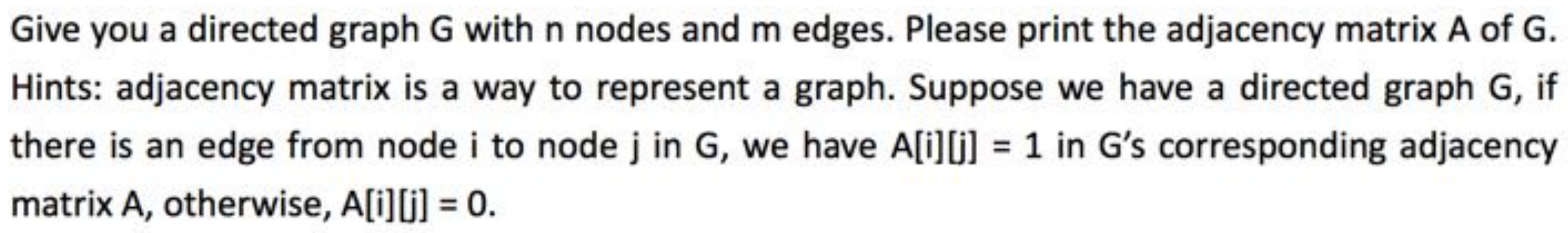
BFS相关的数据结构题回顾

赵耀

Problem A: Description



Give you a directed graph G with n nodes and m edges. Please print the adjacency matrix A of G.
Hints: adjacency matrix is a way to represent a graph. Suppose we have a directed graph G, if there is an edge from node i to node j in G, we have A[i][j] = 1 in G's corresponding adjacency matrix A, otherwise, A[i][j] = 0.

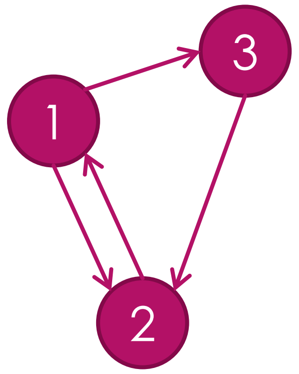
Problem A: Adjacency Matrix

- Key points:
- How to represent a directed graph using adjacency matrix
- Please notice that use double dimensional array to store data
- Use buffer to deal with the input, use StringBuilder to deal with output string(Java)
- Time complexity： $O(n^2)$

Problem A: Sample

```
2
3 5
1 2
2 1
1 3
3 2
2 3
```

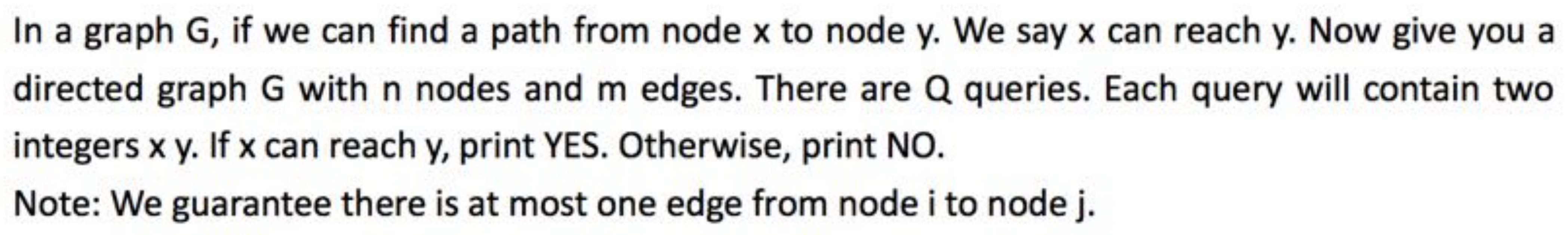
Adjacency Matrix:



	0	**1**	**2**
0	0	1	1
1	1	0	0
2	0	1	0

Graph[row-1][col-1]=1

Problem B: Description



In a graph G, if we can find a path from node x to node y. We say x can reach y. Now give you a directed graph G with n nodes and m edges. There are Q queries. Each query will contain two integers x y. If x can reach y, print YES. Otherwise, print NO.

Note: We guarantee there is at most one edge from node i to node j.

Problem B: Reachability Testing

- Key points:
  - How to represent a directed graph using adjacency list(It is instructed in DAAS LAB6)
  - BFS a graph(The difference from tree is that you must check if the node is visited!)
  - Or DFS can also do it
- Please notice that you should use flags to label the visited nodes.
- Time complexity：$O(Qn)$

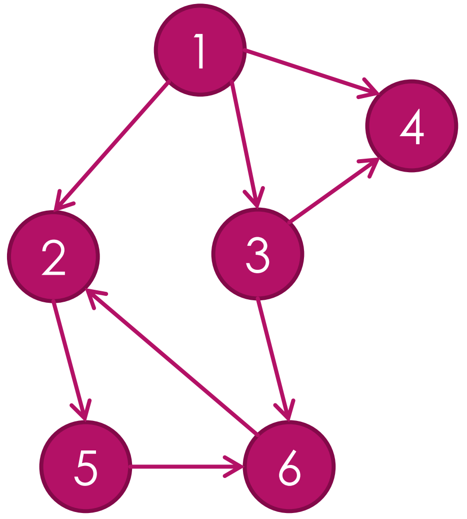
Problem B: Sample

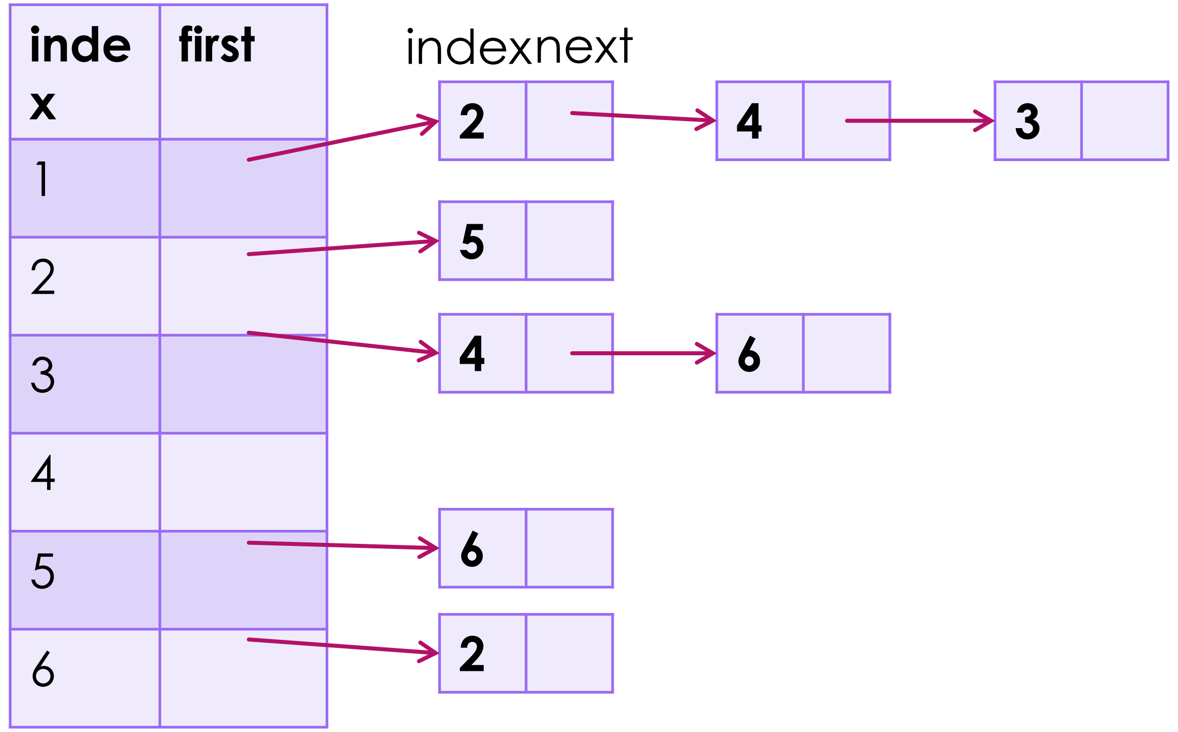
Input sequence:
1 2
1 4
1 3
3 4
3 6
6 2
2 5
5 6

Tow nodes:

1 6	YES
2 4	NO
3 2	YES
3 5	YES
4 6	NO

Adjacency List:



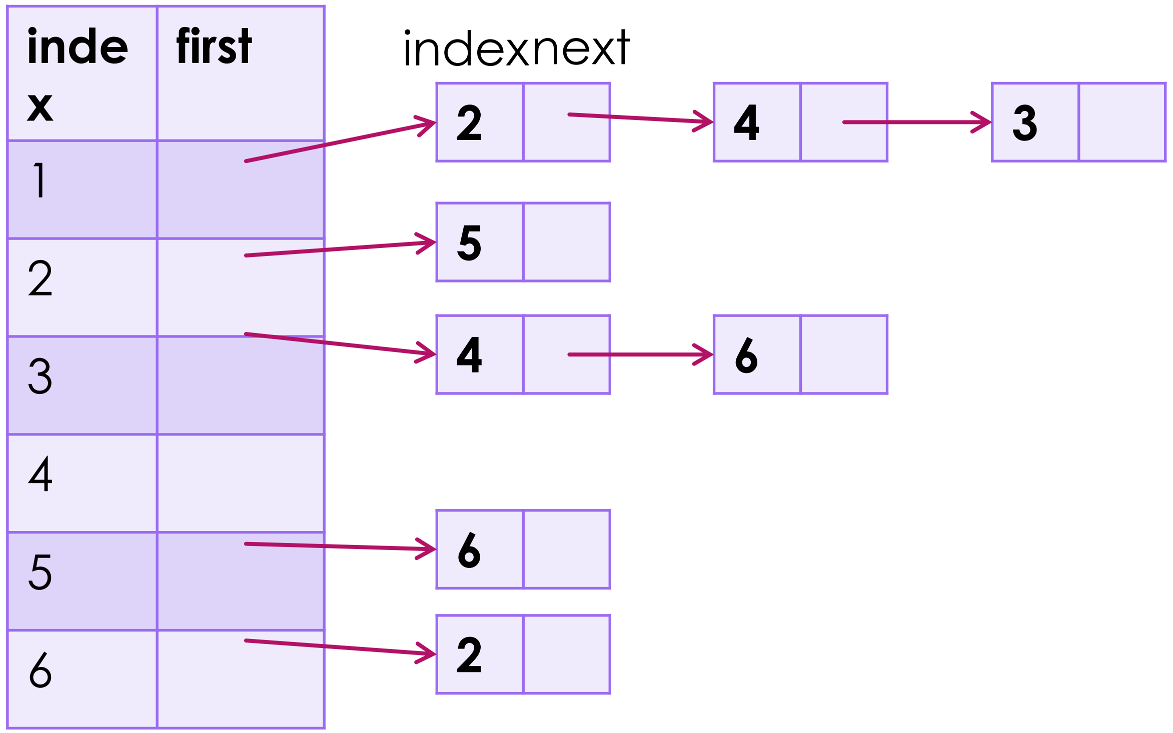


Problem B:BFS

BFS: for example 1 6

1. queue: 1 1 dequeue, 2 4 3 enqueue
2. queue: 2 4 3
3. queue: 4 3 5 2 dequeue, 5 enqueue
4. queue: 3 5 4 dequeue
5. queue: 5 6 3 dequeue, 4 has been visited only 6 enqueue

Then we find the node 6, it can be break and print YES



BFS in graph

A pseudo-code in P.3 in DAAS LAB7
we can make a little change

```
long BFS(Node T){
    queue.init();
    queue.add(T);
    max = T.getSum();
    while(!queue.empty()){
        e = queue.remove();
        while(e.hasNextChild){
            child = e.getNextChild();
            child.setSum(e.getSum()+child.getSum());
            if (max < child.getSum()){
                max = child.getSum();
            }//end if
            queue.add(child);
        }//end while
    }//end while
    return max;
} //end BFS
```

→

```
Boolean BFS(Node T, Node goal){
    queue.init();
    queue.add(T);

    while(!queue.empty()){
        e = queue.remove();
        e.setVisited(); Attention !
        while(e.hasNextChild){
            child = e.getNextChild();
            if (child.equals(goal)){
                return true;
            }
            if (!child.isVisited()&&!queue.exist(child))//Attention !
                queue.add(child);
        }//end while
    }//end while
    return false;
} //end BFS
```

DFS in graph-2 ways

```
Boolean DFS(Node T, Node goal) {
        stack.init();

        T.setVisited();
        stack.push(T);  /* push the src node into the stack*/
        while(!stack.empty()) {

                e = stack.gettop();

                if (e.equals(goal)){

                        return true;

                }

                if (e.hasUnvisitedChild()){

                        child = e.getOneUnvisitedChild();

                      if (!child.isVisited()&&!stack.exist(child)){//Attention

                            child.setVisited();

                            stack.push(child);

                      }

                }else{

                    stack.pop();

                }

        }

        return false;
}
```

```
Boolean DFS(Node T, Node goal) {
        stack.init();
        stack.push(T);  /* push the src node into the stack*/
        while(!stack.empty()) {

                e = stack.pop();

                if (e.equals(goal)){

                        return true;

                }

                e.setVisited();

                while(e.hasNextChild()){

                      if (!child.isVisited()&&!stack.exist(child))//Attention

                            stack.push(child);

                }

        }

        return false;
}
```

Problem C: Description

There are C kinds of colors and n * m grids. Each grid was paint by one color. Grid u at position (x, y) and grid v at position (a,b) are connected if and only if (1) they have the same color, (2) (|x-a| = 0 and |y-b|=0) or (|x-a| = 1 and |y-b|=0) or (|x-a|=0 and |y-b|=1). In a n * m grids, it contains several connected grids. Now give you n, m, C and the color of each grid. Please print the number of connected regions in it.

Problem C: Connected Regions

- Key points:
- BFS/DFS according the requirement
- solution: every time to choose a unlabeled grid using BFS/DFS, if one neighbor satisfy the requirement, label it and continue search the neighbor's neighbor; if not, stop search in this branch.
- Please notice that search district do not out of the boundary
- Time complexity： $O(n * m)$

Problem C: Sample

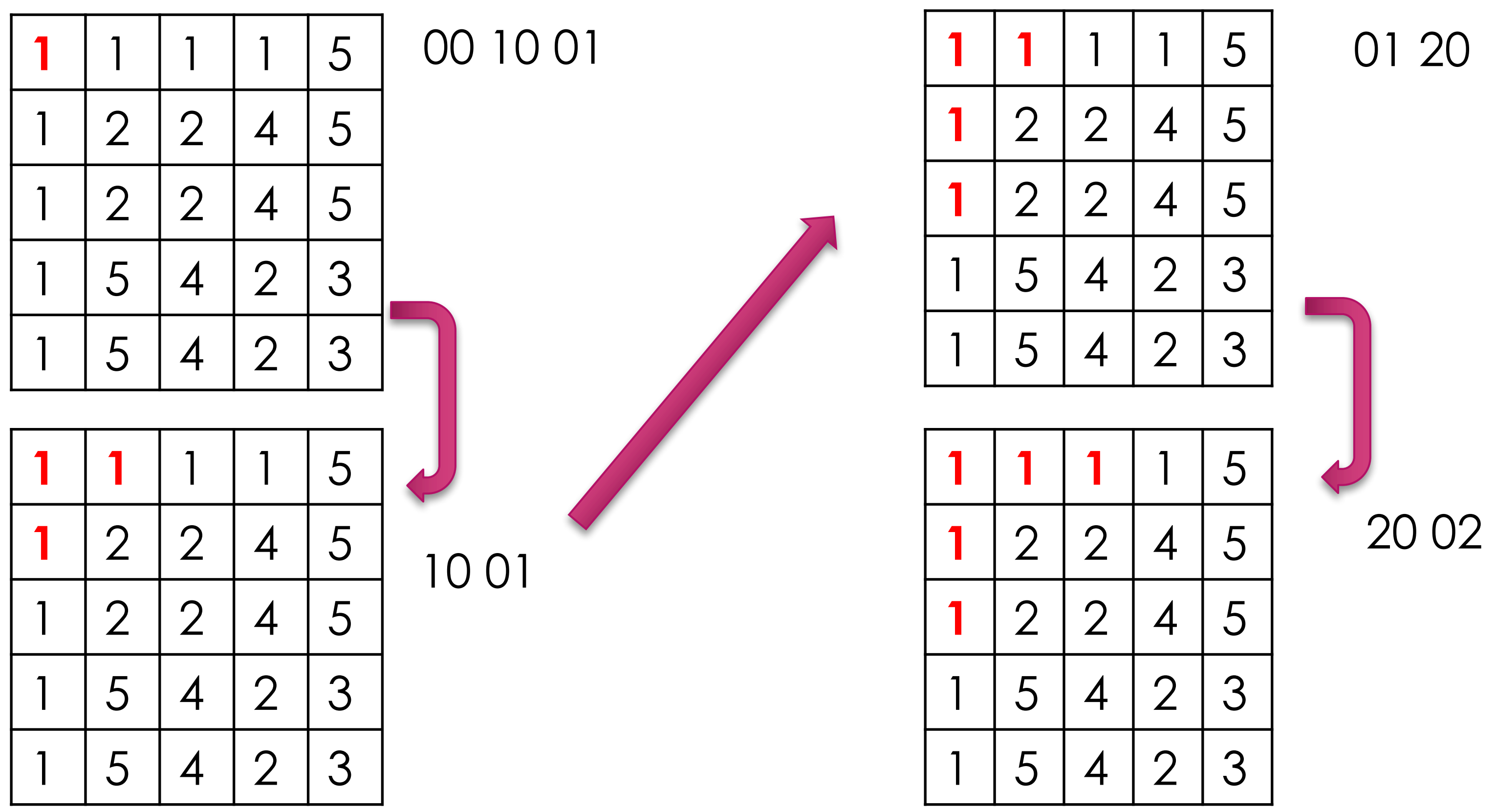
```
2
3 3 2
2 2 2
2 1 2
2 2 2
5 5 5
1 2 3 4 5
1 2 3 4 5
1 3 2 4 5
1 5 4 2 3
1 5 4 2 3
```

1	1	1	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

```
graph[row][col]
for(int i=0;i<row;i++){
    for(int j=0;j<col;j++){
         if(graph[i][j] haven't been visited){
                 find(i, j);
          }
    }
}
```

How to find ?
Down: graph[i-1][j]
Right: graph[i][j+1]
Up: graph[i+1][j]
Left: graph[i][j-1]
So we can regard the down, right, up and left as 4 adjacent nodes as the selected node

Problem C:BFS-1



1	1	1	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

00 10 01

1	**1**	1	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

10 01

1	**1**	1	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

01 20

1	**1**	**1**	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

20 02

Problem C:BFS-2

1	**1**	**1**	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

02 30

1	**1**	**1**	**1**	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

30 03

1	**1**	**1**	**1**	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

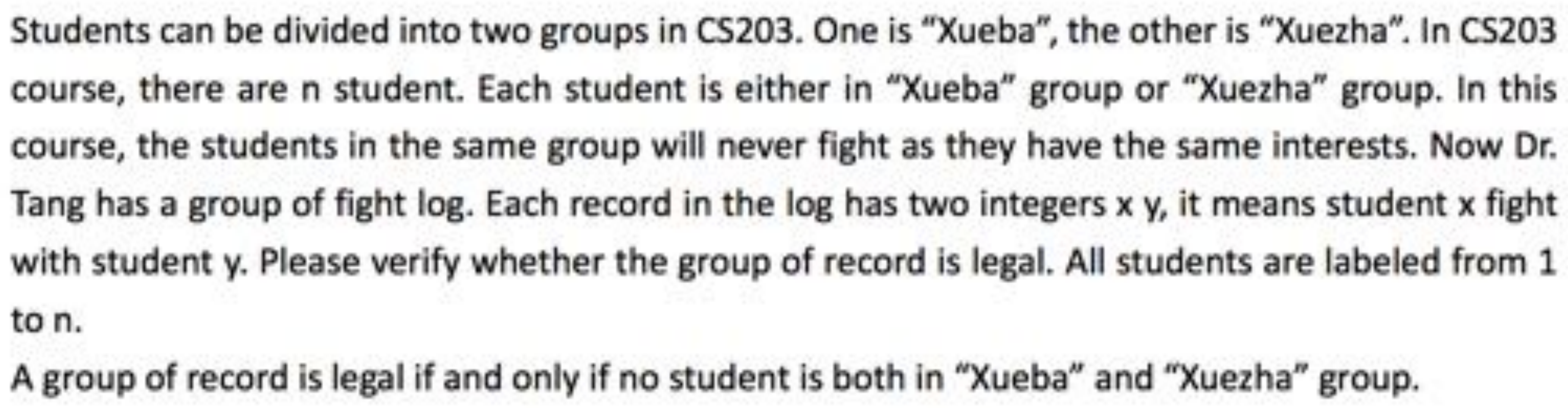
03 40

result:

1	1	1	1	5
1	2	2	4	5
1	2	2	4	5
1	5	4	2	3
1	5	4	2	3

{1 1 1 1 1 1 1 1 1}
{5 5 5}
{2 2 2 2}
{4 4}
{5 5}
{4 4}
{2 2}
{3 3}

Problem E: Description



Students can be divided into two groups in CS203. One is “Xueba”, the other is “Xuezha”. In CS203 course, there are n student. Each student is either in “Xueba” group or “Xuezha” group. In this course, the students in the same group will never fight as they have the same interests. Now Dr. Tang has a group of fight log. Each record in the log has two integers x y, it means student x fight with student y. Please verify whether the group of record is legal. All students are labeled from 1 to n.

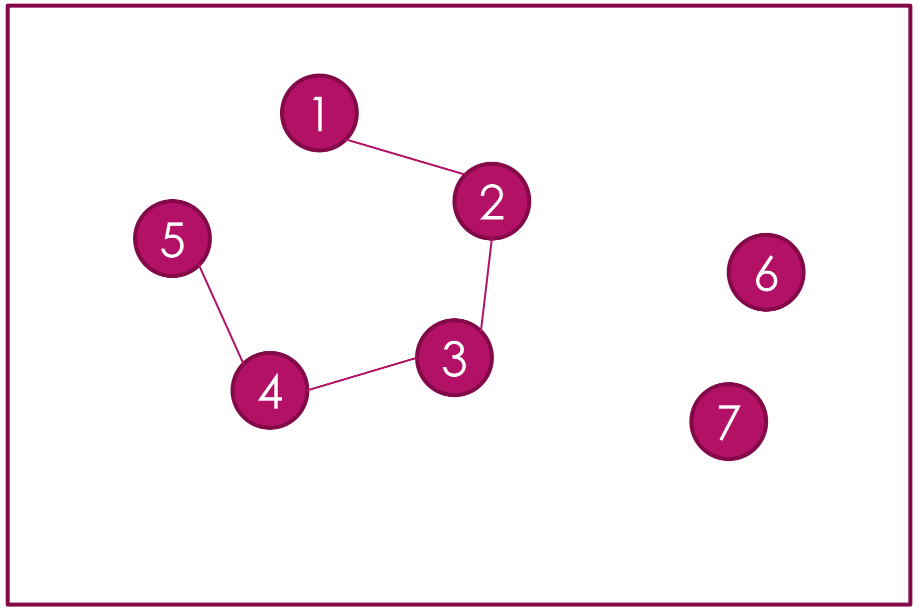
A group of record is legal if and only if no student is both in “Xueba” and “Xuezha” group.

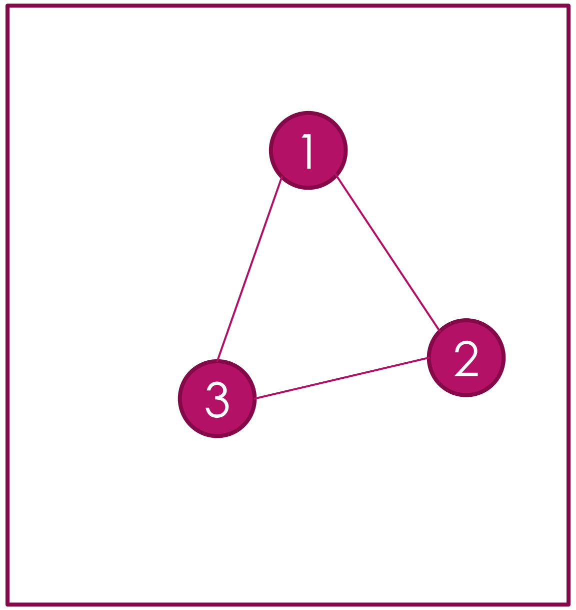
Problem E: Students in CS203

- Key points:
- It is a bipartite graph problem
- Judge if given edges can construct a bipartite graph
- Solution：Two-coloring. Each node is given the opposite color to its parent in the search forest, If, when a node is colored, there exists an edge connecting it to a previously-colored node with the same color, return false. If the algorithm terminates and all nodes can be correctly colored, return true. You can use BFS or DFS to find nodes' children and children's children and so on.
- Please notice that it is a undirected graph.
- Time complexity：$O(n)$

Problem E： Sample

2
7 4
1 2
2 3
3 4
4 5
3 3
1 2
2 3
3 1



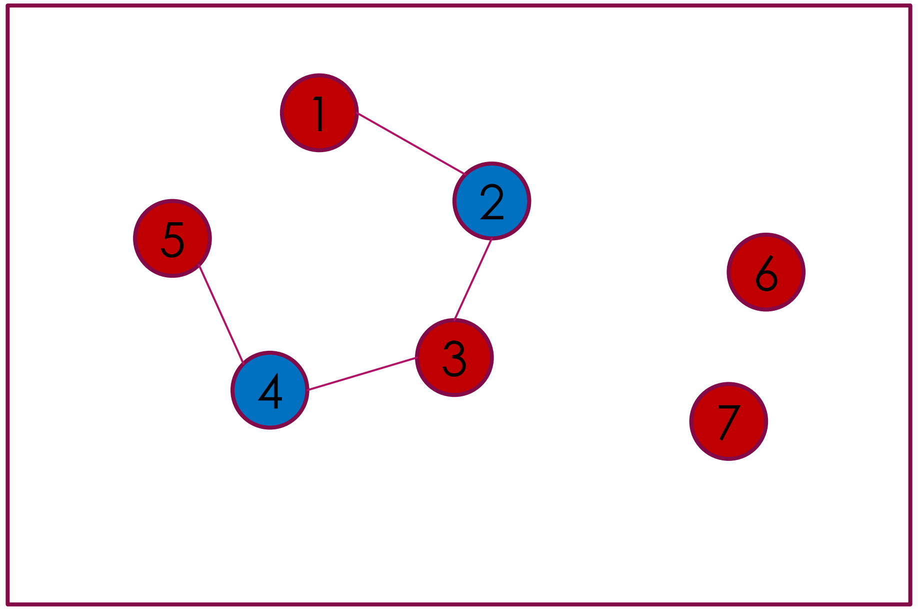


Two-coloring-sample1

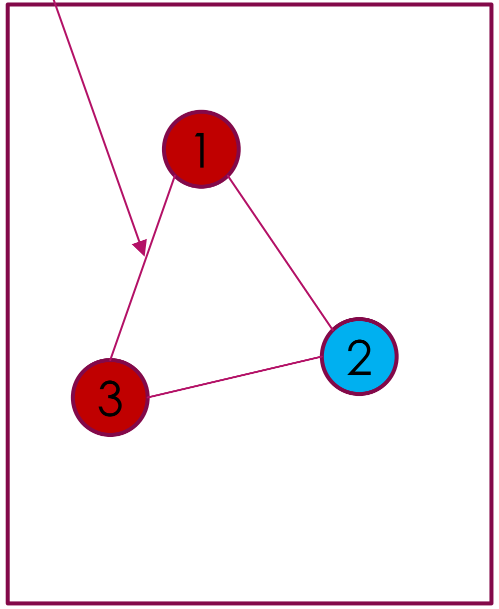
- choose two color to coloring the nodes.

  (1:red 2:blue)

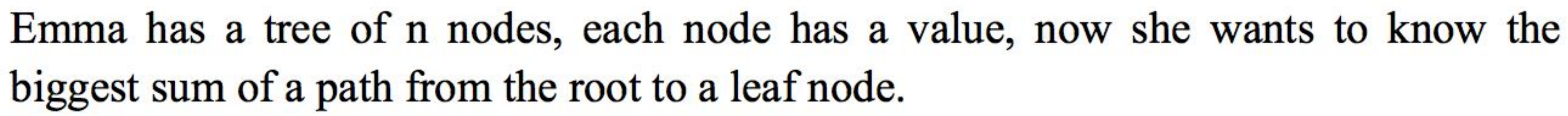
- Init all node uncolored(0)
- choose one uncolored node.(suppose node 1) Set it to 1(red)
- BFS or DFS to find all the node's children, set them to 2(blue)
- find the children's children, if they have no color, set them to 1(red); if they have color before and deferent from the color you want to set, return false.
- repeat until all node be colored, return true



node 3 wants to color node 1 to blue. but find that node is already red, so return false



Others: BFS a tree



Emma has a tree of n nodes, each node has a value, now she wants to know the biggest sum of a path from the root to a leaf node.

Problem A: Find the most valuable path

- Key point:
  - How to construct a tree
  - BFS a tree
- Please notice that:
  - the first node is not always the root node. You can use a flag to label whether a node is root or not.
  - A parent node may have more than 2 children.
  - A node’s value is not equals to this number, notice the range of the value. The sum of a path should be stored in long type.
- Time complexity： O(n)

BFS pseudo code

```
class Node{
    long sum;
    public setSum{…};
    public getSum{…};
}
```

init all nodes' sum = input value

```
long BFS(Node T){
    queue.init();
    queue.add(T);
    max = T.getSum();
    while(!queue.empty()){
        e = queue.remove();
        while(e.hasNextChild){
            child = e.getNextChild();
            child.setSum(e.getSum()+child.getSum());
            if (max < child.getSum()){
                max = child.getSum();
            }//end if
            queue.add(child);
        }//end while
    }//end while
    return max;
} //end BFS
```

Level traversal a tree

- Key point:
- How to construct a tree
- BFS a tree
- Please notice that the first node is not always the root node. You can use a flag to label whether a node is root or not.
- Pseudo-code about lever traversal can be found on P24 in Week 09.pdf(Prof.Luo)
- Time complexity： O(n)

How to construct a tree?

- For this problem, we can use adjacency list to store a tree
- You can define a tree like this in JAVA:

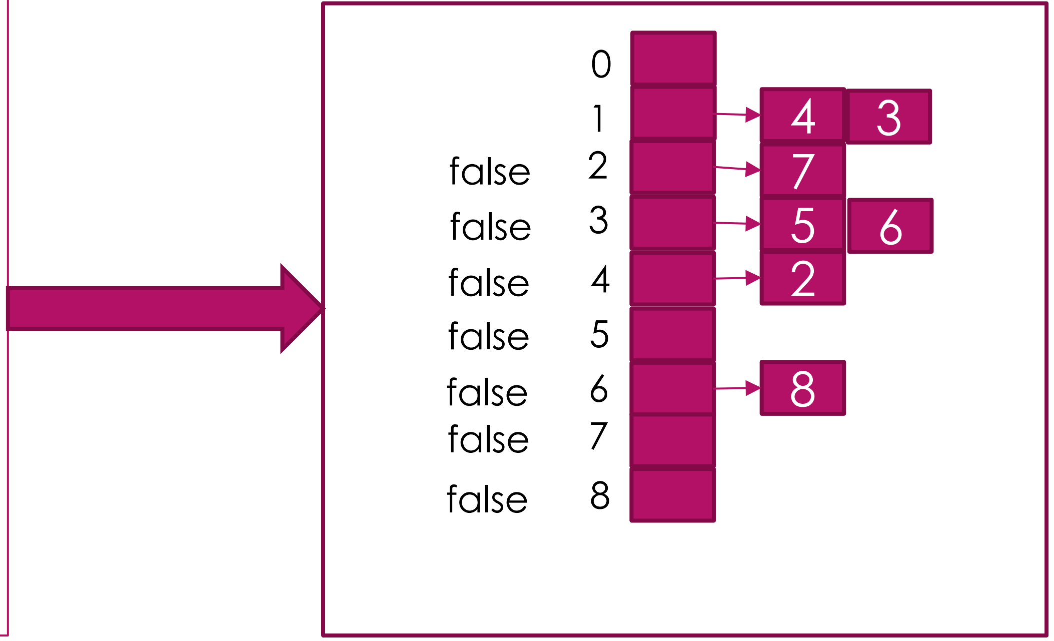
```
public class Btree{
    ArrayList<ArrayList<Integer>> nodes;
    ArrayList<Boolean>          isRoot;
}
```

- Another define:

```
public class Btree{
    ArrayList<Node> nodes;
}
class Node{
  Boolean  isRoot;
  int          leftChild;
  int          rightChild;
}
```

How to construct a tree? – A Sample

```
8
1 4
1 3
4 2
2 7
3 5
3 6
6 8
```



From above graph, we can easy find out which one is root node. Root node in a tree is the only node which in-degree is zero.

Finding the diameter of a binary tree

- Choose any node in a tree, set the node u.
- Find the farthest node from u, set this farthest node s(BFS).
- Find the farthest node from s, set this farthest node t(BFS).
- The distance between s and t is the longest distance between any nodes in the tree

How to calculate distance when BFS

```
void BFS(Btree T){
    queue.init();
    queue.add(T);
    T.setDist(0);
    while(!queue.empty()){
        e = queue.remove();
        if (e.getLchild()!=null){
            queue.add(e.getLchild());
            e.getLchild().setDist(e.getDist()+1);
        }
        if (e.getRchild()!=null){
            queue.add(e.getRchild());
            e.getRchild().setDist(e.getDist()+1);
        }
    }
}
```