# 数据结构与算法

## DATA STRUCTURE

第二十讲 图（二）

胡浩栋

信息管理与工程学院

2018 - 2019 第一学期

# 课堂内容

- DFS应用
- BFS

# 回忆DFS递归算法

**DFS**(Node u)

u.status =

**foreach** （v是u的邻接顶点）

if v.status ==

DFS(v)

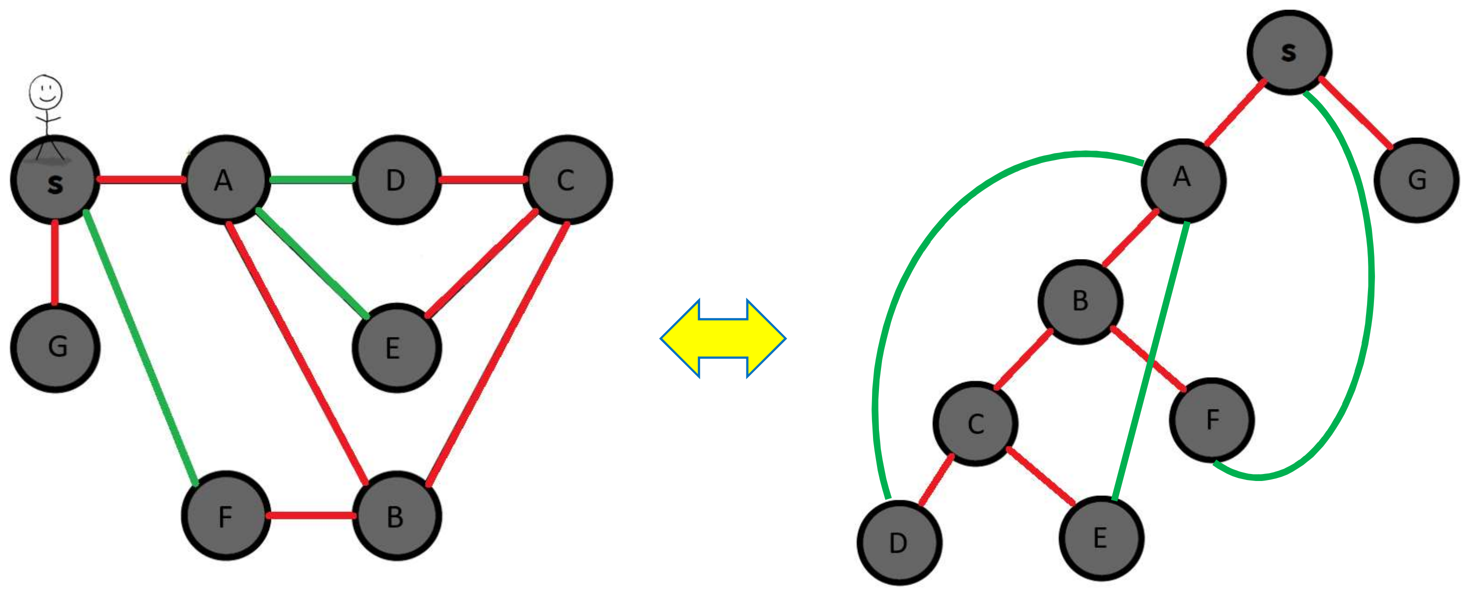
u.status =

**Runtime:** O( |V|+|E| )

```
void MyGraph::Dfs(int start)
{
    cout << "\n Start DFS search:" << endl;
    bool *visited = new bool[_nodes.size()]();
    DfsInternal(start, visited);

    cout << endl;
    delete [] visited;
}

void MyGraph::DfsInternal(int uid, bool *visited)
{
    visited[uid] = true;
    cout << "#" << uid << ", ";

    for (auto arc : _nodes.at(uid)->arcs)
    {
        int vid = arc->V->id;
        if (!visited[vid])
        {
            DfsInternal(vid, visited);
        }
    }
}
```

# 为什么叫depth first

- 算法过程隐含了一个DFS树




- 注意在有向图里，不同的起始节点有不同的dfs树，甚至dfs森林
- 一个事实是在节点的dfs子树里的子节点都是可达的

# 回忆DFS算法+始末时间

**DFS**(Node u, Time currentTime)

u.start = currentTime
currentTime += 1
u.status =
**foreach** （v是u的邻接顶点）
if v.status ==
currentTime = **DFS**(v, currentTime)
currentTime +=1
u.end = currentTime
u.status =
**return** currentTime

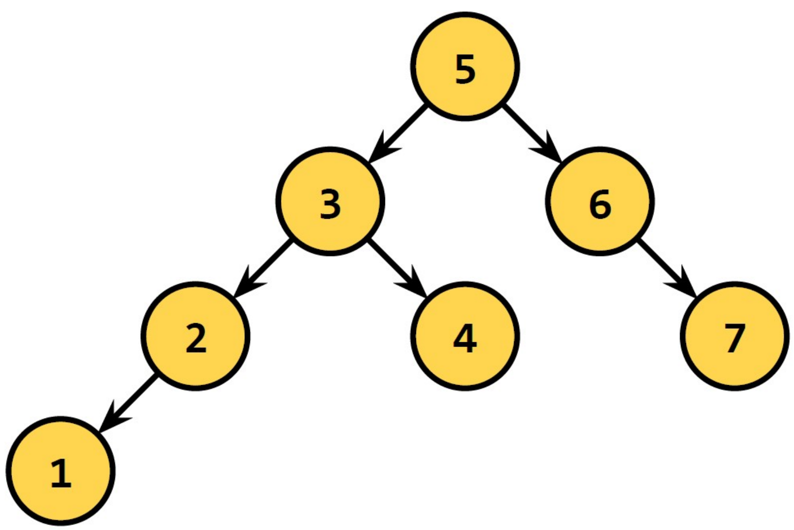
```
void MyGraph::DfsTime(int start)
{
    cout << "\n Start DFS Time search:" << endl;
    bool *visited = new bool[_nodes.size()]();
    pair<int, int> *travelTimes = new pair<int, int>[_nodes.size()];

    DfsTimeInteral(start, 0, visited, travelTimes);

    delete [] visited;
    delete [] travelTimes;
}

int MyGraph::DfsTimeInteral(int uid, int currentTime,
                            bool *visited, pair<int, int> *travelTimes)
{
    visited[uid] = true;
    travelTimes[uid].first = currentTime++;
    cout << "#" << uid << ", ";

    for (auto arc : _nodes.at(uid)->arcs)
    {
        int vid = arc->V->id;
        if (!visited[vid])
        {
            currentTime = DfsTimeInteral(vid, currentTime, visited, travelTimes);
            currentTime++;
        }
    }

    travelTimes[uid].second = currentTime;
    return currentTime;
}
```

# DFS应用

- DFS算法过程能找到所有能从起始点可达的节点
- 在无向图里这个能达到的所有节点的集合就是连通分支
    - 找出所有的连通分支（当作无向图运行dfs）
    - BST排序
    - 拓扑排序topology ordering
    - 找强连通分支strongly connected components

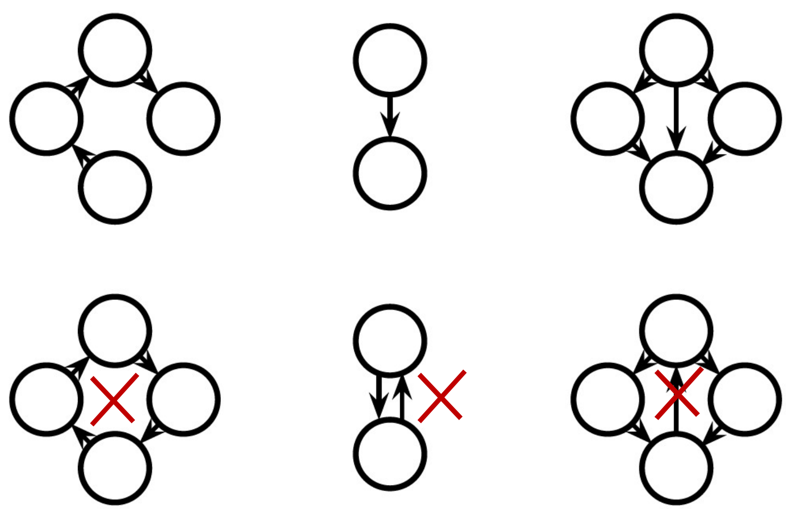
# DFS应用例子

- 给BST的元素排序
- 用DFS对BST中序遍历



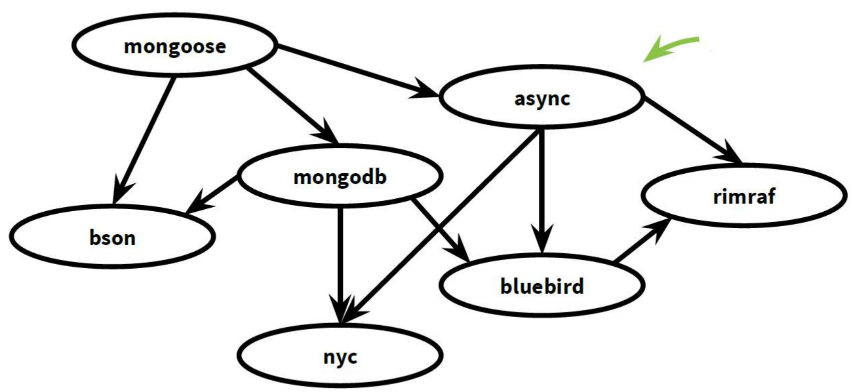
# 有向无环图

- Directed Acyclic Graph（DAG）
- 即有向图里不存在起点和终点一样的路径
- 在DAG上我们可以进行topology ordering



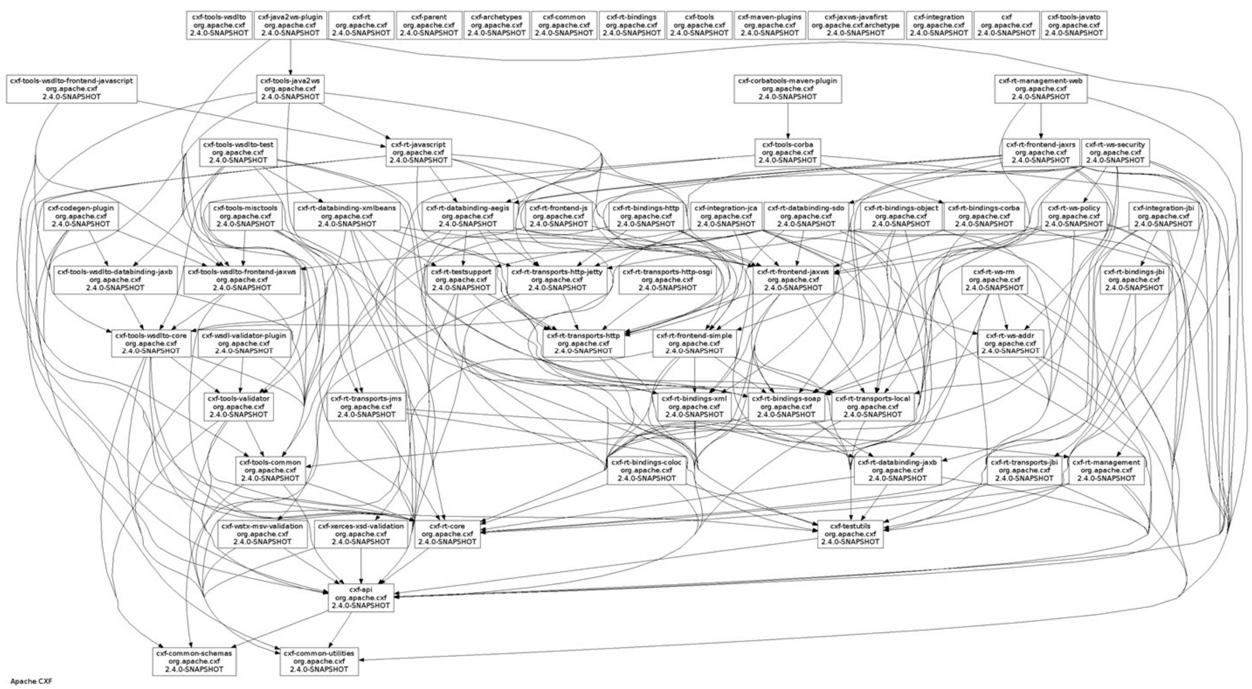
# 例子

- 比如软件安装的依赖关系如图，那么应该按什么顺序安装
- 依赖关系图是DAG，因为不会有环路
- 这个可以肉眼识别



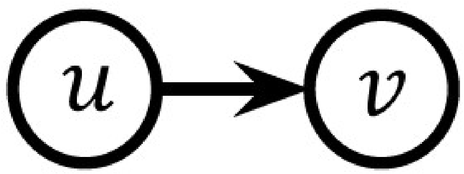
# 例子

- 大型项目的编译依赖关系更复杂
- 肉眼识别不可能，可以应用**topology ordering**



# 什么是topology ordering

- 定义：DAG里，如果$(u, v) \in E$，那么$u$的topology ordering应该排在$v$之前
- 因为没有cycle，这个关系是可传递的，即$u$和$v$是可达的，那么$u$排在$v$之前



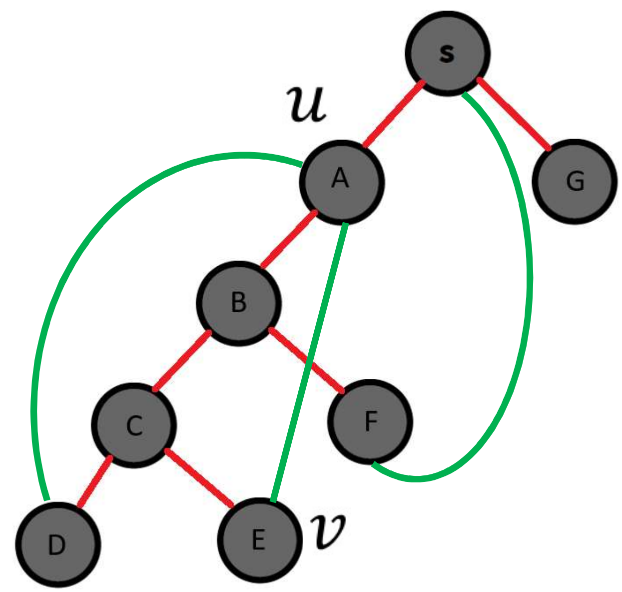

- Claim：在DFS+始末时间算法里，$u$的结束时间 > $v$的结束时间
- 结论：DFS算法可以实现topology ordering

# 一个事实 （即使有cycle也成立）

- 如果在DFS树里，$v$是$u$的子节点:

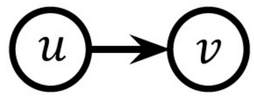

- 反过来的话:


- 如果互相不是子节点:

# 证明

- 在DAG里，如果$u$和$v$是可达的，那么$u$的结束时间 > $v$的结束时间



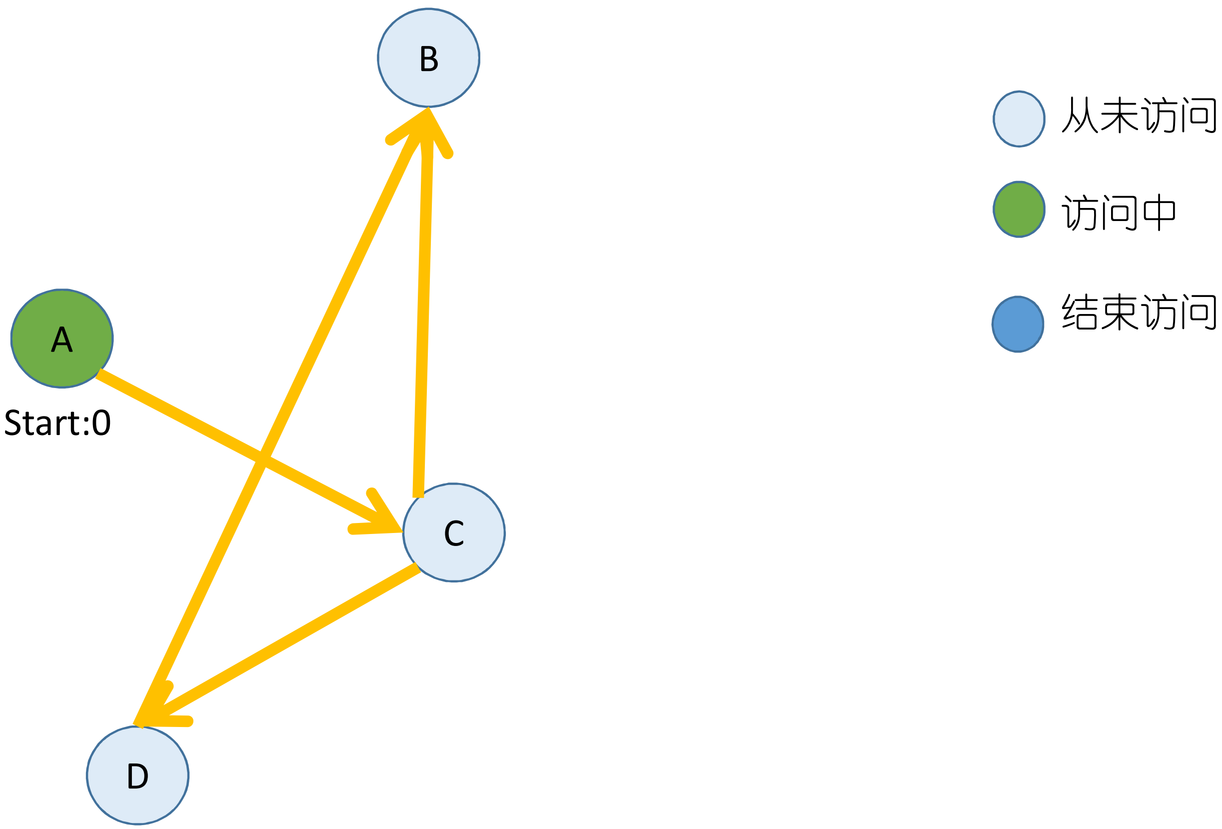

1) 如果先访问$v$，因为没有环路， $u$不可能在$v$的dfs子树里面。所以是当$v$结束后$u$还没开始。
2) 如果先访问$u$，那么$v$一定在$u$的dfs子树中

- 从而$u$的结束时间 > $v$的结束时间
- 所以按DFS访问的结束时间排序就是topology ordering

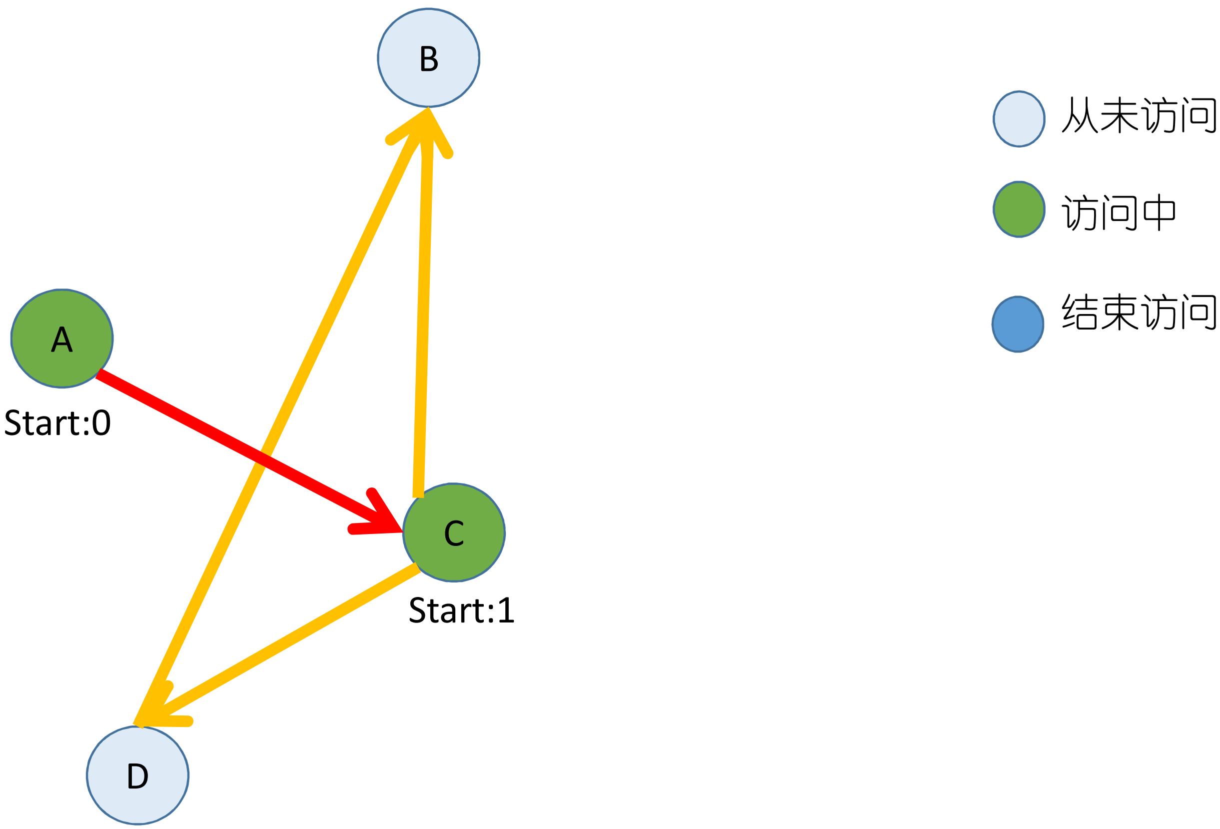
# 拓扑排序算法

```
TopologyOrdering(Node u, Time currentTime)
    u.start = currentTime
    currentTime += 1
    u.status =
    foreach （v是u的邻接顶点）
        if v.status ==
            currentTime = TopologyOrdering(v, currentTime)
            currentTime +=1
    u.end = currentTime
    u.status =
    TopologyList.push(u)
    return currentTime
```

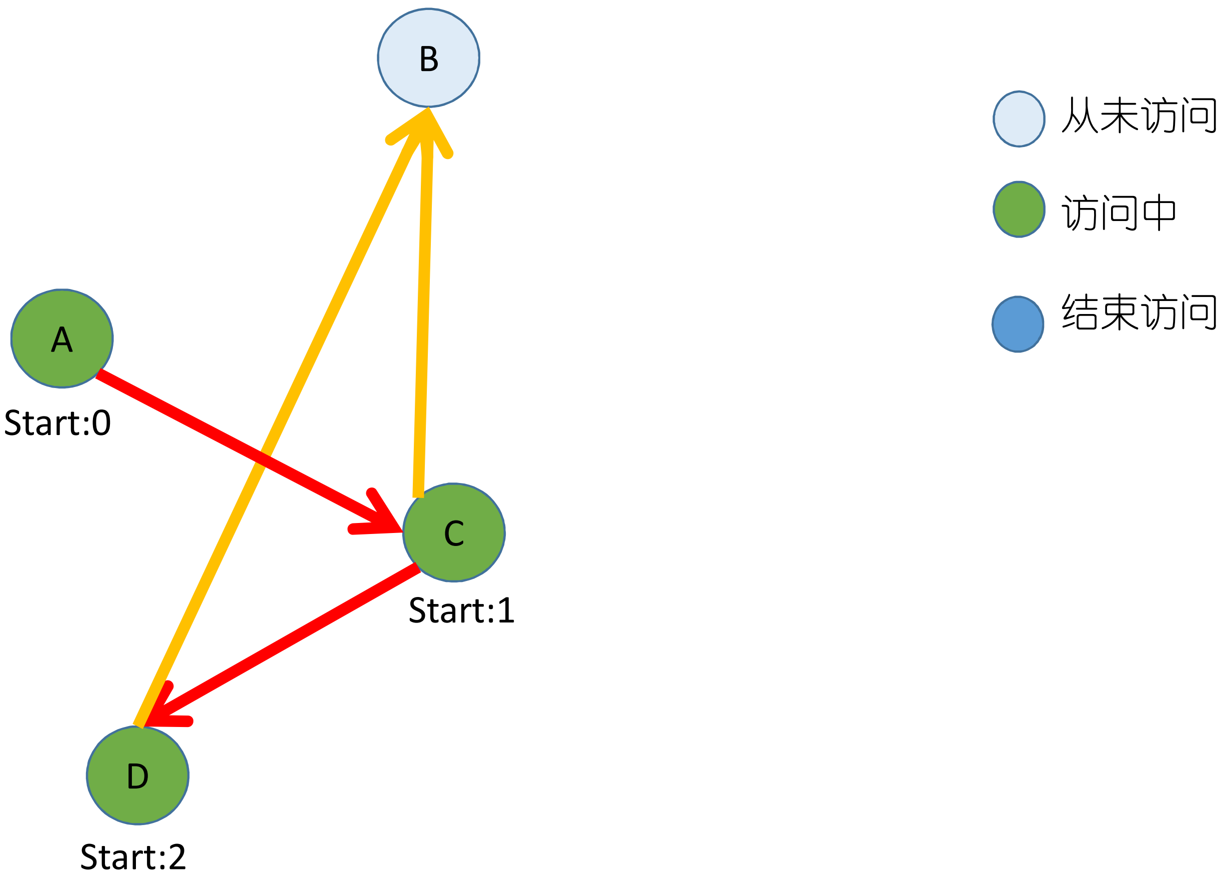
# 例子：



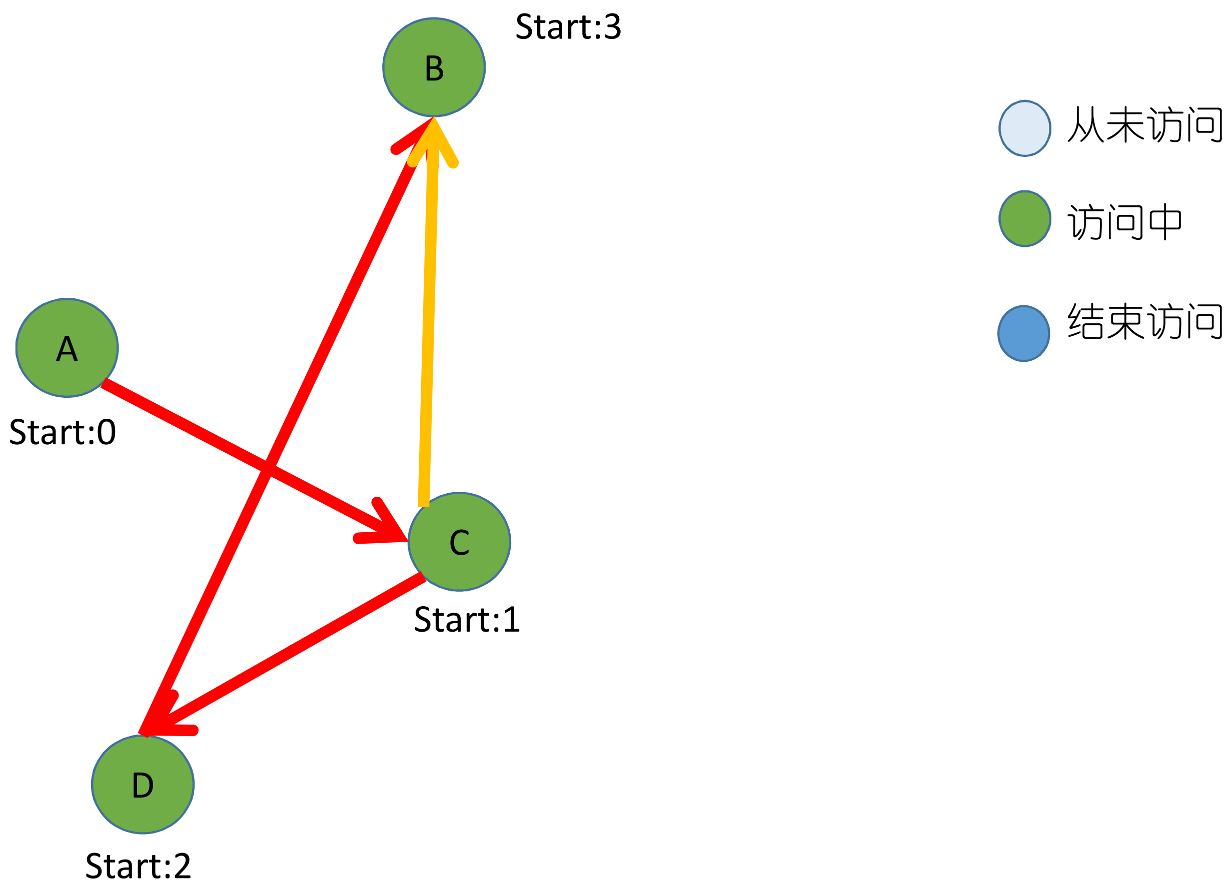
# 例子



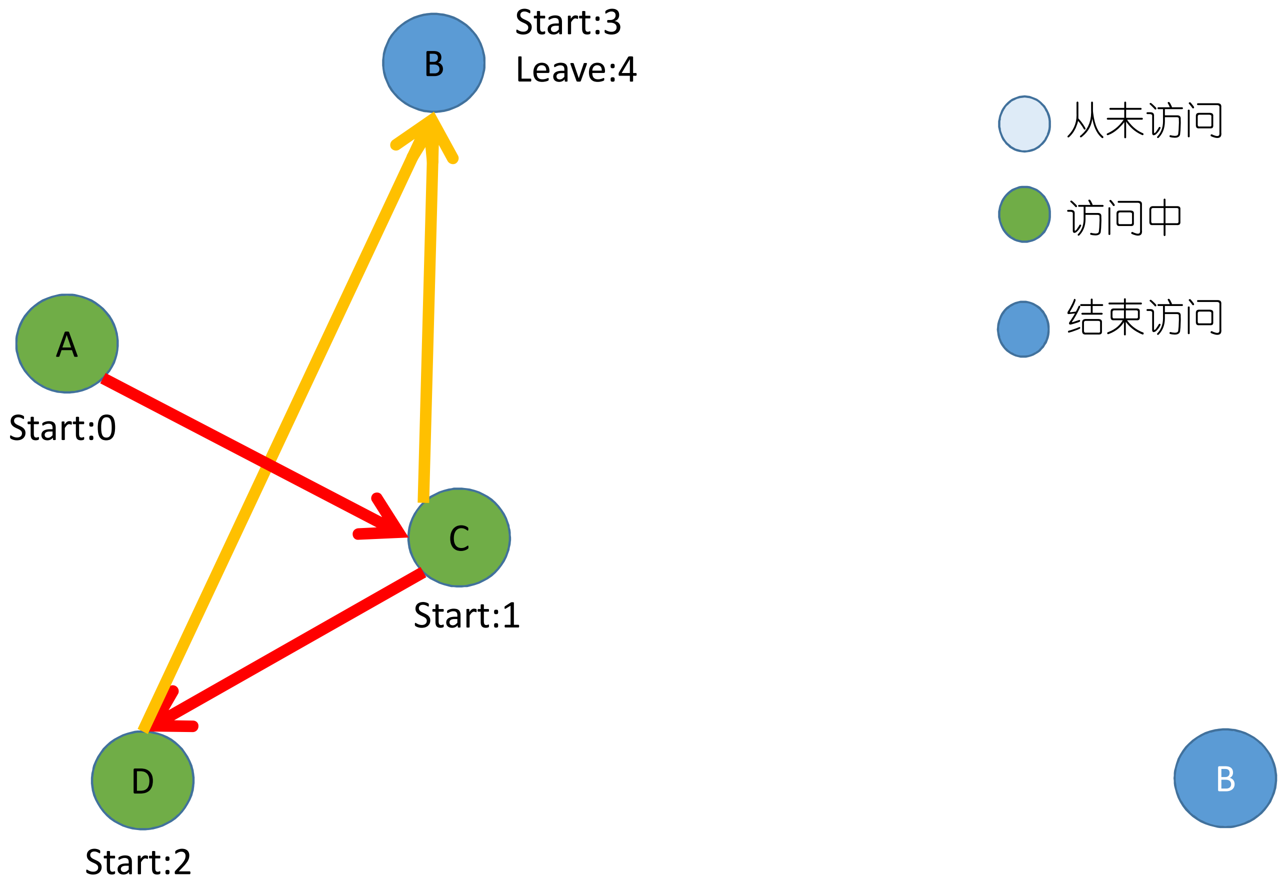
# 例子



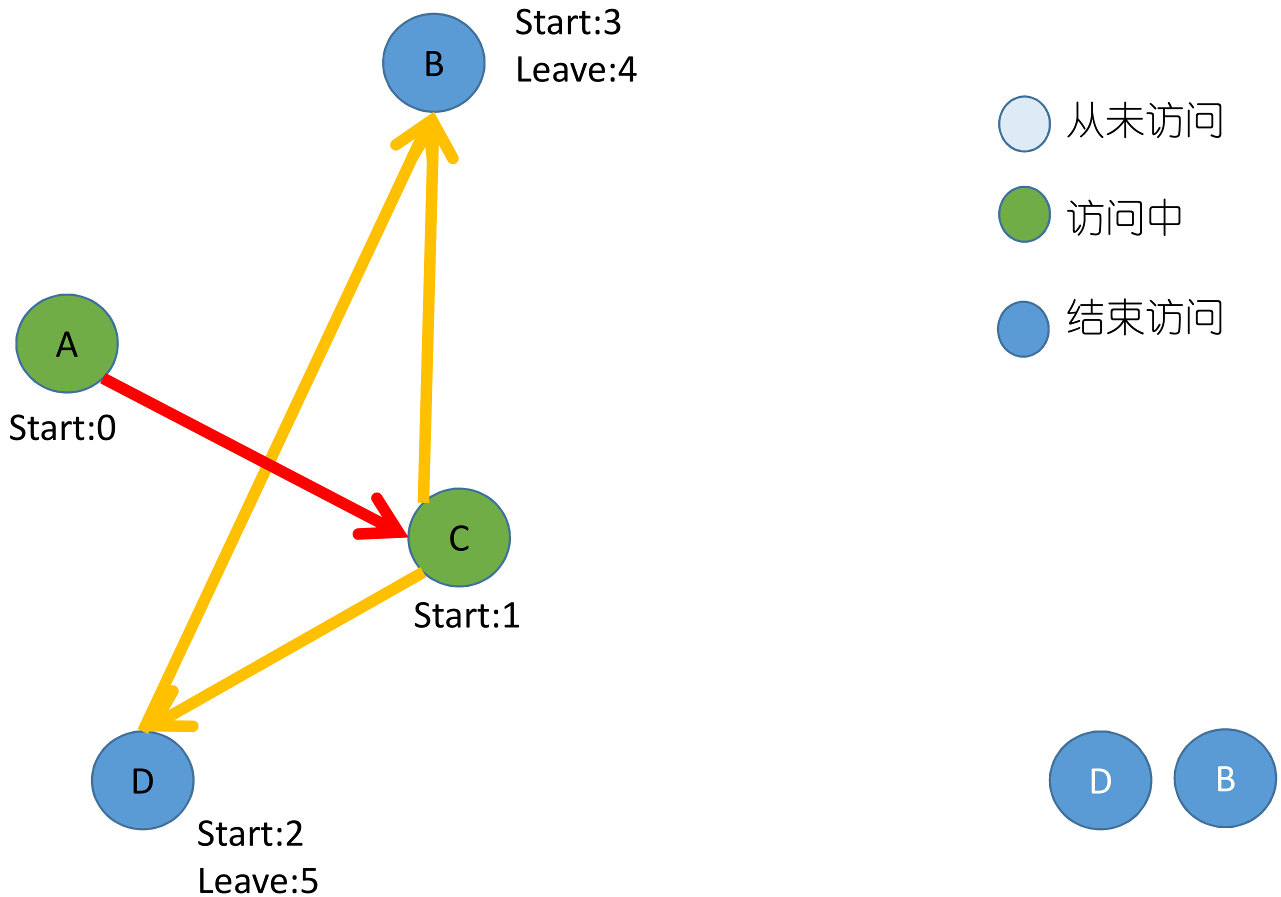
# 例子



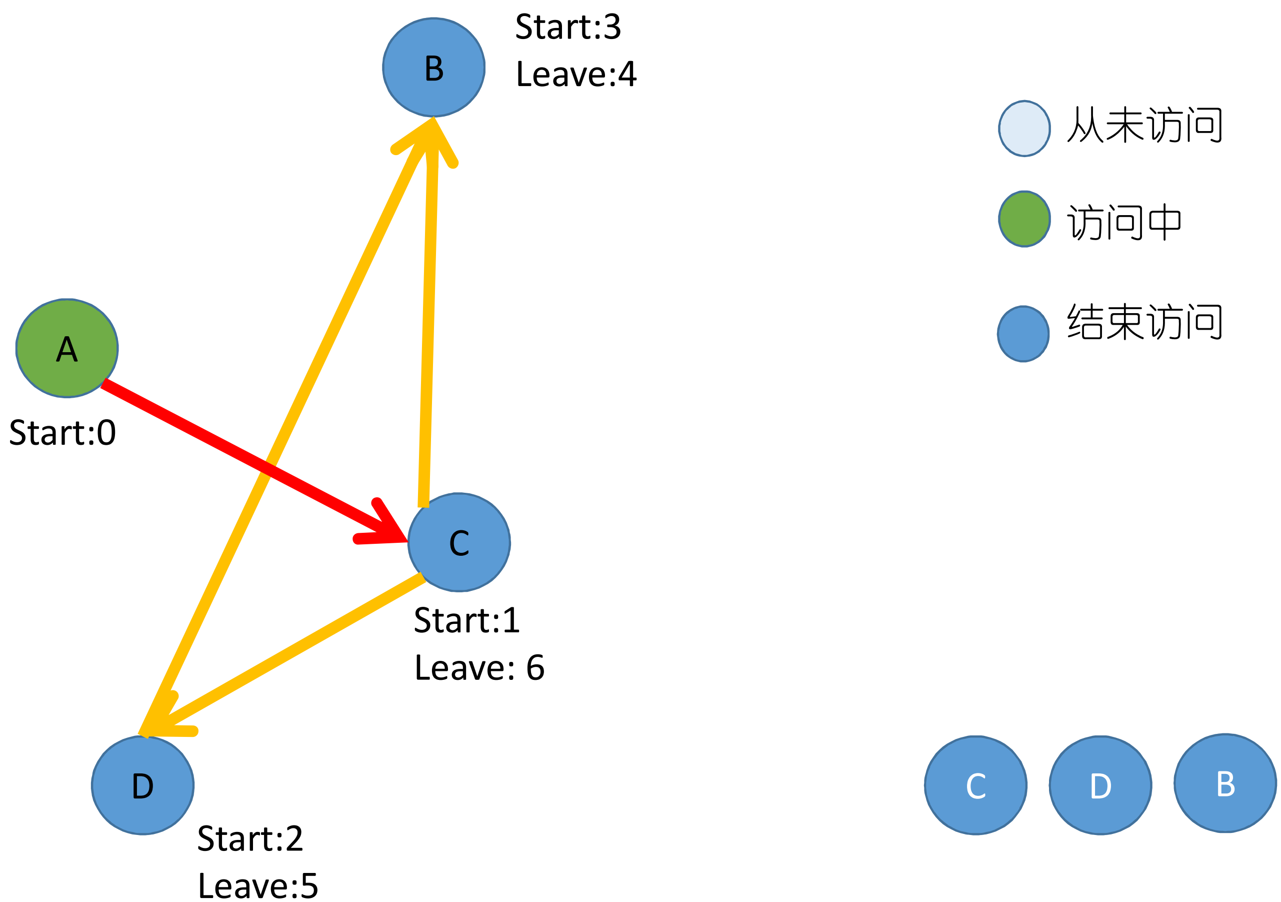
# 例子



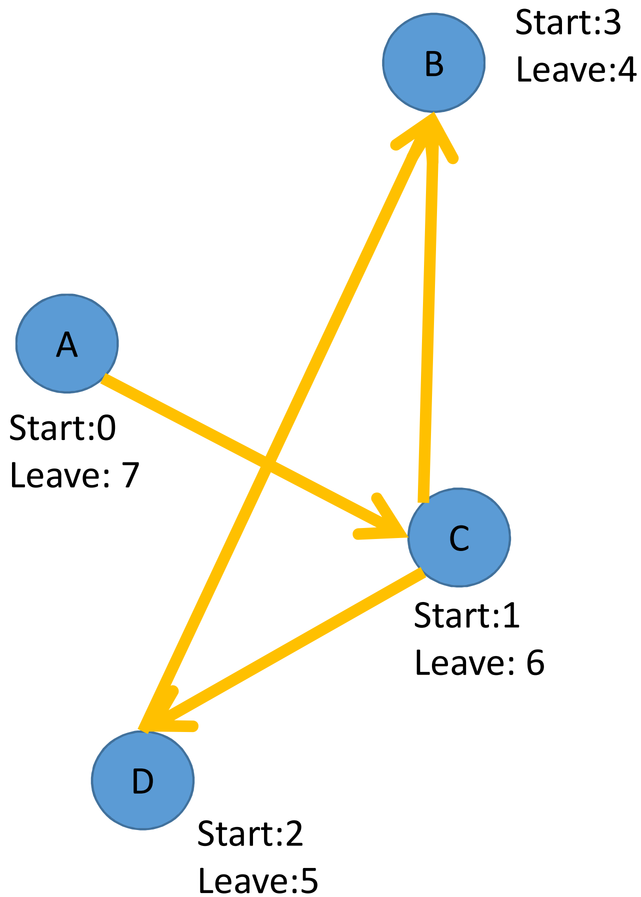
# 例子


B
Start:3
Leave:4
A
Start:0
C
Start:1
D
Start:2
Leave:5
从未访问
访问中
结束访问
D
B

# 例子

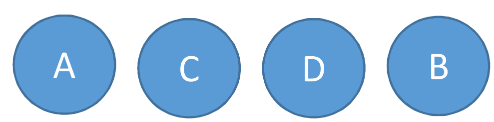

B
Start:3
Leave:4
从未访问
访问中
结束访问
A
Start:0
C
Start:1
Leave: 6
D
Start:2
Leave:5
C
D
B

# 例子


B
Start:3
Leave:4
A
Start:0
Leave: 7
C
Start:1
Leave: 6
D
Start:2
Leave:5

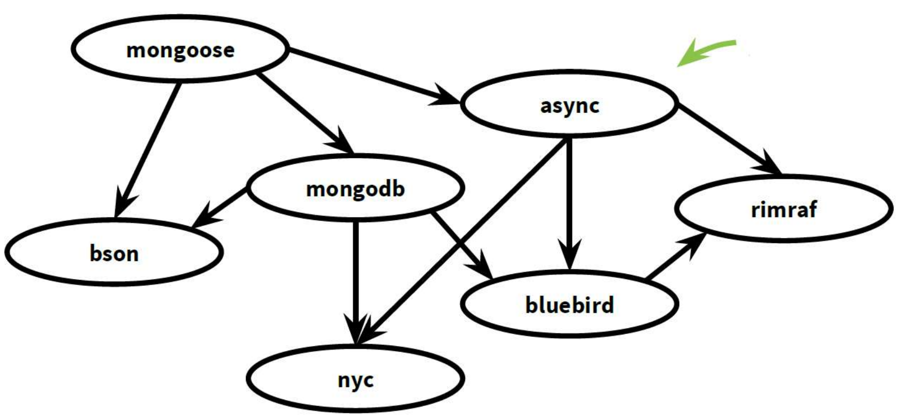

从未访问
访问中
结束访问

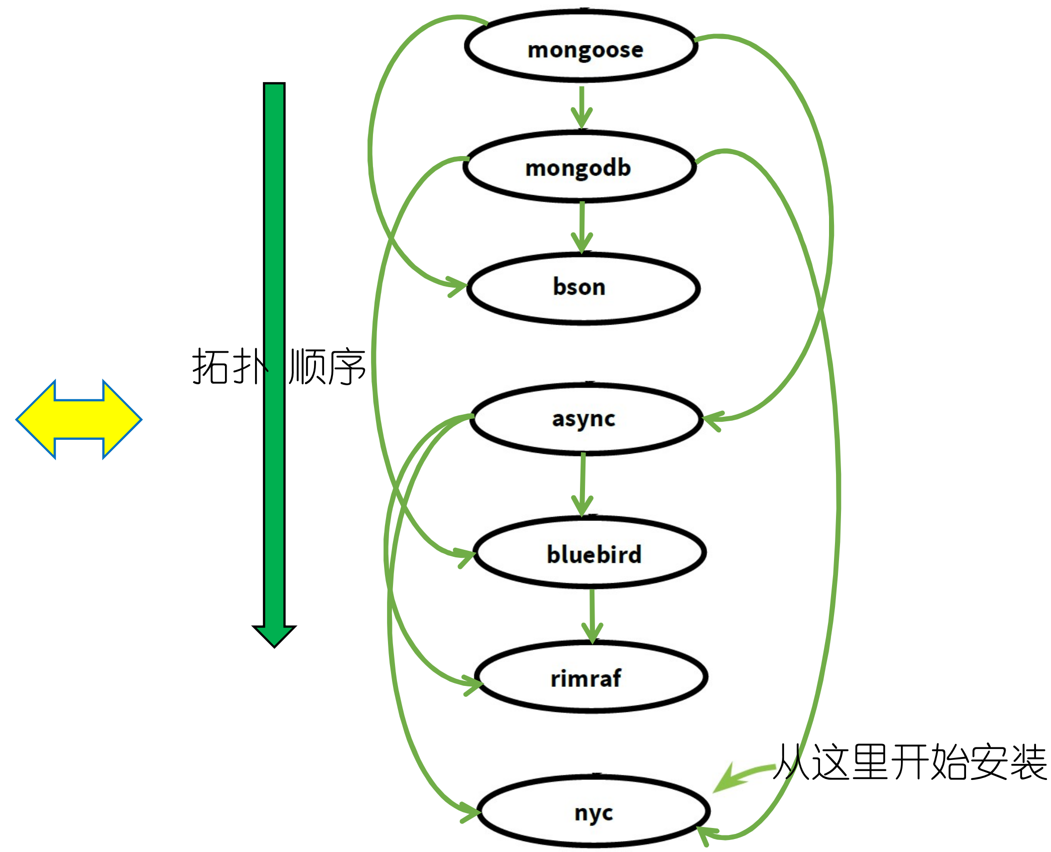
按结束时间减小:


A
C
D
B

# DFS应用例子

- 软件安装次序

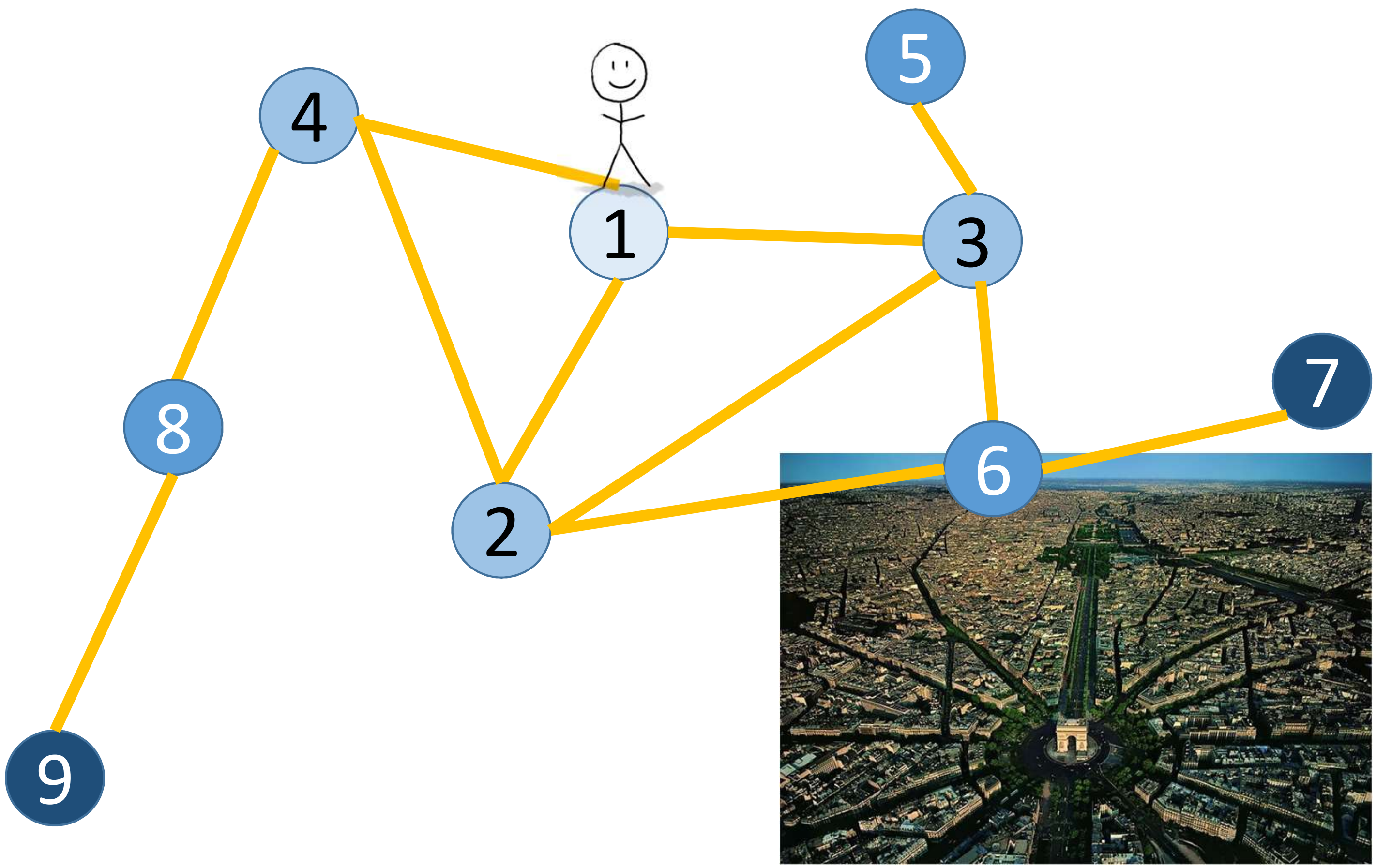


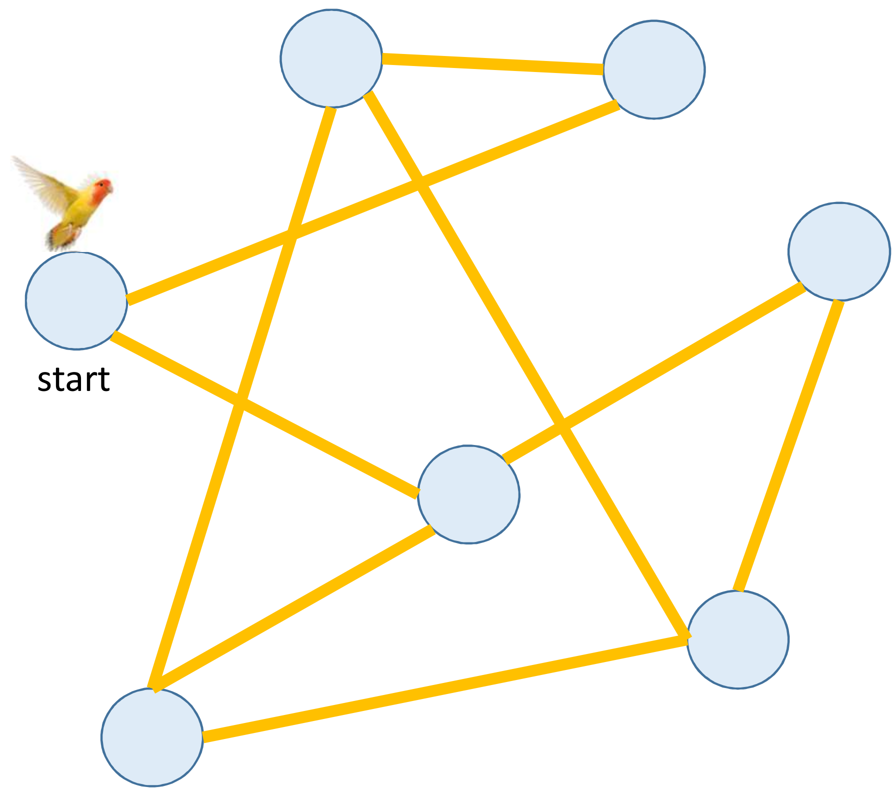
# 广度优先BFS

# How do we explore a graph?

无人机



# BFS从鸟瞰的角度探索图




从未访问

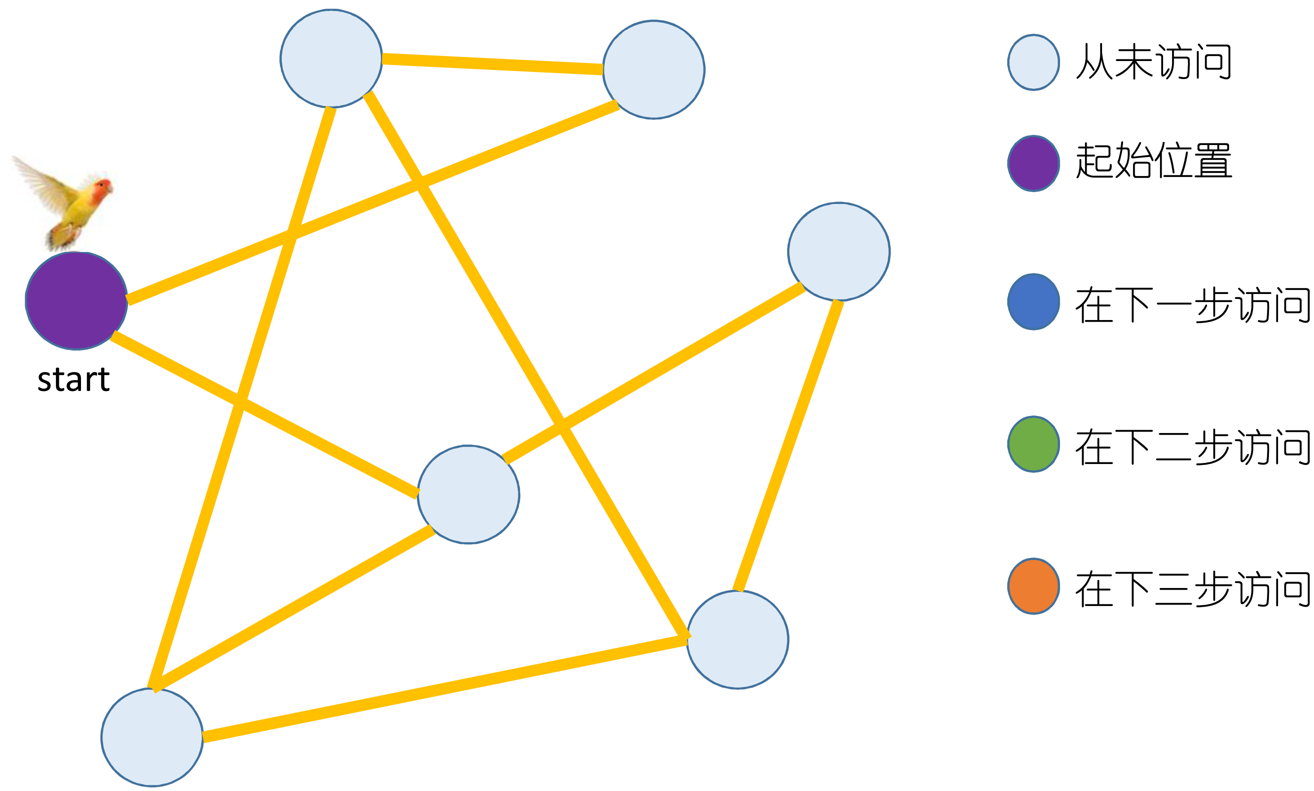
起始位置

在下一步访问

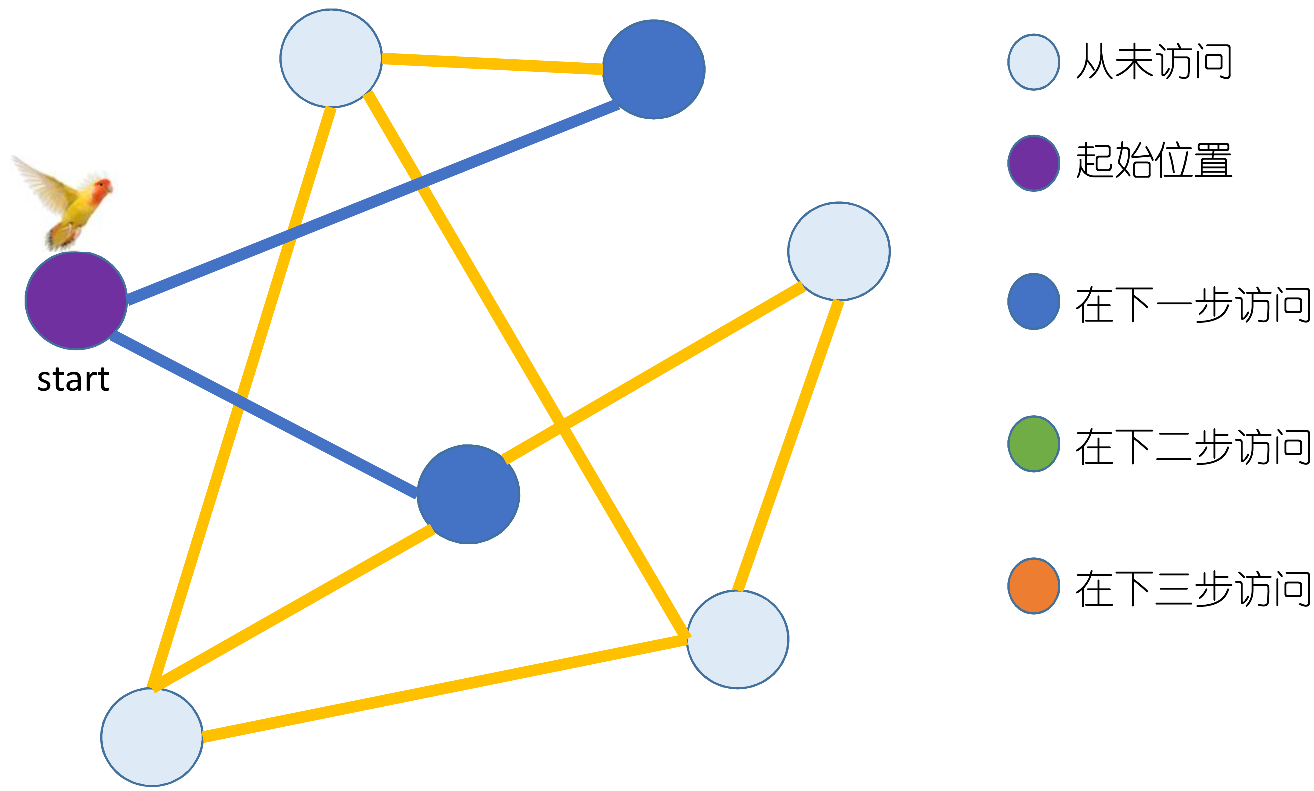
在下二步访问

在下三步访问

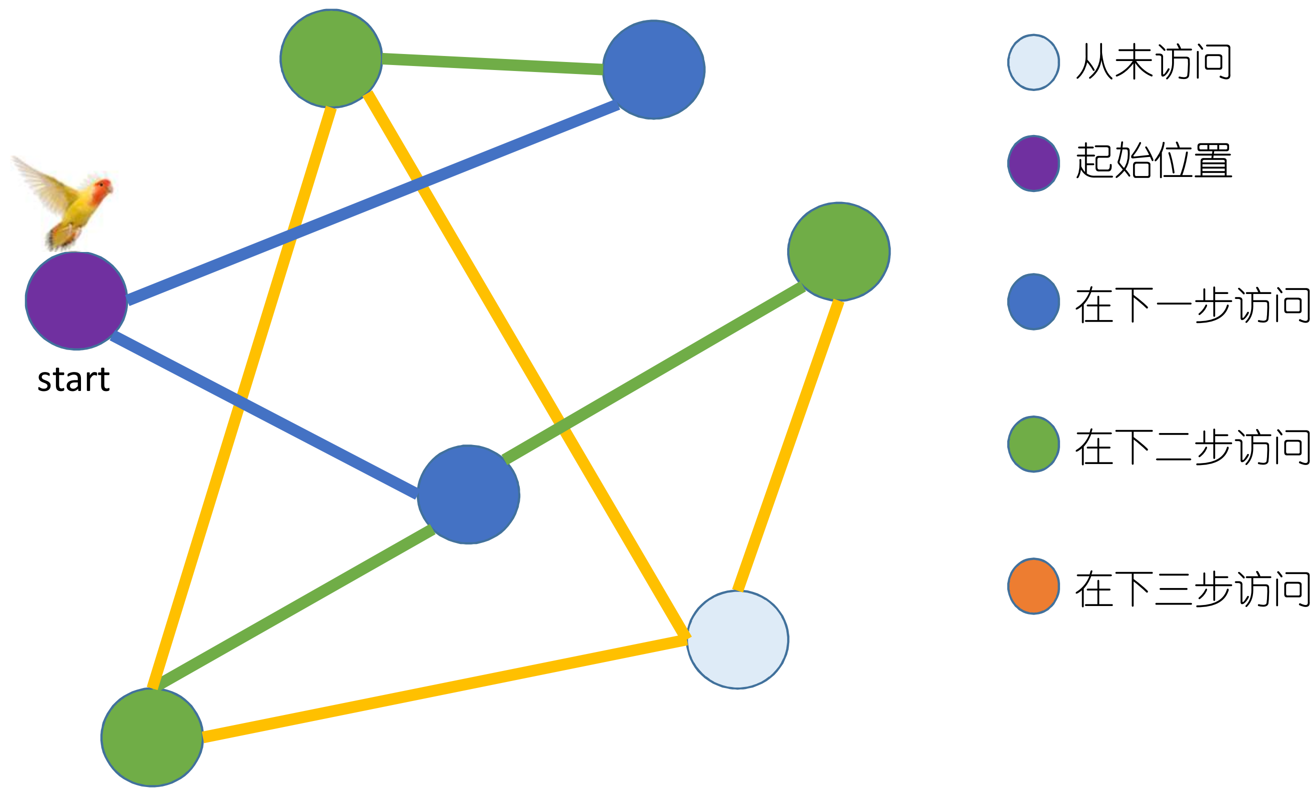
# BFS从鸟瞰的角度探索图



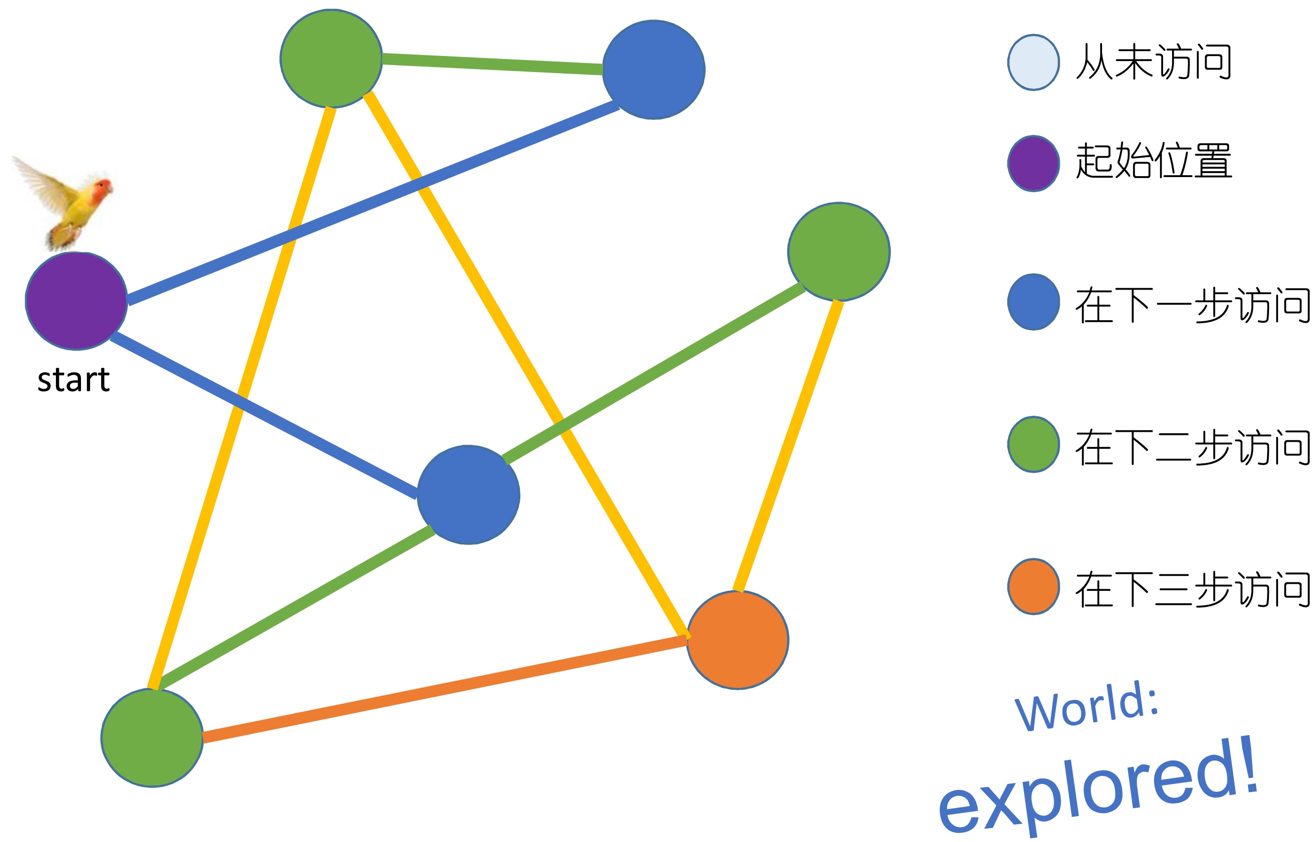
# BFS从鸟瞰的角度探索图



# BFS从鸟瞰的角度探索图



# BFS从鸟瞰的角度探索图



# BFS迭代算法

**BFS_Iterative**(Node u)

queue Q

Q.enqueue(u)

**while**（Q不空）

u = Q.deque()

**foreach**（v是u的邻接顶点）

if v.status == ○

v.status = ●

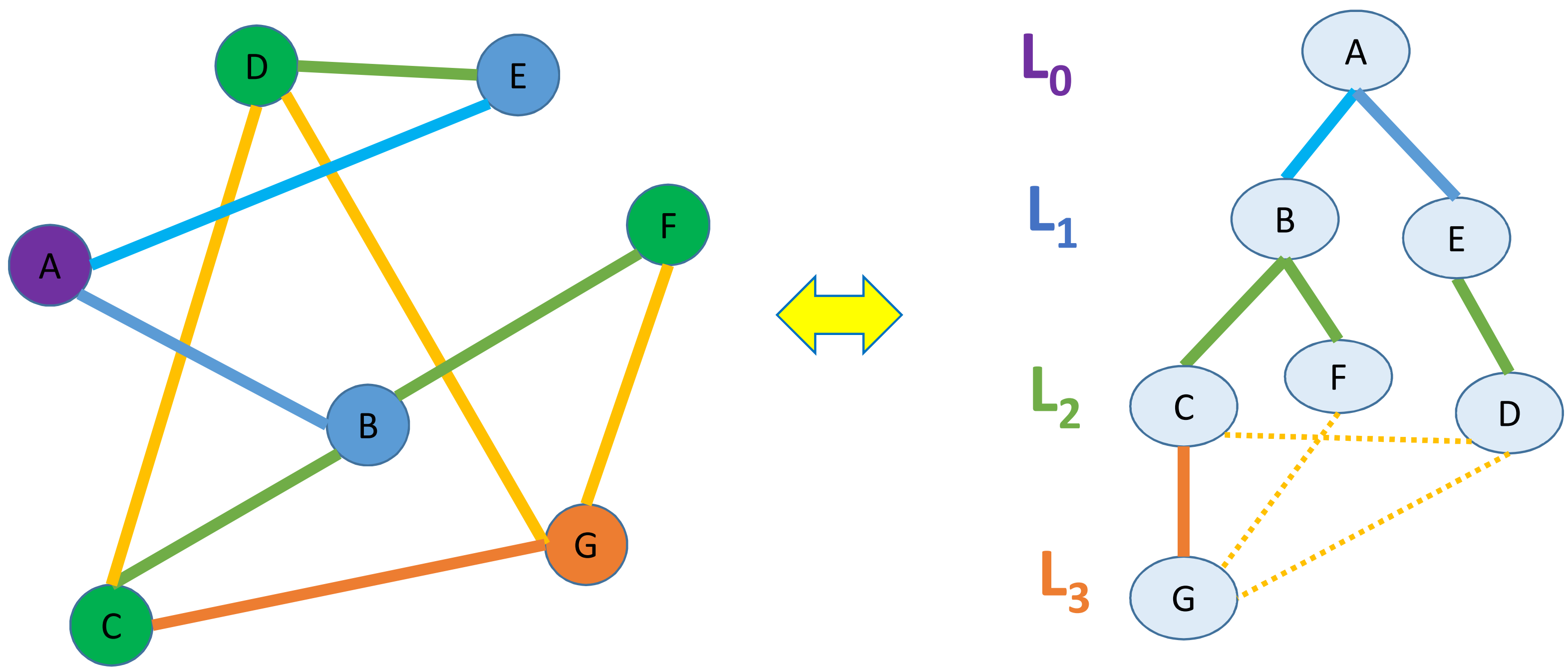
**Q.enqueue(v)**

```
void MyGraph::Bfs(int start)
{
    bool *visited = new bool[_nodes.size()]();
    cout << "\n Start BFS search:" << endl;

    queue<int> que;
    que.push(start);
    visited[_nodes.at(start)->id] = true;

    while(!que.empty())
    {
        Node *U = _nodes.at(que.front());
        que.pop();
        cout << "#" << U->id << ", ";

        for (auto arc : U->arcs)
        {
            int vid = arc->V->id;
            if (!visited[vid])
            {
                visited[vid] = true;
                que.push(vid);
            }
        }
    }

    delete [] visited;
}
```

**Runtime:** O( |V|+|E| )

# 为什么叫breadth first

- 算法过程隐含了一个BFS树

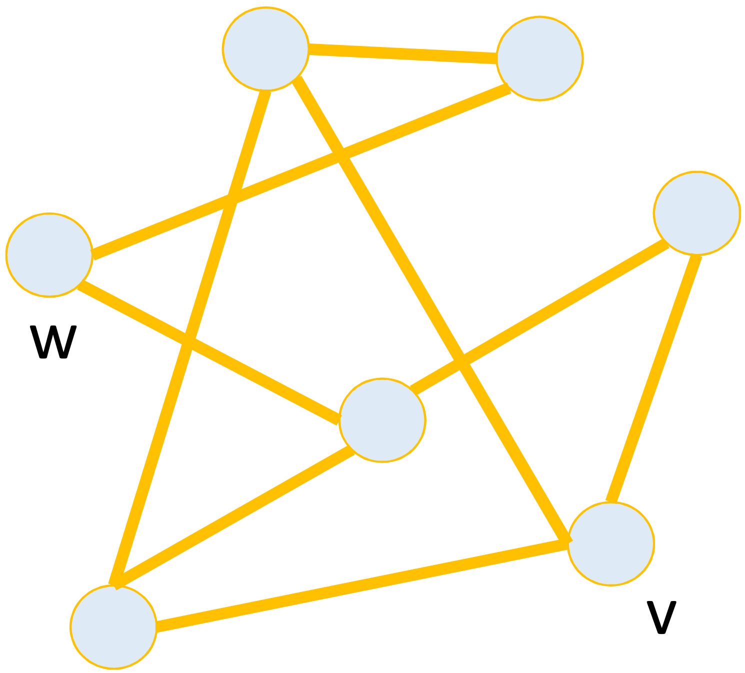


# BFS应用

- BFS算法过程能找到所有能从起始点达到的节点
    - 找出所有的连通分支
    - 最短路径（边的数目）
    - 测试二分图bipartite graph

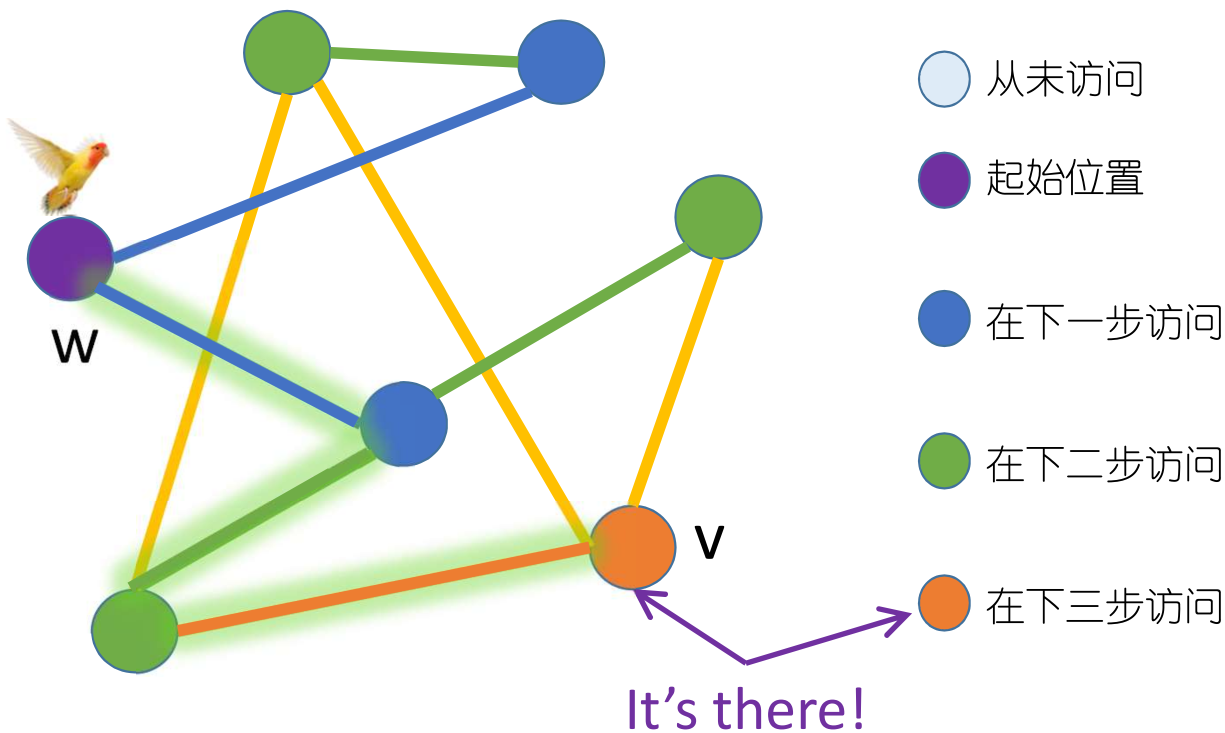
# 两点间最短路径

- w和v之间的最短路径（边的数目）是多少？



# 两点间最短路径

- w和v之间的最短路径是多少？



# 最短路径算法

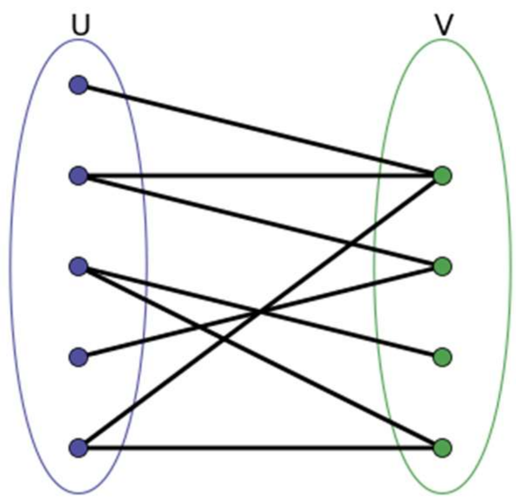
```
int ShortestPath(Node u, Node x)
    queue Q
    Q.enqueue(u)
    while （Q不空）
        size = Q.size();  depth++;
        while (size-- > 0)
            u = Q.deque()
            foreach （v是u的邻接顶点）
                if v.status == ○
                    v.status   = ●
                    Q.enqueue(v)
                if v == x
                    return depth
```

为什么bfs能找到最短路径？
用数学归纳法可以证明离起点最短距离为i的节点都是在BFS树的第i层

# 测试二分图

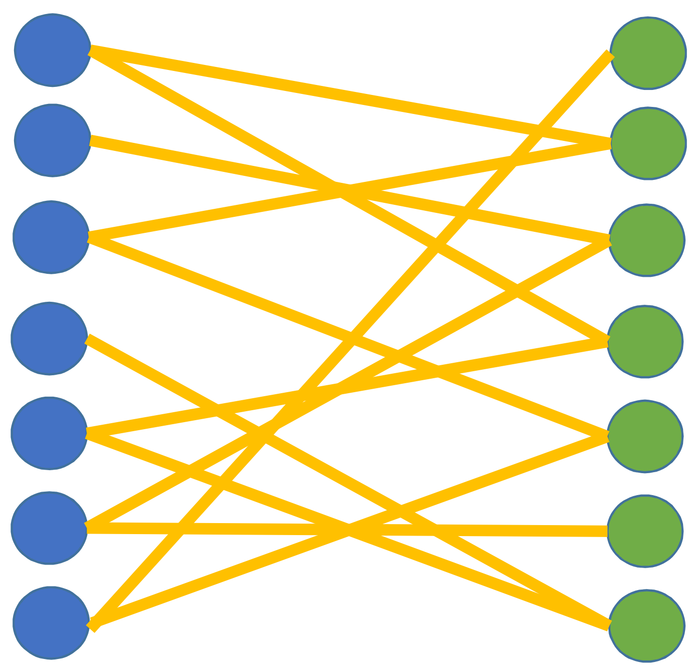
# 二分图

- 图里的顶点可以分为两组，同一组内的顶点之间没有边连接
- 看作把顶点染色，要么蓝色，要么绿色，使得同颜色顶点之间没有边

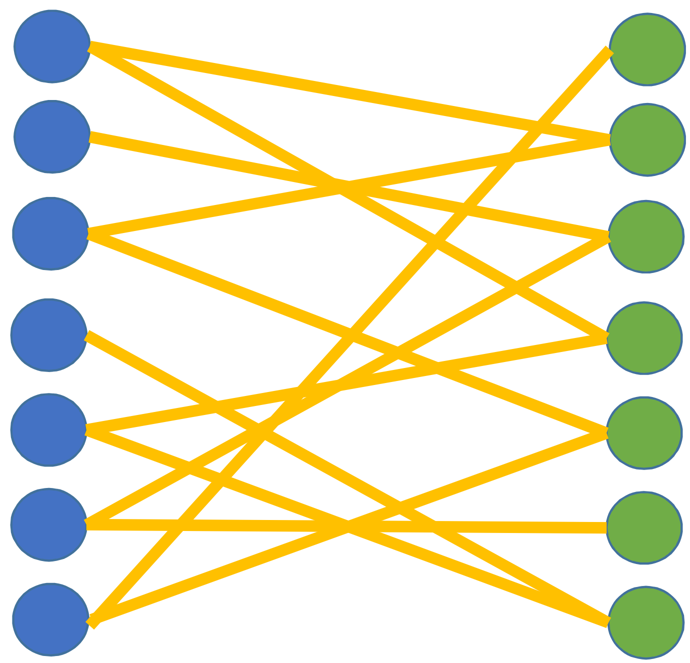


# 测试是不是二分图

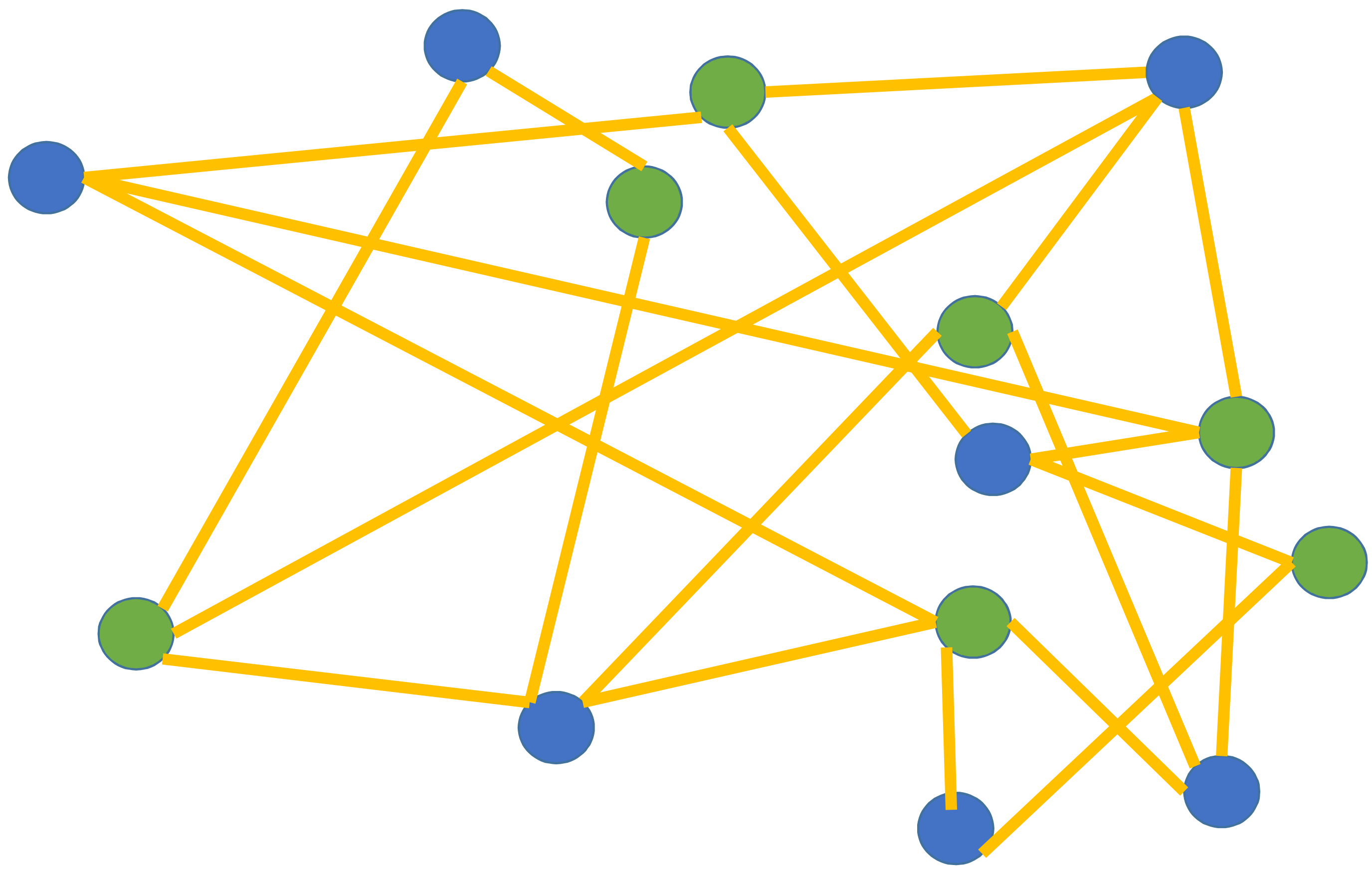
- 看可以给图中顶点染两种颜色之一，使得同颜色顶点之间没有边
- 比如：



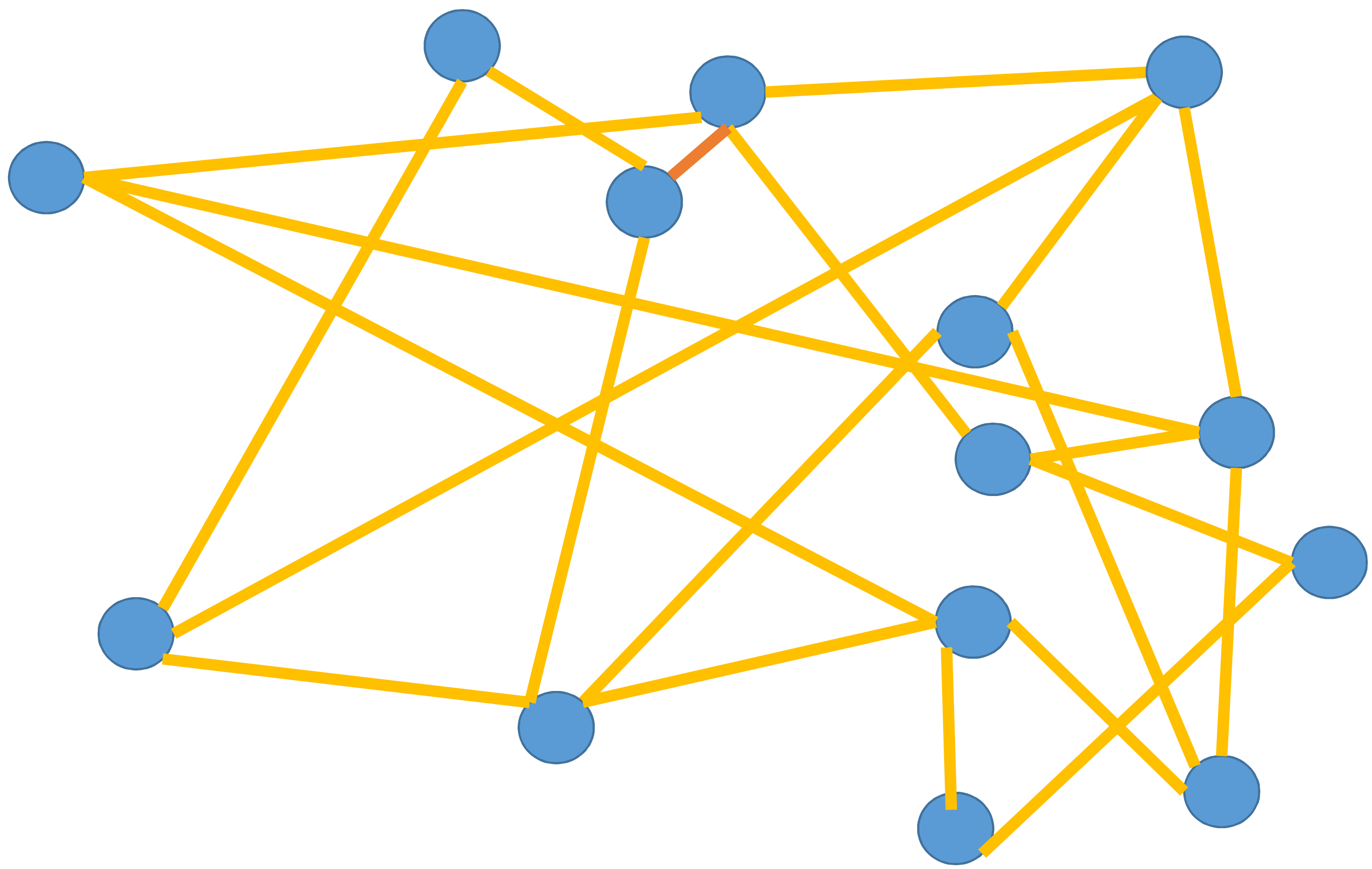
是不是?



是不是?

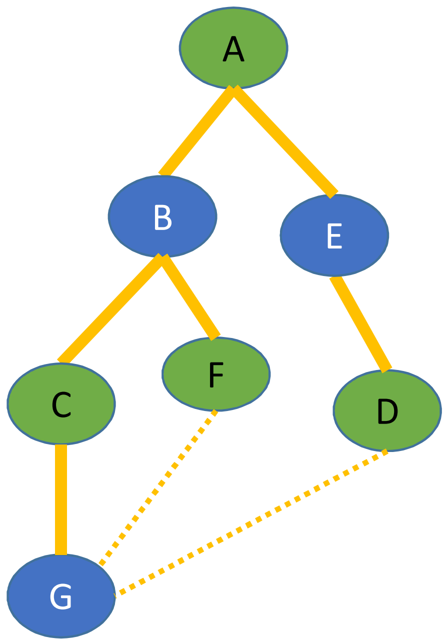


是不是?

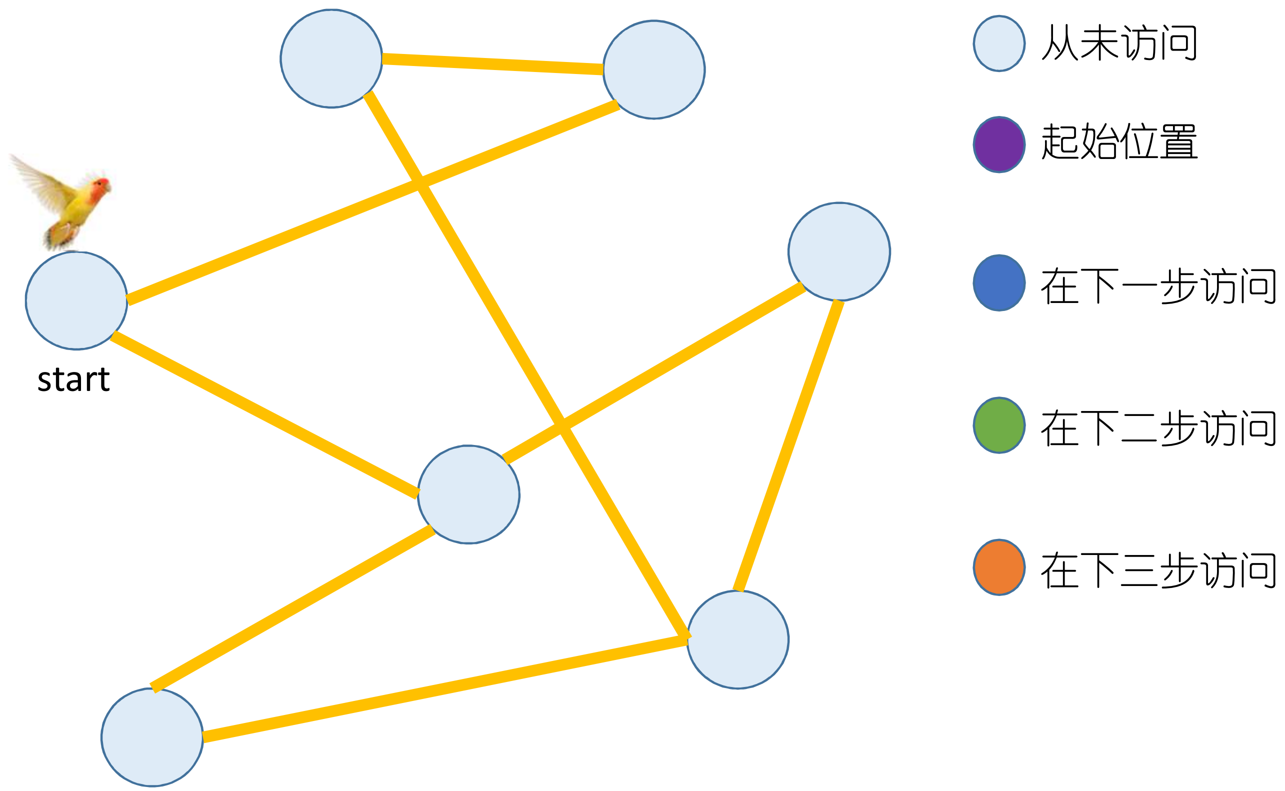


# 测试二分图算法

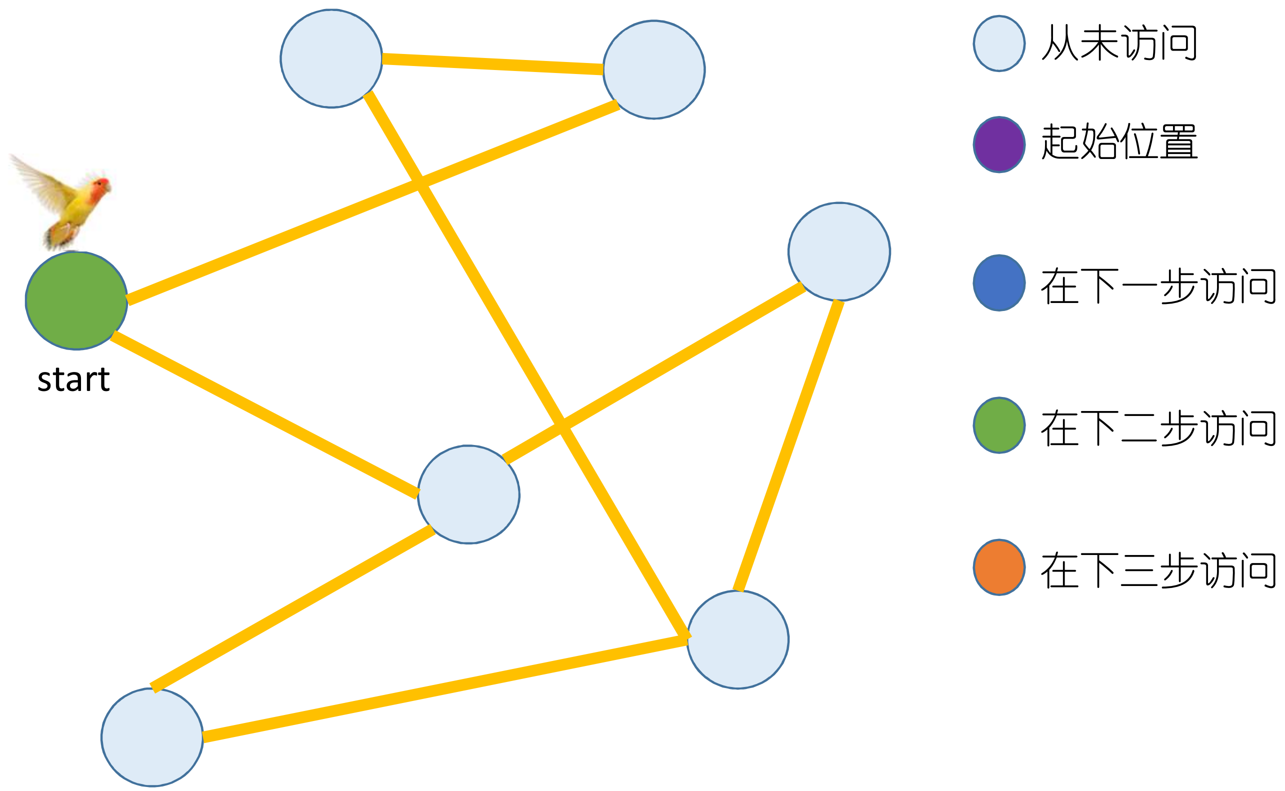
- 用BFS来对不同层的节点轮流染不同色
- 如果当前访问的顶点的邻接顶点已染相同颜色，就不是二分图



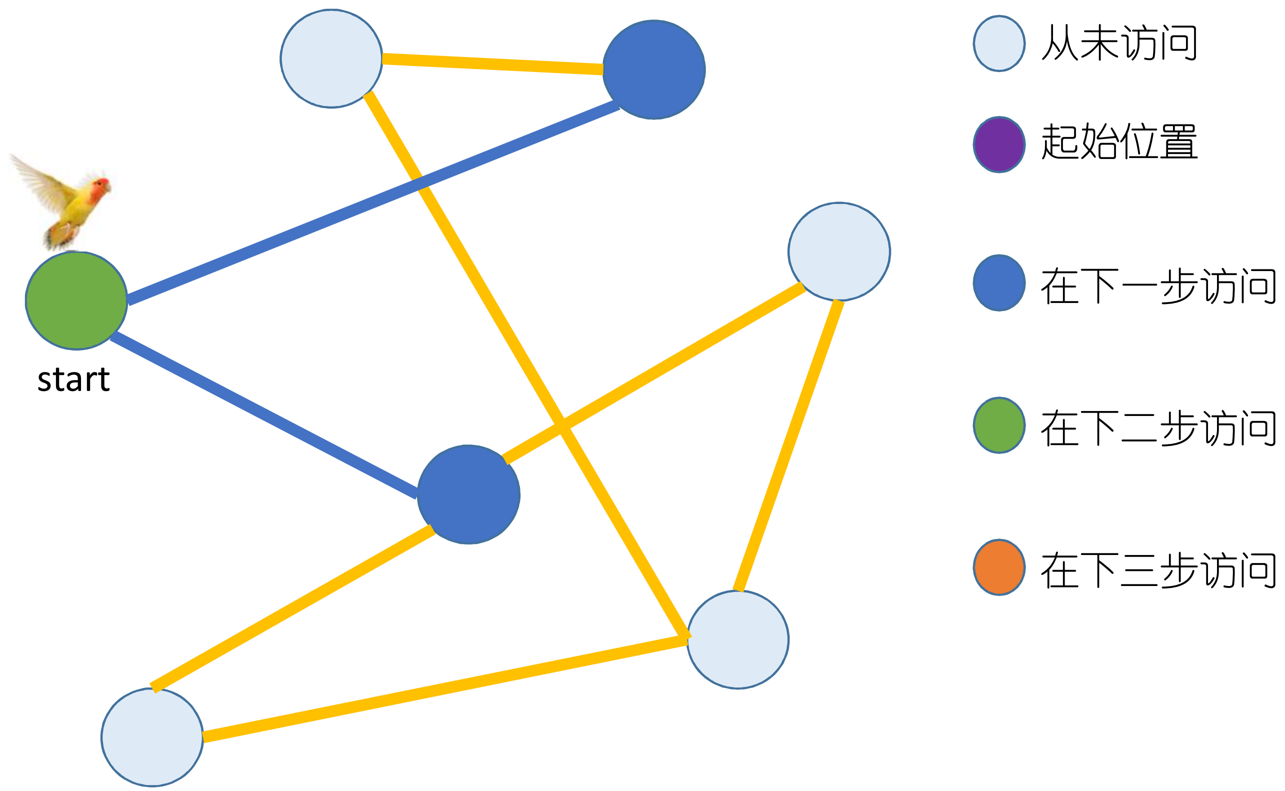
# 测试二分图



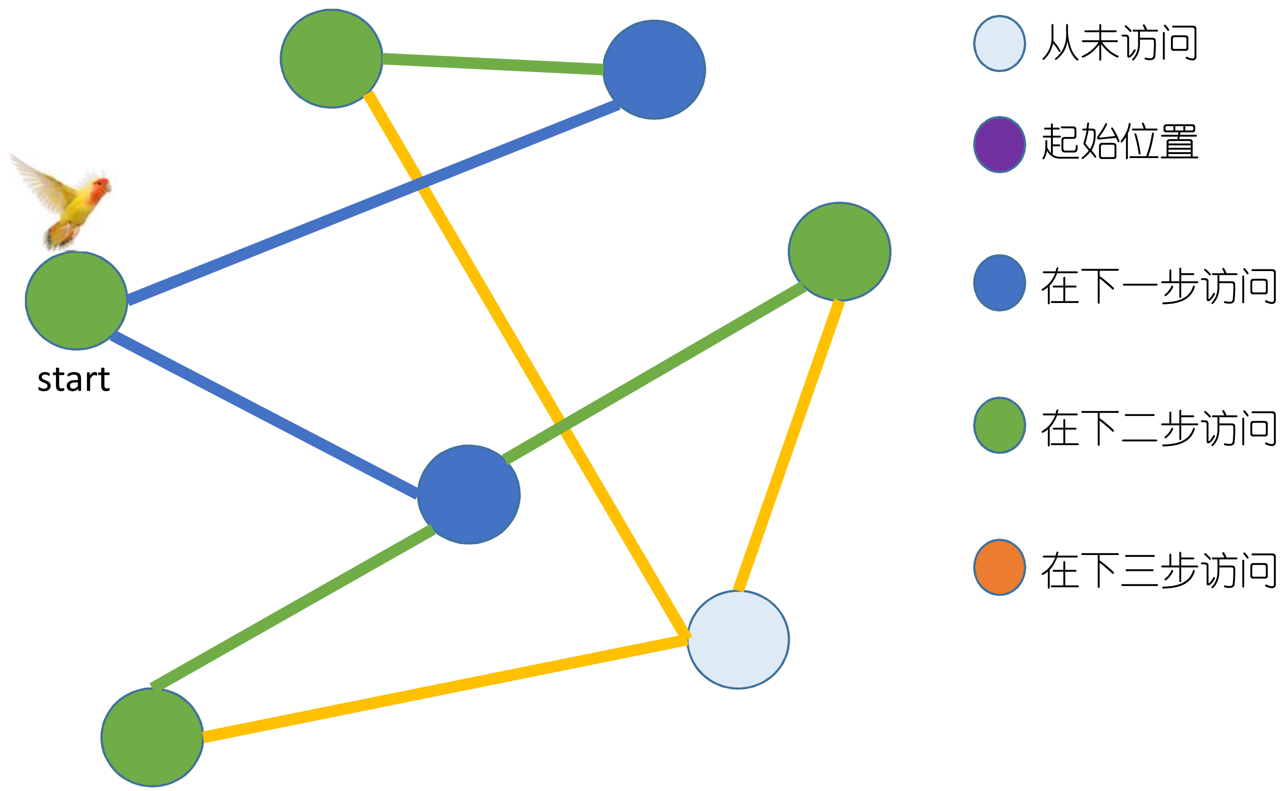
# 测试二分图



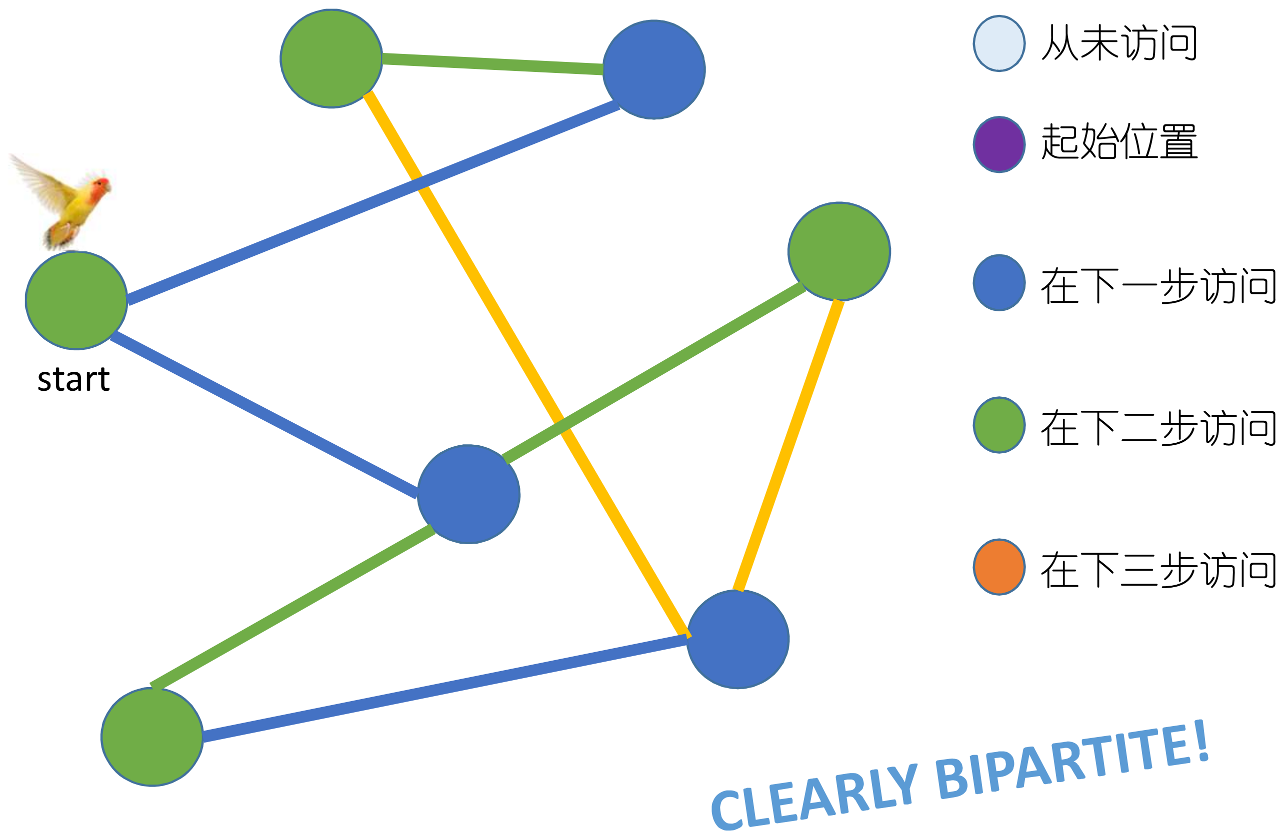
# 测试二分图



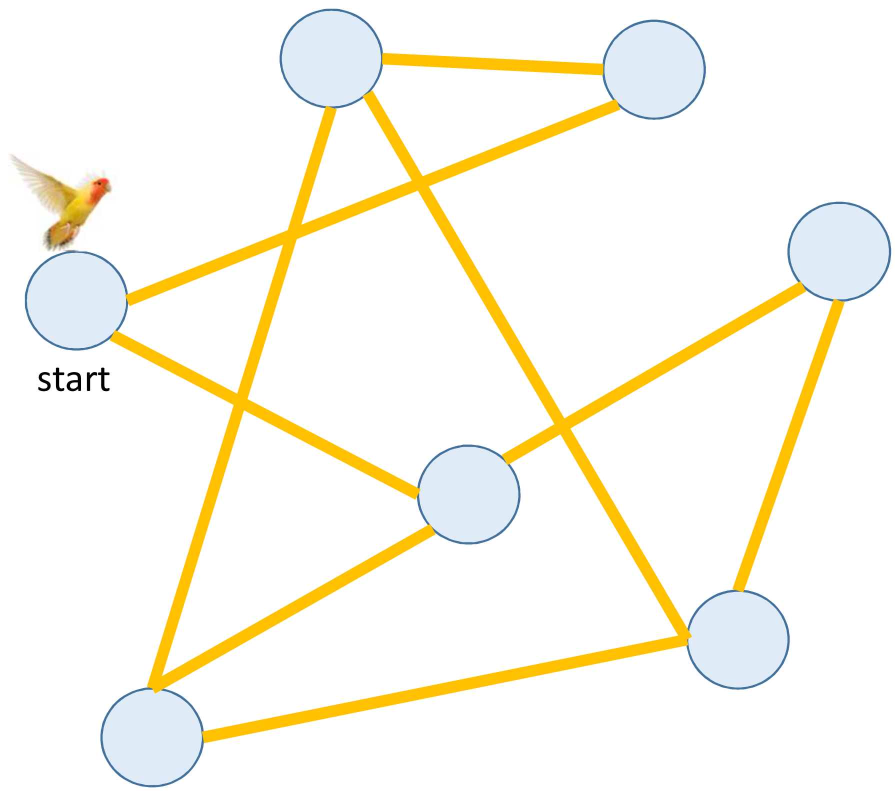
# 测试二分图



# 测试二分图



# 测试二分图（二）




从未访问

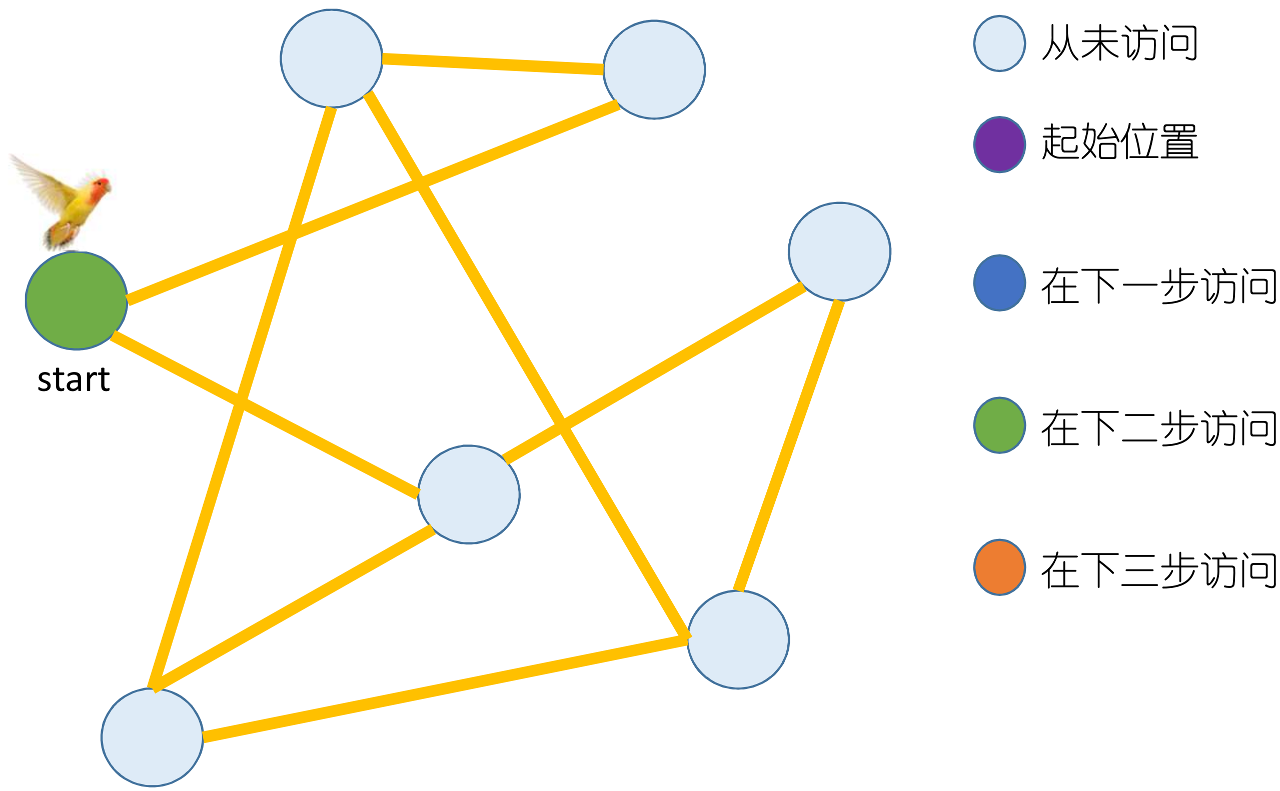
起始位置

在下一步访问

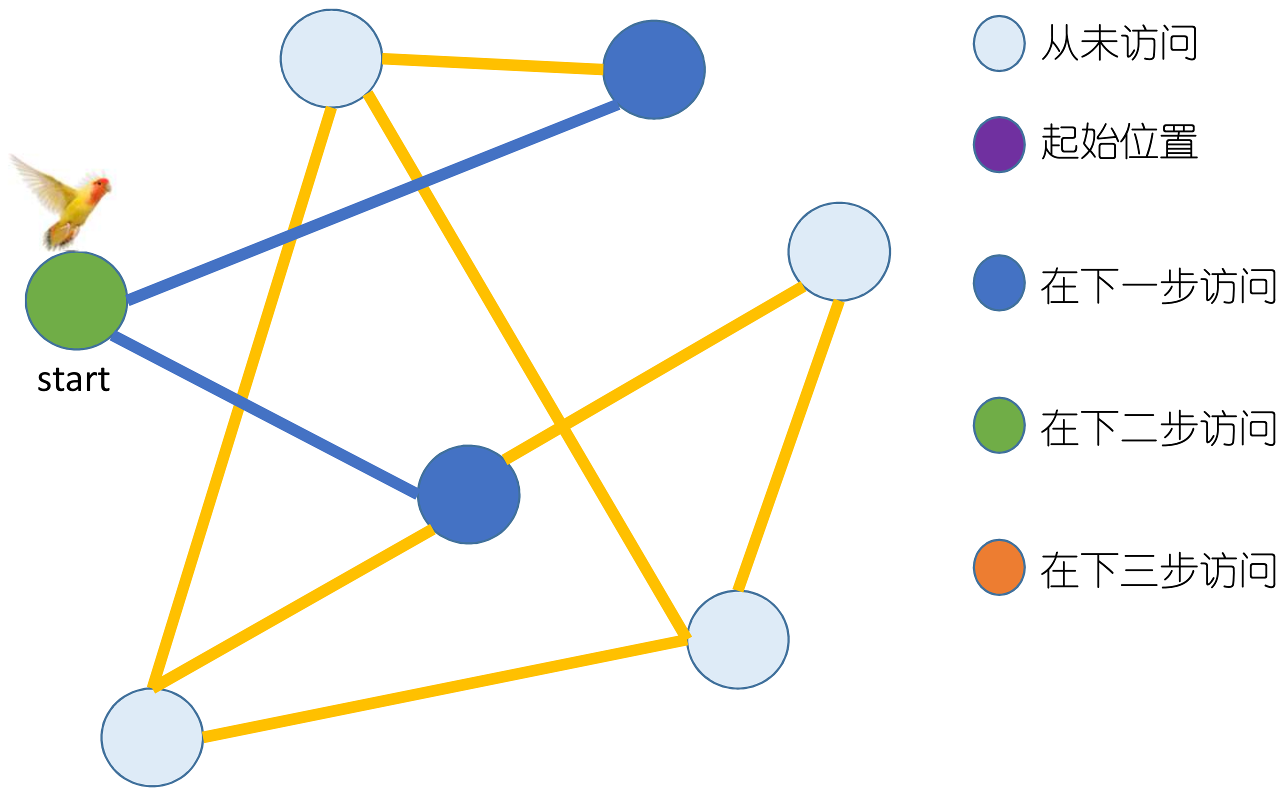
在下二步访问

在下三步访问

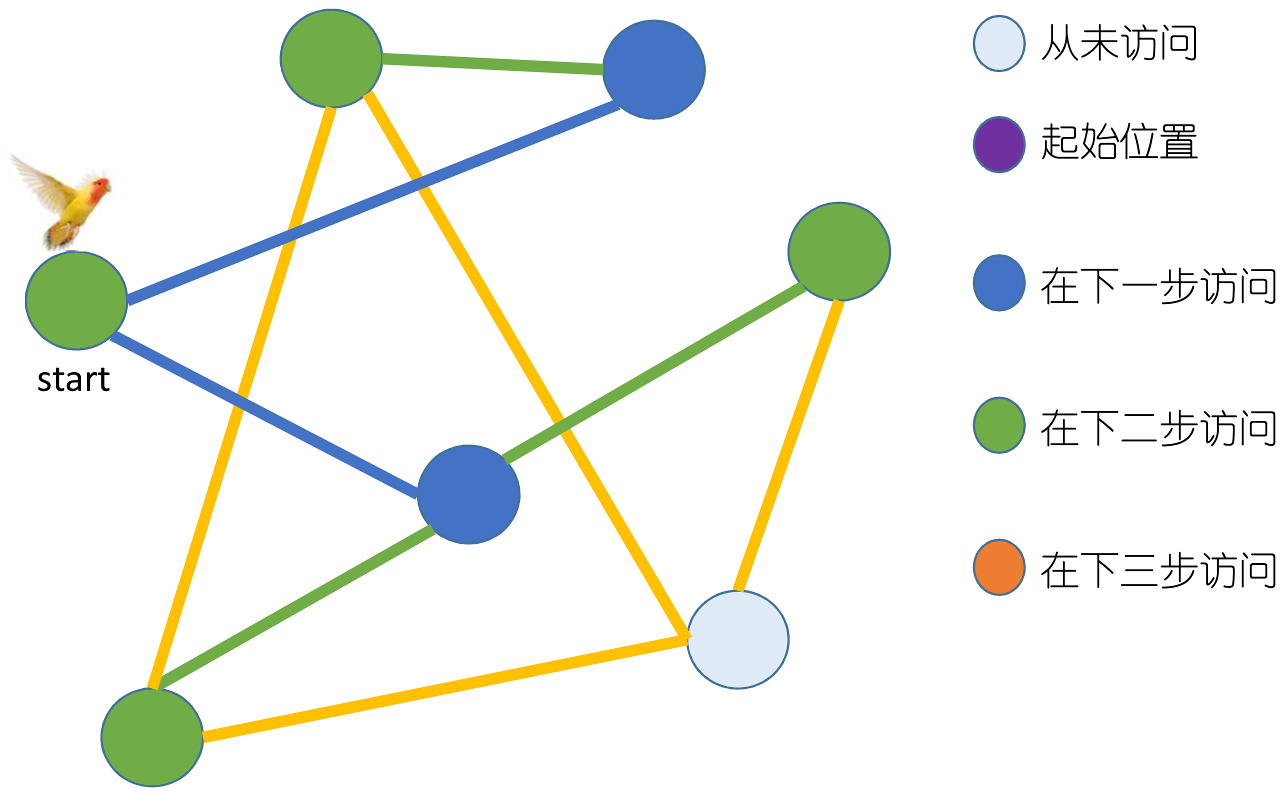
# 测试二分图（二）



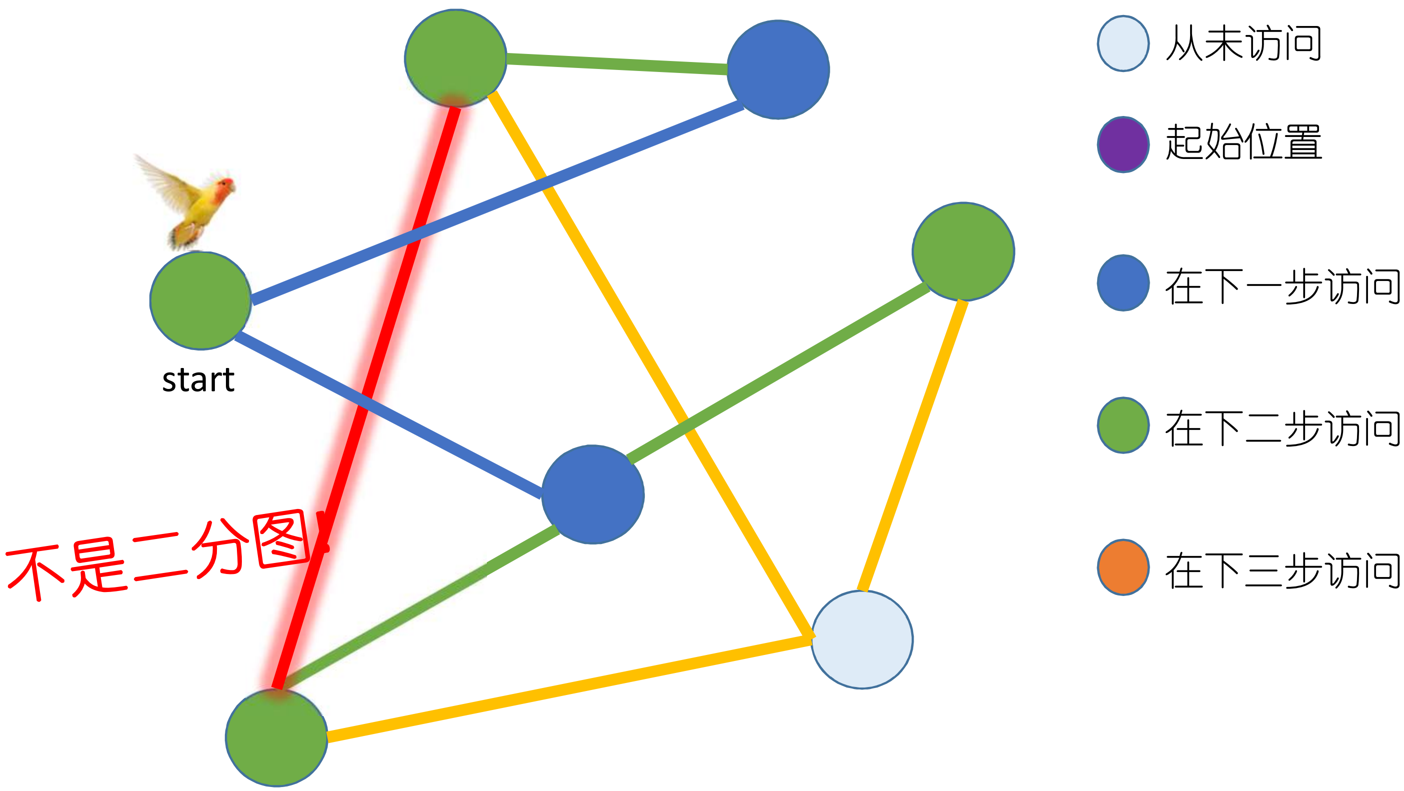
# 测试二分图（二）



# 测试二分图（二）

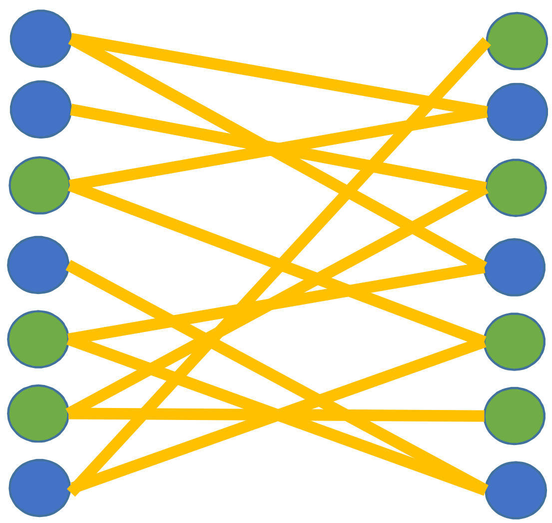


# 测试二分图（二）



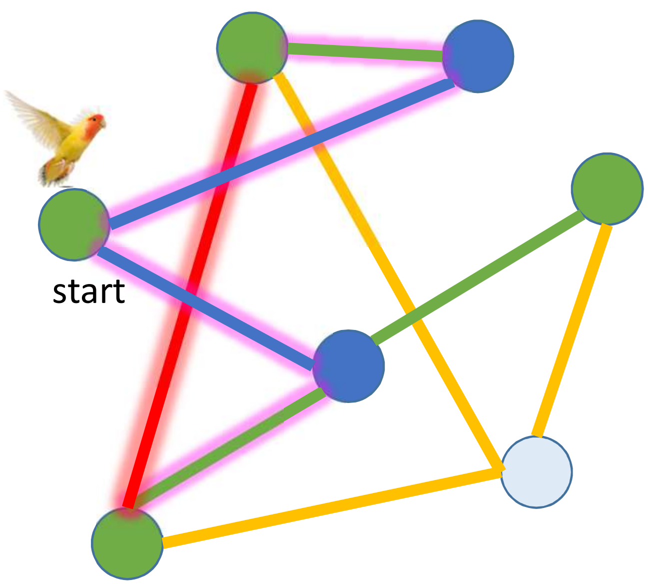
# 严格来说，

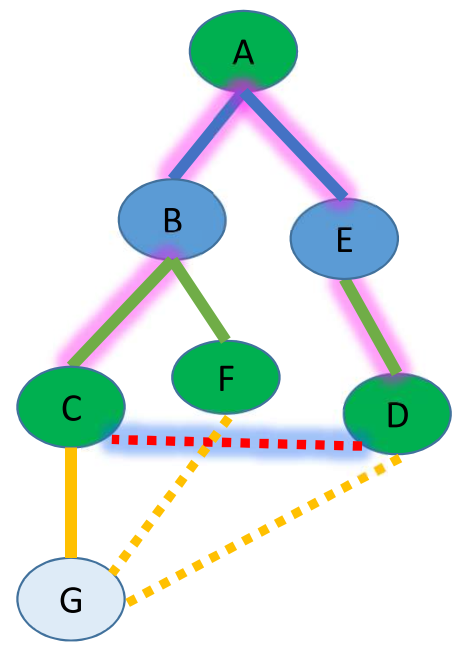
- 上面算法只是说明了一种染色方法不对
- 还不能说明这不是二分图
- 比如：



# 首先，

- **Claim**：如果BFS把两个相邻的顶点染成同一种颜色，那么一定存在长度为奇数的环路

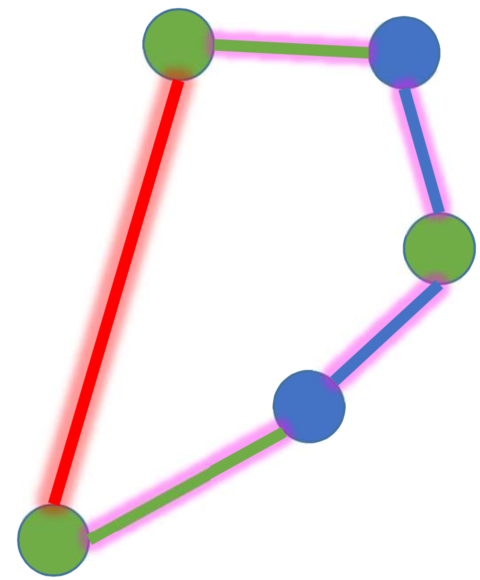


# 证明

- 如果BFS把两个相邻的顶点分成同一种颜色，
- 存在长度为奇数的环路
- 把它作为一个子图

- 那么不可能给这个子图染两种颜色，使得相邻顶点颜色不同



# 小结

- DFS
  - 用来topological ordering
  - BST中序遍历排序
- BFS
  - 找最短路径
  - 测试二分图
- DFS/BFS
  - 都可以用来遍历图，找连通分支，等

# 更多……

- 在有向图中找强连通分支strongly connected component
- 在加权图上找最短路径
- 用动态规划找两两之间的最短路径
- 用贪婪法找最小生成树，最大流

# Q&A

Thanks!