8.4 图遍历的应用

8.4.1 基于深度优先遍历算法的应用

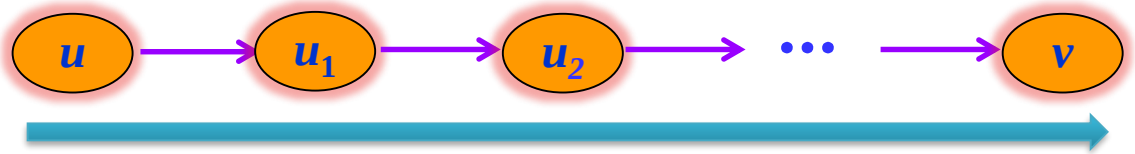
DFS过程：

一步一步向前走，当没有可走的相邻顶点时便回退。

【例8-2】假设图G采用邻接表存储，设计一个算法，判断顶点u⇨v是否有简单路径。

求解思路

❶ 从顶点u开始进行深度优先遍历，当搜索到顶点v时表明从顶点u到v有路径，即：



深度优先遍历过程

❷ 用形参has（调用时其初值置为false）表示顶点u→v是否有路径。

```
void ExistPath(AGraph *G,int u,int v,bool &has)
{  //has表示u到v是否有路径,初值为false
      int w;  ArcNode *p;
      visited[u]=1;                          //置已访问标记
      if (u==v)                              //找到了一条路径
      {       has=true;                      //置has为true并结束算法
              return;
      }
   p=G->adjlist[u].firstarc;                 //p指向顶点u的第一个相邻点
   while (p!=NULL)
   {       w=p->adjvex;                      //w为顶点u的相邻顶点
           if (visited[w]==0)                //若w顶点未访问,递归访问它
               ExistPath(G,w,v,has);
           p=p->nextarc;                     //p指向顶点u的下一个相邻点
   }
}
```

深度优先遍历

【例8-3】假设图G采用邻接表存储，设计一个算法输出图G中从顶点u⇨v的一条简单路径（假设图G中从顶点u⇨v至少有一条简单路径）。

求解思路

- 采用深度优先遍历的方法。
- 增加path和d形参，其中path存放顶点u到v的路径，d表示path中的路径长度，其初值为-1。
- 当从顶点u遍历到顶点v后，输出path并返回。

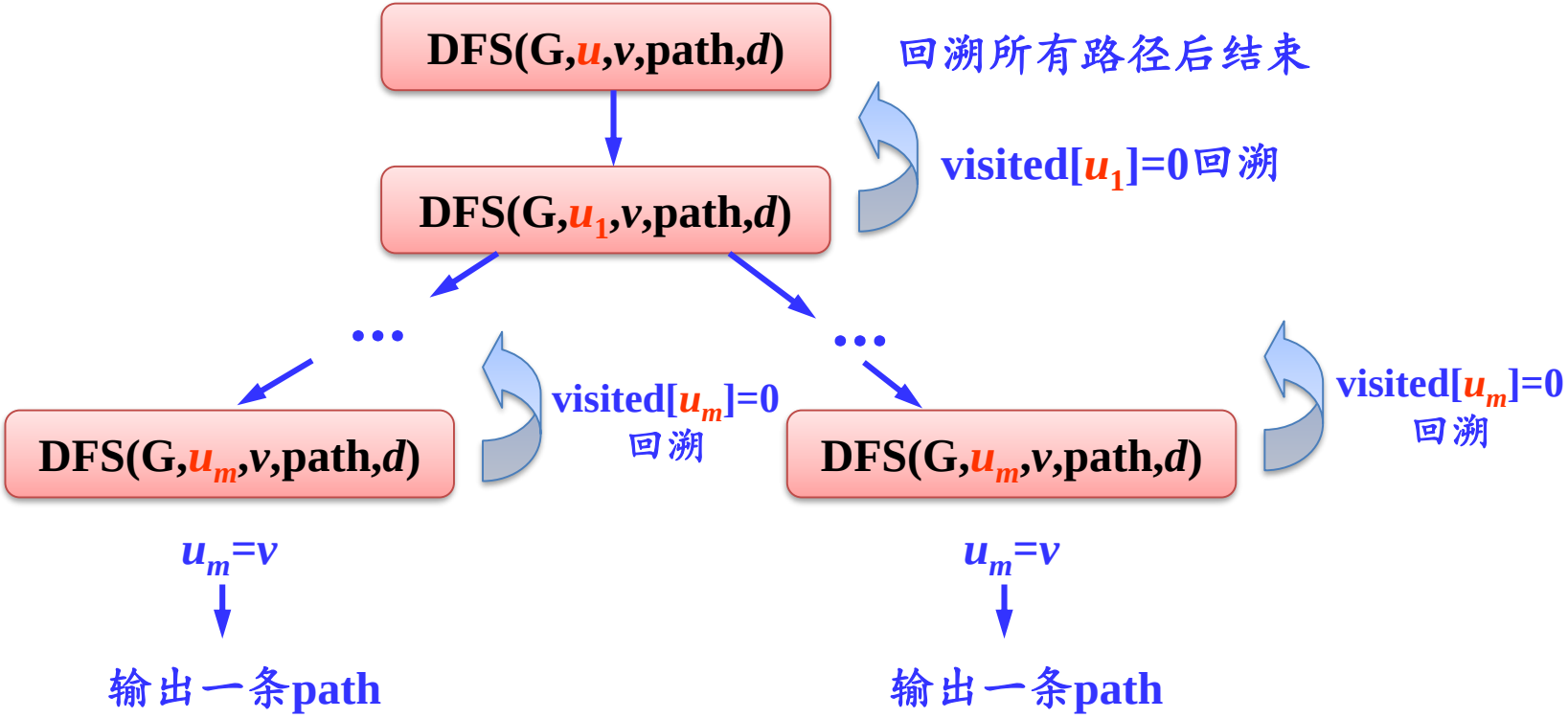
```
void FindaPath(AGraph *G,int u,int v,int path[],int d)
{   //d表示path中的路径长度，初始为-1
    int w,i;  ArcNode *p;
    visited[u]=1;
    d++; path[d]=u;                    //路径长度d增1，顶点u加入到路径中
    if (u==v)                          //找到一条路径后输出并返回
    {       printf("一条简单路径为:");
            for (i=0;i<=d;i++)  printf("%d ",path[i]);
            printf("\n");
            return;                    //找到一条路径后返回
    }
    p=G->adjlist[u].firstarc;          //p指向顶点u的第一个相邻点
    while (p!=NULL)
    {     w=p->adjvex;                 //相邻点的编号为w
          if (visited[w]==0)
              FindaPath(G,w,v,path,d);
          p=p->nextarc;                //p指向顶点u的下一个相邻点
    }
}
```

深度优先遍历

【例8-4】假设图G采用邻接表存储，设计一个算法，输出图G中从顶点u ⇨v的所有简单路径。

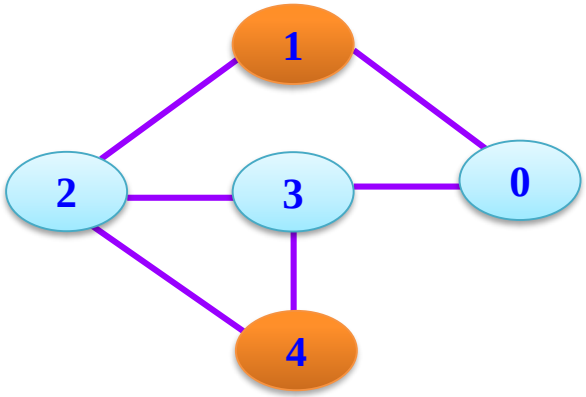
求解思路

- 利用回溯的深度优先遍历方法。
- 从顶点u开始进行深度优先遍历。增加path和d记录存走过的路径。
- 若当前扫描的顶点$u=v$时，表示找到了一条路径，则输出路径path。
- 当从顶点u出发的路径找完后，置visited[u]=0，即回溯。


DFS(G,u,v,path,d)
回溯所有路径后结束
DFS(G,u1,v,path,d)
visited[u1]=0回溯
visited[um]=0
回溯
DFS(G,um,v,path,d)
um=v
输出一条path
visited[um]=0
回溯
DFS(G,um,v,path,d)
um=v
输出一条path

```
void FindPath(AGraph *G,int u,int v,int path[],int d)
{   //d表示path中的路径长度，初始为-1
    int w,i;  ArcNode *p;
    d++; path[d]=u;                    //路径长度d增1，顶点u加入到路径中
    visited[u]=1;                      //置已访问标记
    if (u==v && d>=1)                  //找到一条路径则输出
    {     for (i=0;i<=d;i++)
              printf("%2d",path[i]);
          printf("\n");
    }
    p=G->adjlist[u].firstarc;          //p指向顶点u的第一个相邻点
    while (p!=NULL)
    {     w=p->adjvex;                 //w为顶点u的相邻顶点
          if (visited[w]==0)           //若w顶点未访问,递归访问它
             FindPath(G,w,v,path,d);
          p=p->nextarc;                //p指向顶点u的下一个相邻点
    }
    visited[u]=0;                      //恢复环境，使该顶点可重新使用
}                                      //深度优先遍历
```

算法执行结果



从1到4的所有路径:

1 2 4

1 2 3 4

1 0 3 4

1 0 3 2 4

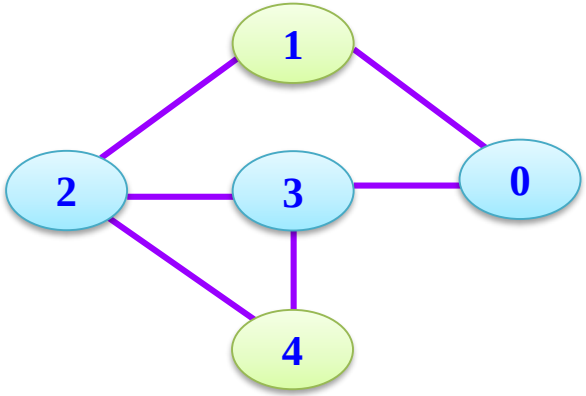
【例8-5】假设图G采用邻接表存储，设计一个算法，输出图G中从顶点u到v的长度为l的所有简单路径。

遍历思路和上例相似，只需将路径输出条件改为：$u==v$且$d==l$。

```
void PathAll(ALGraph*G,int u,int v,int l,int path[],int d)
//d是到当前为止已走过的路径长度，调用时初值为-1
{     int w，i; ArcNode *p;
      visited[u]=1; d++;                    //路径长度增1
      path[d]=u;                            //将当前顶点添加到路径中
      if (u==v && d==l)                     //输出一条路径
      {     for (i=0;i<=d;i++) printf("%d "，path[i]);
            printf("\n");
      }
   p=G->adjlist[u].firstarc;                //p指向u的第一条边的边头节点
   while (p!=NULL)
   {    w=p->adjvex;                        //w为u的邻接顶点
        if (visited[w]==0)                  //若顶点未标记访问，则递归访问之
              PathAll(G,w,v,l,path,d);
       p=p->nextarc                         //找u的下一个邻接顶点
   }
    visited[u]=0;  ←恢复环境，使该顶点可重新使用
}
```

深度优先遍历

算法执行结果



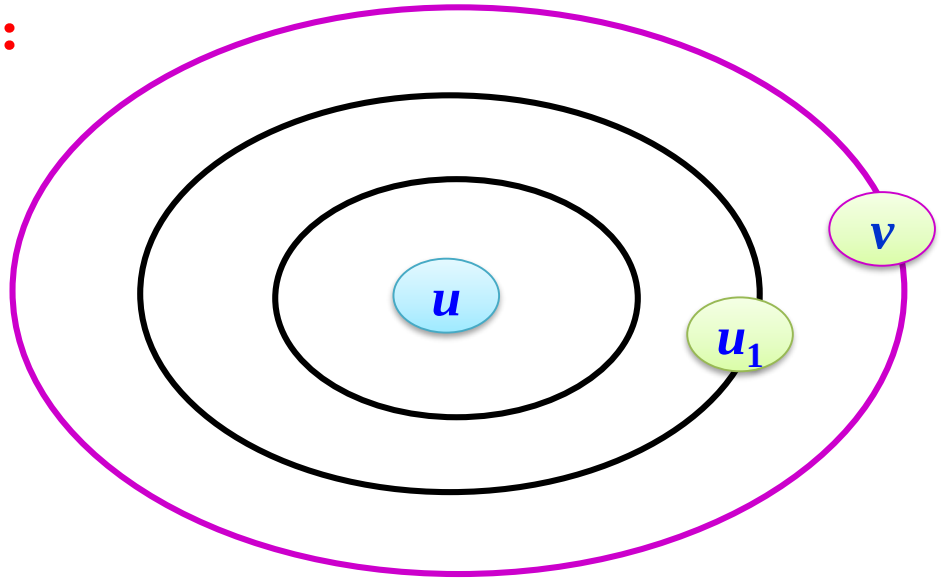
从1到4的所有长度为3路径:

1 0 3 4

1 2 3 4

8.4.2 基于广度优先遍历算法的应用

BFS过程：



一圈一圈向外走。

以 $u \rightarrow v$ 的最短路径构成分层

【例8-6】假设图G采用邻接表存储，设计一个算法，求不带权无向连通图G中从顶点u⇨v的一条最短路径（路径上经过的顶点数最少）。

最好采用广度优先遍历来实现。

```
typedef struct
{    int data;        //顶点编号
     int parent;      //前一个顶点的位置
} QUERE;
```

非循环队列类型

```
void ShortPath(ALGraph *G,int u,int v)
{  //输出从顶点u到顶点v的最短逆路径
     ArcNode *p;  int w,i;
     QUERE qu[MAXV];                  //定义非循环队列
     int front=-1,  rear=-1;          //队列的头、尾指针
     int visited[MAXV];
     for (i=0;i<G->n;i++)             //访问标记置初值0
          visited[i]=0;
     rear++;                          //顶点u进队
     qu[rear].data=u;
     qu[rear].parent=-1;
     visited[u]=1;
```

```
while (front!=rear)                          //队不空循环
{     front++;                               //出队顶点w
      w=qu[front].data;
```

```
if (w==v)
{    i=front;
      while (qu[i].parent!=-1)
      {    printf("%2d ",qu[i].data);
           i=qu[i].parent;
       }
       printf("%2d\n",qu[i].data);
       break;
  }
```

输出逆路径

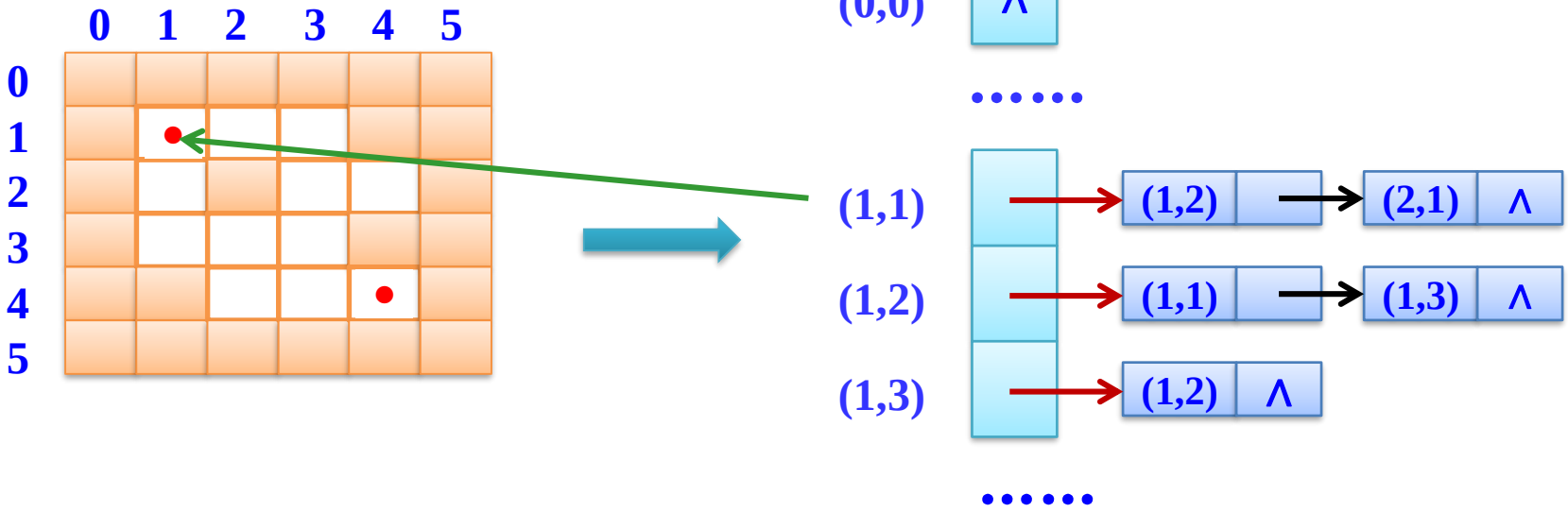
```
        p=G->adjlist[w].firstarc;                    //找w的第一个邻接点
        while (p!=NULL)
        {     if (visited[p->adjvex]==0)
              {     visited[p->adjvex]=1;
                    rear++;                          //将w的未访问过的邻接点进队
                    qu[rear].data=p->adjvex;
                    qu[rear].parent=front;
              }
              p=p->nextarc;                          //找w的下一个邻接点
        }
    }
}
```

思考题

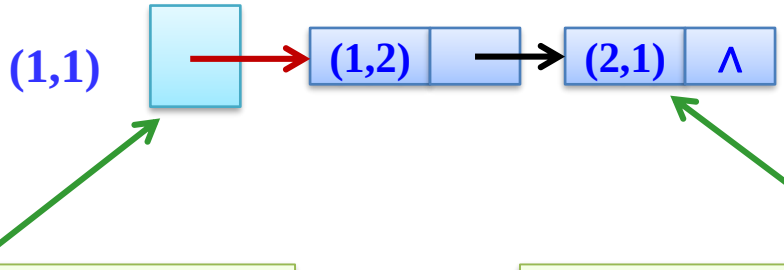
如果一个图是带权图，能够采用上例广度优先遍历方法求顶点u到v的最短路径吗？

用DFS和BFS求解迷宫问题

创建迷宫问题的邻接表：



邻接表设计：




```
typedef struct Vnode
{
        ArcNode *firstarc;
}  VNode;
```

```
typedef struct ANode
{      int i, j;
       struct ANode *nextarc;
}  ArcNode;
```

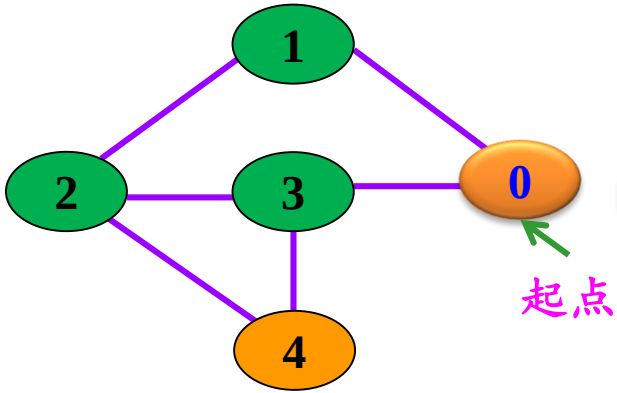
```
typedef struct
{
        VNode adjlist[M+2][N+2];
}  ALGraph;      //迷宫图的邻接表类型
```

算法设计

- 采用DFS或者BFS算法
- 入口作为初始顶点
- 结束条件为找到出口
- visited改为二维数组

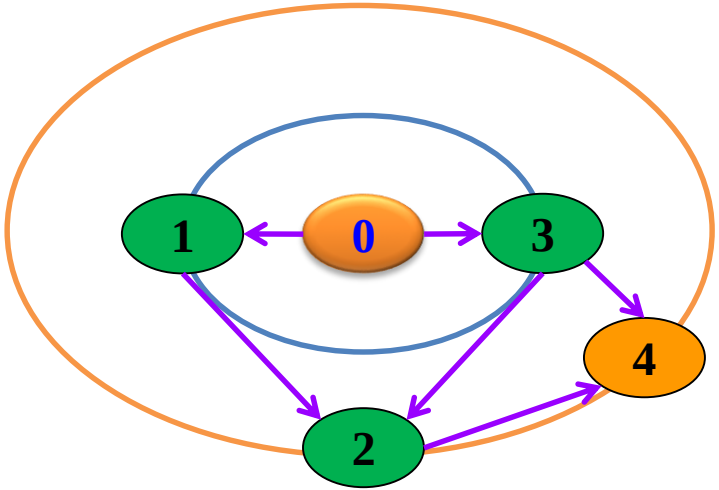
具体算法请你实现！

DFS和BFS求解迷宫问题的差别



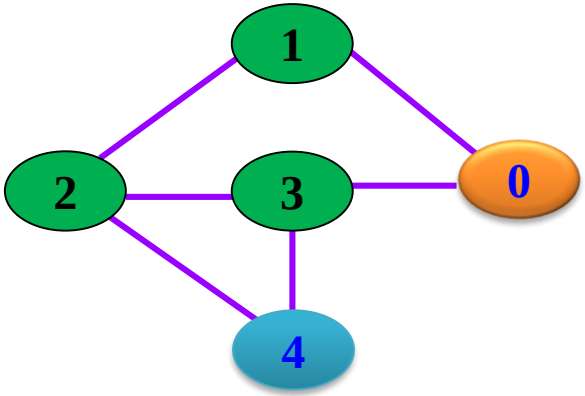
每个顶点表示一个方块

以0→v的
最短路径
构成分层

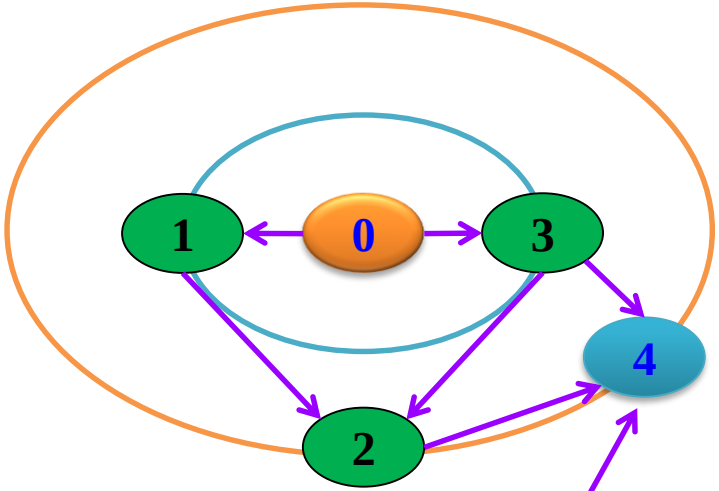


表示相邻的边

深度优先遍历可能找到0⇨4的一条路径：



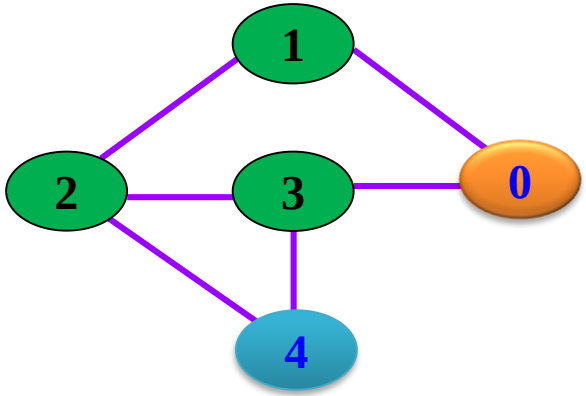
路径：0→ 1 → 2 → 4

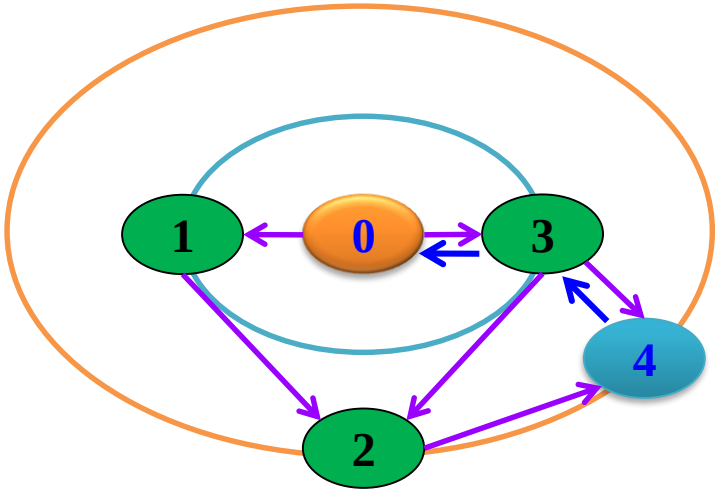


路径上的顶点可能在同一层

不一定是最短路径

广度优先遍历找到0⇨4的路径同一层只能有一个顶点。





逆路径：4 → 3 → 0

路径：每一层只有一个顶点

一定是最短路径

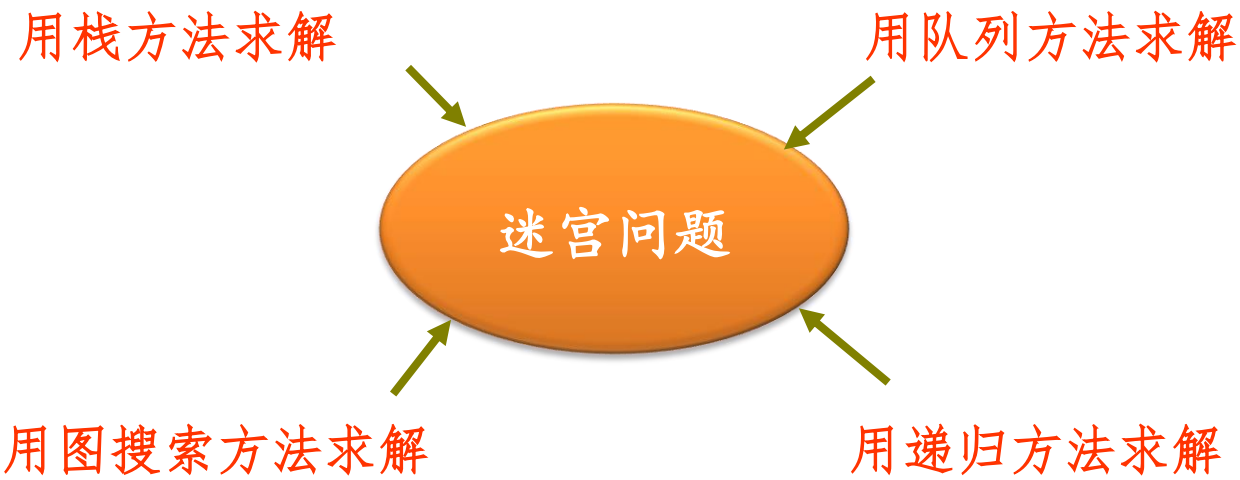
结论：

以路径上经过的边数来衡量路径长度

- 广度优先遍历找到的路径一定是最短路径，而深度优先遍历则不一定。
- 深度优先遍历能找所有路径，而广度优先遍历难以实现。

数据结构算法的多维性

同一问题的多种解法。



各种求解方法的特点和差别

——本讲完——