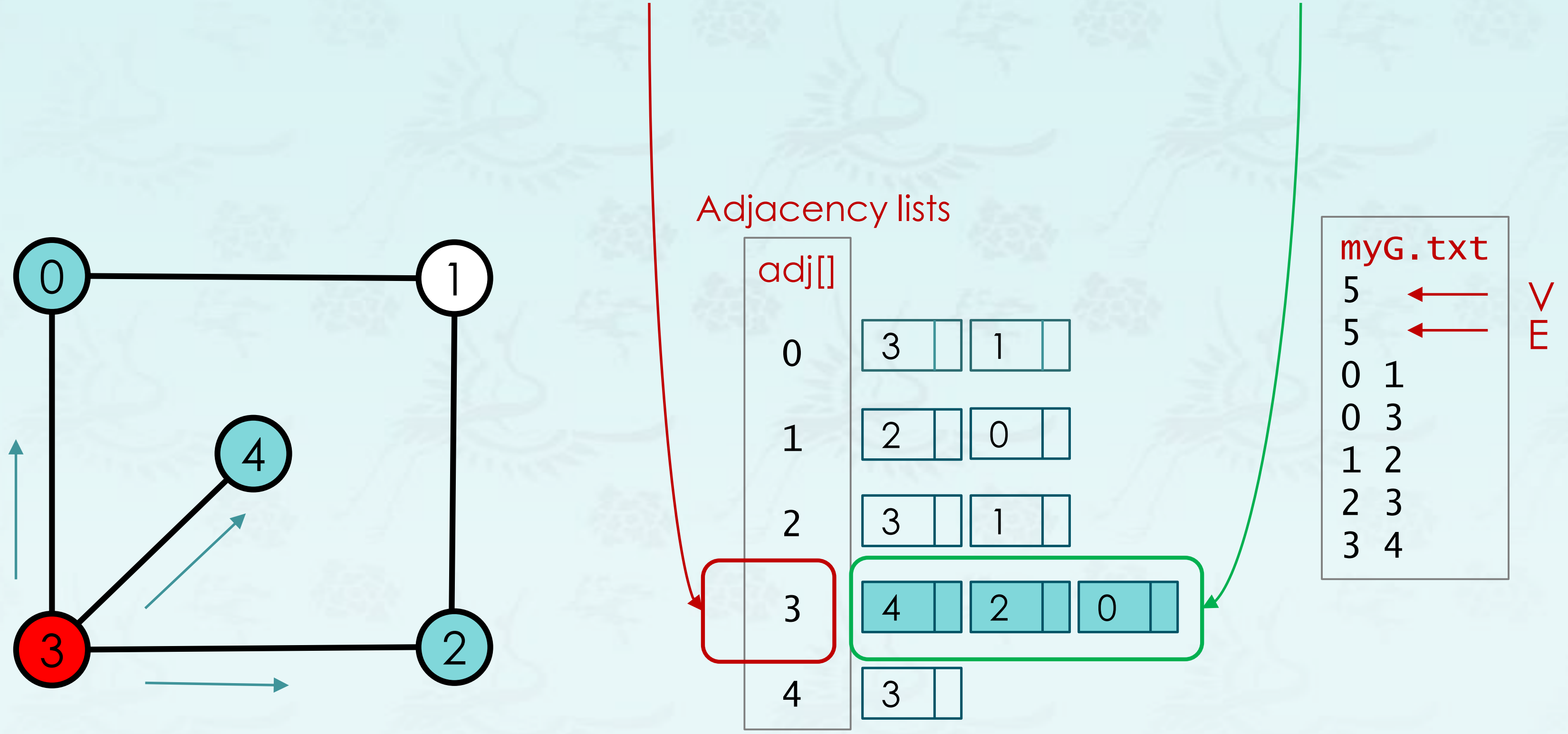
Solution: for every v, the color of adj[v] is different from those of adj[v]'s list vertices, if it is bipartite.



Graph g:

Graph-processing challenge 1 – bigraph two-colorability coding

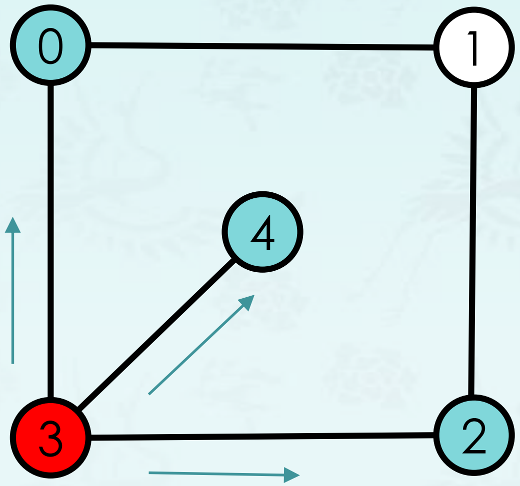
Solution: for every v, the color of adj[v] is different from those of adj[v]'s list vertices, if it is bipartite.



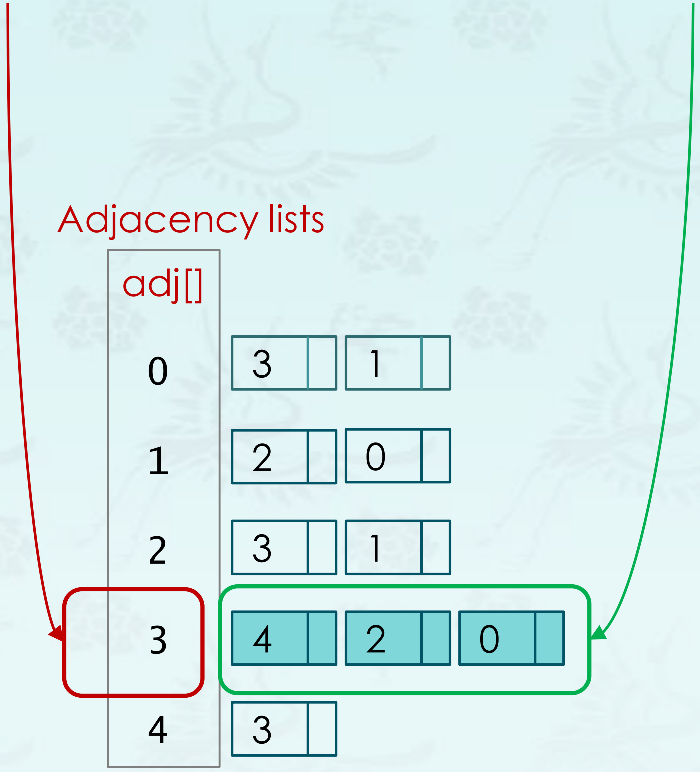
Graph g:

Graph-processing challenge 1 – bigraph two-colorability coding

Solution: for every v, the color of adj[v] is different from those of adj[v]'s list vertices, if it is bipartite.



Graph g:




```
myG.txt
5    ← V
5    ← E
0 1
0 3
1 2
2 3
3 4
```

v	marked[]	color[]
1	F	-1
2	F	-1
3	F	-1
4	F	-1
5	F	-1