数据结构

2017秋季　刘鹏远

图
GRAPH

- 图的基本概念
- 图的存储表示
- 图的遍历与连通性
- 图的遍历应用
- 最小生成树
- 最短路径
- 活动网络(选修)

邻接矩阵\邻接表相关操作实现
图的深度优先、广度优先遍历
图的遍历算法应用
图的连通分量

邻接矩阵 几个操作实现

用邻接矩阵表示的图结构的定义


```
#define MaxValue INT_MAX
#define NumEdges  500
#define NumVertices  50
typedef char VertData;
typedef int EdgeData;
typedef struct {
    VertData VexList[NumVertices];
    EdgeData edge[NumVertices][NumVertices];
    int n, e;
    //int kind;    //图的种类 如有向无向有无权值等
} MGraph;
```

操作


```
int IsEmpty(MGraph* G)
{ return G->n == 0; }//判图G空否,空则返回1,否则返回0。

EdgeData GetWeight (MGraph* G, int u, int v) {
//给出以顶点 u 和 v 为两端点的边上的权值
    if (u != -1 && v != -1) return G->edge[u][v];
    else return 0;//严格的u,v边界自行设置
}

VertData GetValue (MGraph* G, int i ){
//给出第 i 个顶点的数据值（顶点表中）
     return i >= 0 && i < G->n ? G->VexList[i] : '\0';
}    //超限返回'\0'或ERROR或其他约定
```

Print(G)

操作

```
int GetFirstNeighbor (MGraph* G, int v) {
//给出顶点 v（位置）的第一个邻接顶点的位置
  if ( v != -1 ) {//边界条件自行处理
     for ( int j = 0; j < G->n;  j++ )
        if ( G->Edge[v][j] > 0 &&
             G->Edge[v][j] < MaxValue ) return j;
      //第 v 行从0寻找第一个邻接顶点
  }
  return -1;} O（n）//找不到返回-1是约定
```

操作


```
int GetNextNeighbor (MGraph* G, int v, int w) {
//给出顶点v的下一个邻接顶点（相对于w顶点）
    if ( v != -1 && w != -1 ) {
      for ( j = w+1; j < G->n; j++ ) {
          if ( G->edge[v][j] > 0 &&
               G->edge[v][j] < MaxValue ) return j;
          //在第 v 行顺序寻找w下一个邻接顶点
    }  return -1;}
```

邻接表 几个操作实现

邻接表 (Adjacency List)表示


```
#define NumVertices 50;
typedef char VertData;
typedef float EdgeData;

typedef struct node {
    int dest;                    //目标顶点下标
    EdgeData cost;               //边上的权值
    Struct node * next;          //下一边链接指针
} EdgeNode;                      //边结点
```

邻接表 (Adjacency List)表示


```
typedef struct {                        //顶点结点
    VertData data;                      //数据域
    EdgeNode * first_adj;               //头指针
} VertNode;

typedef struct {                        //图的邻接表定义
    VertNode VexList [NumVertices];     //邻接表
    int n, e;                           //图中当前的顶点个数与边数
} AdjGraph;
```

操作

```
EdgeData GetWeight (AdjGraph* G, int u, int v) {
//给出以顶点 u 和 v 为两端点的边上的权值
    if (u != -1 && v != -1) {//其他边界条件自行处理
      EdgeNode * p = G->VexList[u].FirstAdj;//u节点的指针
        while ( p) {
          if ( p->dest == v ) return p->cost;
            else p = p->next;
        }
    }
    return 0;//其他情况，或MAX_INT，或其他
}O不易确定，可暂视为：O(e/n)
```

操作


```
VertData GetValue (AdjGraph* G, int i )//取某顶点值
    return i >= 0 && i < G->n ? G->VexList[i].data : '\0';
```

Free(G)

Print(G)

操作

```
int GetFirstNeighbor (AdjGraph* G, int v) {
//查找顶点 v 第一个邻接顶点在邻接表中的位置(编号)
    if ( v != -1 ) {                          //若顶点存在
        EdgeNode * p = G->VexList[v].FirstAdj;
        if ( p ) return p->dest;       }
    return -1;      }       //若不存在邻接点
```

操作

```
int GetNextNeighbor (AdjGraph* G, int v, int w ) {
//给出顶点v的下一个邻接顶点（相对于w顶点）
    if ( v != -1 ) {
        EdgeNode * p = G->VexList[v].FirstAdj;
        while ( p  ) {
            if ( p->dest == w && p->next )
                return p->next->dest;
                    //返回下一个邻接顶点在邻接表中位置
           else p = p->next;//继续找w
        }    }
    return -1;       //没有查到下一个邻接顶点}
```

图的遍历与连通性

■线性结构的遍历

1:1

■层次结构的遍历

1:2或m，分支

■图结构的遍历

m:n，并且.....

从已给的连通图中某一顶点出发，沿着一 些边访遍图中所有的顶点，且使每个顶点仅被访问一次，就叫做图的遍历 (Graph Traversal)。

图的遍历与连通性

- 有的教材称为："周游"。
- 图中可能存在回路，且图的任一顶点都可能与其它顶点相通，在访问完某个顶点之后可能会沿着某些边又回到了曾经访问过的顶点。（怎么办？）
- 1：设置一个标志顶点是否被访问过的辅助数组 visited [ ]，设为全局变量
- 或2：visited[]作为遍历时候的参数
- 或3：更改顶点结构体，增加一个visited域。

图的遍历与连通性

- 辅助数组 visited [ ] 的初始状态为 0或False, 在图的遍历过程中, 一旦某一个顶点 *i* 被访问, 就让 visited [*i*] 为 1或True，防止它被多次访问。
- 图的遍历的分类:
  - 深度优先搜索

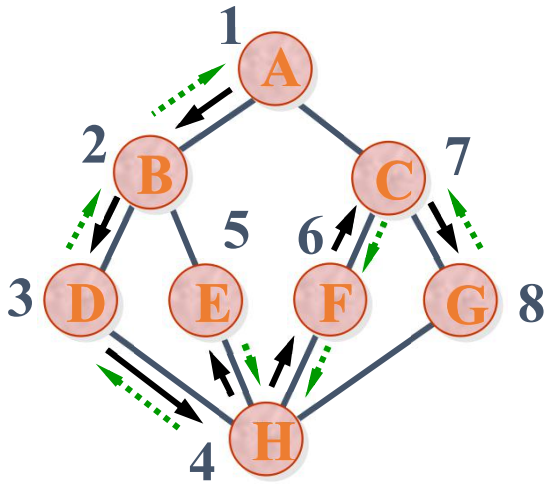
    DFS (Depth First Search)
  - 广度优先搜索

    BFS (Breadth First Search)

深度优先搜索DFS (Depth First Search)

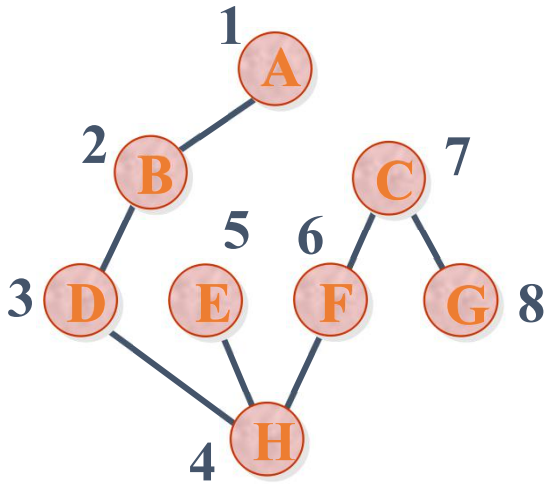
DFS
作业
图例

■深度优先搜索的示例



前进 ——→

深度优先搜索过程



回退 ·····→

深度优先生成树

深度优先搜索的基本思路

- 由 v 出发, 访问它的任一邻接顶点 w_1; 再从 w_1 出发,访问与 w_1邻 接但还没有访问过的顶点 w_2; 然后再从 w_2 出发, 进行类似的访问, … 如此进行下去，直至所有的邻接顶点都被访问过的顶点 u 为止。
- 退回一步，退到前一次刚访问过的顶点，看是否还有其它没有被访问的邻接顶点。如果有，则访问此顶点, 之后再从此顶点出发，进行与前述类似的访问; 如果没有, 就再退回一步进行搜索。
- 重复每个节点。

图的深度优先搜索算法

邻接矩阵邻接表通用递归框架：

有DFS（G,v）方法可以从v开始遍历G。

DFS（G,v）：

1、对当前顶点v，访问，标记其已被访问

2、对当前顶点的每一个邻接点w：

如果没有被访问过，则DFS（G,w）之

递归出口：

1、v已被访问/v不合法

2、v的所有w都被访问

图的深度优先搜索算法


```
void DFS (AdjGraph* G, int v) {
    if visited[v]=1 return;          //出口
    visit(v) ； visited[v] = 1;     //访问(可printf)并做标记
    int w = GetFirstNeighbor (G, v); //取 v 第一个邻接顶点 w
    while ( w != -1 ) {              //若邻接顶点 w 存在
        if ( !visited[w] ) DFS (G, w);
          //若顶点 w 未访问过, 递归访问顶点 w
        w = GetNextNeighbor (G, v, w );
          //取顶点 v 排在 w 后的下一个邻接顶点
    }}//O ( n² ) ,O ( n+e ) 通用递归框架
```

动画

图的深度优先搜索算法


```
void GraphTraverse (AdjGraph* G) {
    for ( int i = 0; i < G->n; i++ )
        visited [i] = 0;          //全局数组 visited 初始化
    for ( int i = 0; i < G->n; i++ )
        if ( ! visited[i] ) DFS (G, i);
                                  //从顶点 i 出发开始搜索
}//为什么要对所有顶点都进行DFS呢？
```

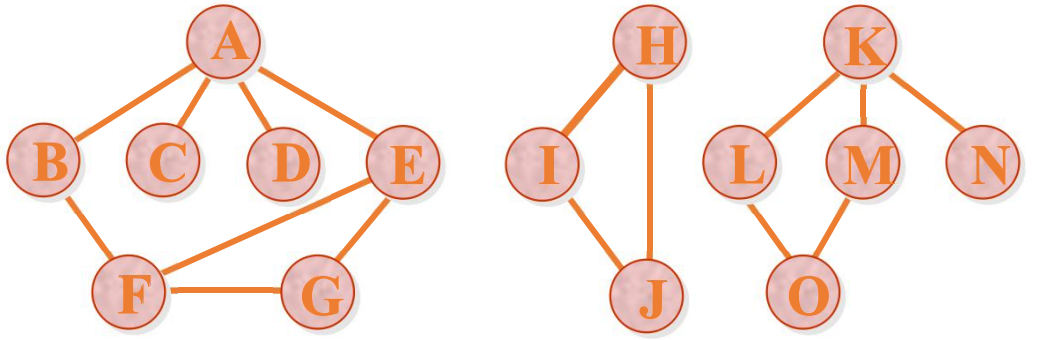
连通分量 (Connected component)

- 当无向图为非连通图时，从图中某一顶点出发，利用深度优先搜索算法或广度优先搜索算法不可能遍历到图中的所有顶点，只能访问到该顶点所在的最大连通子图（连通分量）的所有顶点。
- 若从无向图的**每一个**连通分量中的一个顶点出发进行遍历，可求得无向图的**所有连通分量**。
- 求连通分量的算法需要对图的每一个顶点进行检测：若已被访问过，则该顶点一定是落在图中已求得的连通分量上；若还未被访问，则从该顶点出发遍历图，可求得图的另一个连通分量。

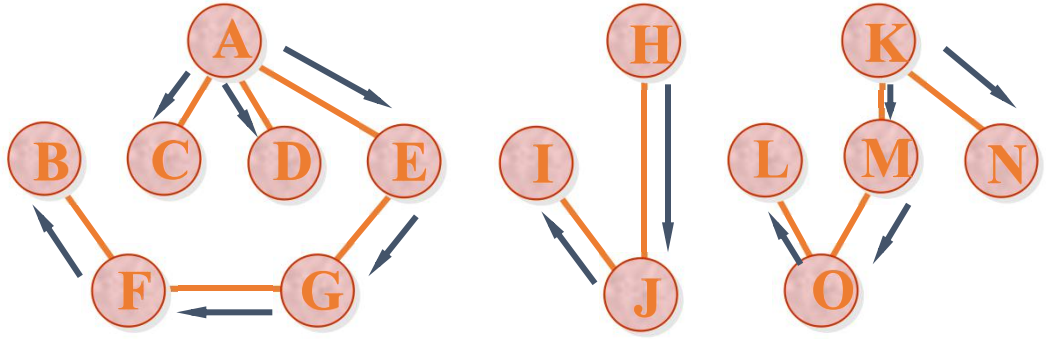
连通分量 (Connected component)

- **对于非连通的无向图，所有连通分量的生成树组成了非连通图的生成森林。**

非连通
无向图

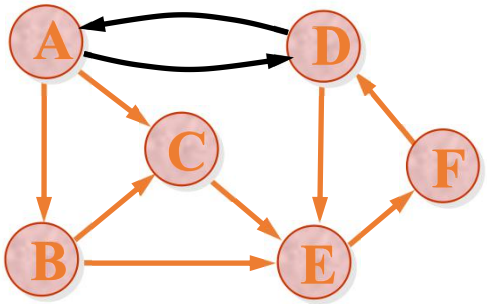


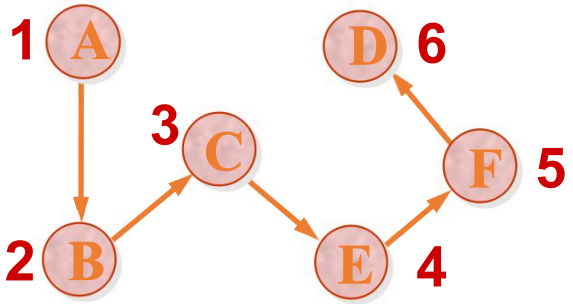
非连通图的
连通分量

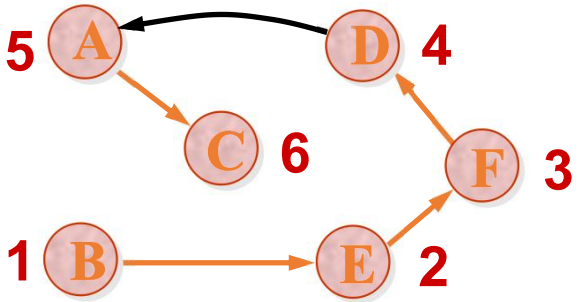


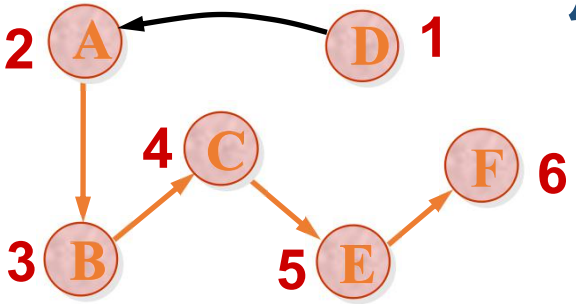
连通分量 (Connected component)

- 从不同顶点出发，通过强连通有向图的遍历，可以得到不同的生成树。非强连通一般得到的是生成森林





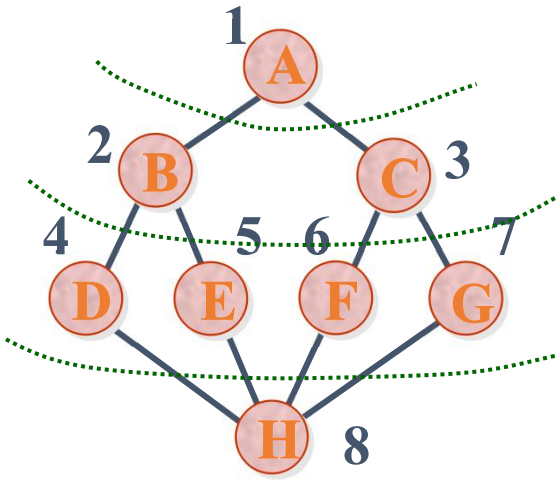




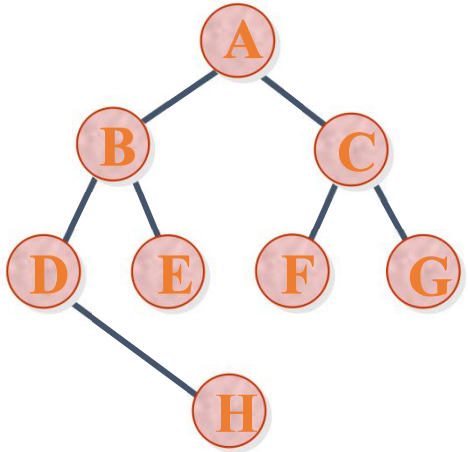
动画

广度优先搜索BFS (Breadth First Search)

■广度优先搜索的示例



广度优先搜索过程



广度优先生成树

广度优先搜索的基本思路

- BFS在访问了起始顶点 v 之后，由 v 出发，依次访问 v 的各个未被访问过的邻接顶点 $w_1, w_2, \ldots, w_t$，然后再顺序访问 $w_1, w_2, \ldots, w_t$ 的所有还未被访问过的邻接顶点。再从这些访问过的顶点出发，再访问它们的所有还未被访问过的邻接顶点，… 如此做下去，直到图中所有顶点都被访问到为止。（与前面树的类似）
- DFS是一种回溯的算法，而BFS不是，它是一种分层的顺序搜索过程，每向前走一步可能访问一批顶点。

广度优先搜索的基本思路

- BFS使用队列，以记忆正在访问的这一层和下一层的顶点，便于向下一层访问。
- 问题：还需要visited[]数组吗？

- 为避免重复访问, 一样需要一个辅助数组 visited [ ]
- 图的遍历主程序与前DFS似，只要在

  void GraphTraverse (AdjGraph* G)　中把调用

DFS (G, i) 改为调用BFS (G, i) 即可。

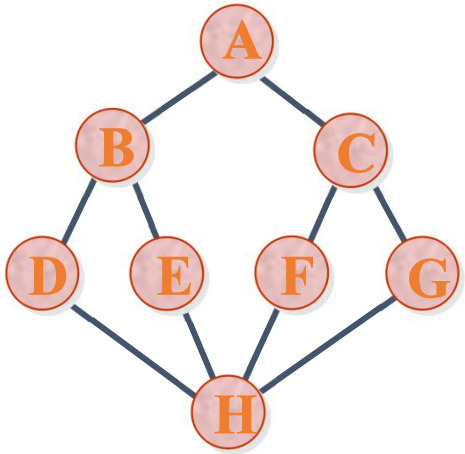
图的广度优先搜索算法

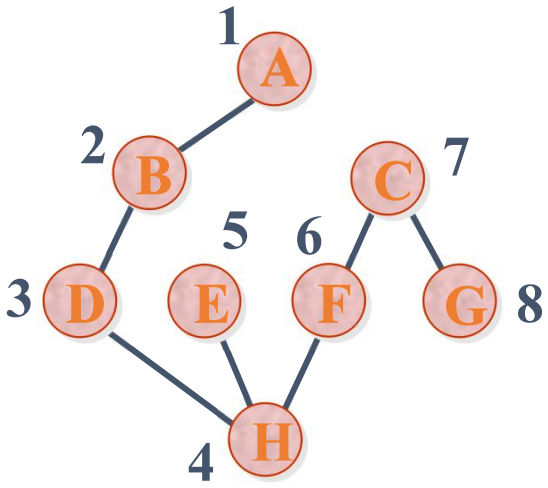

```
void BFS (AdjGraph* G, int v ) {
//在图G中从顶点v出发作广度优先搜索
    visit( v);     //访问顶点v     visited[v] = 1;  //作访问标记
    Queue Q;   InitQueue(Q);          //定义队列
    EnQueue (Q, v);                    //顶点v进队列
    while (!IsEmpty (Q)) {        //队空搜索结束
        DeQueue (Q, v);
        int w = GetFirstNeighbor (G, v);
            //取顶点v的第一个邻接顶点w
```

图的广度优先搜索算法


```
        while ( w != -1 ) {   //若邻接顶点 w 存在
            if ( !visited[w] ) {       //未访问过
                visit(w); //访问w节点
                visited[w] = 1;
                 EnQueue (Q, w);
            }
           w = GetNextNeighbor (G, v, w);//取下一个
        }       //重复检测 v 的所有邻接顶点
    }   //外层循环，判队列空否
}
```

图的遍历算法应用

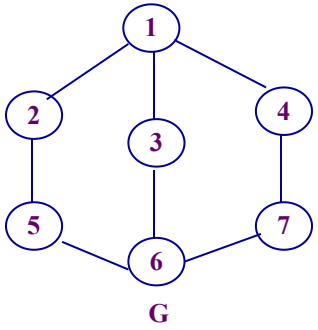




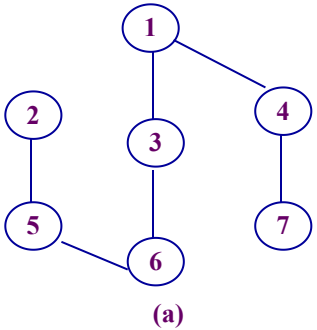
图的遍历可得到线性遍历序列，同时得到生成树

树可以进一步进行遍历得到不同的线性序列

图的遍历算法应用



G



(a)

- 哈密顿路径
- 一笔画问题

题目：求一条包含图中所有顶点的简单路径

思路？

1、是生成树，顶点数为n，有可能不唯一，可通过遍历不同顶点得到

2、由于是简单路径，因此找一条“最深”路径，用DFS

3、该路径上没有回溯。DFS回溯时须设回溯节点未访问

图的遍历算法应用

- 引入全局数组Path，用来保存当前已搜索的简单路径上的顶点，引入全局计数器n用来记录当前该路径上的顶点数。
- 对DFS算法的修改如下：
  - 计数器n的初始化；
  - 访问顶点时，增加将该顶点序号入数组Path中，计数器n++；判断是否已获得所求路径，是则输出结束，否则继续遍历邻接点；
  - 某顶点的全部邻接点都访问后，仍未得到简单路径，则回溯，将该顶点置为未访问，计数器n--

图的深度优先搜索算法


```
void DFSHAM (AdjGraph* G, int v) {
    path[n]=v;  //全局数组
    if (n>=G->n-1 )：输出路径, return;
    n++;   visited[v] = 1;
    int w = GetFirstNeighbor (G, v);
    while ( w != -1 ) {
        if ( !visited[w] ) DFS (G, w);
        w = GetNextNeighbor (G, v, w );
      }
       n--;visited[v] = 0;
}//大体框架如上，有兴趣自行实现
```

图的遍历算法应用

- **思考**

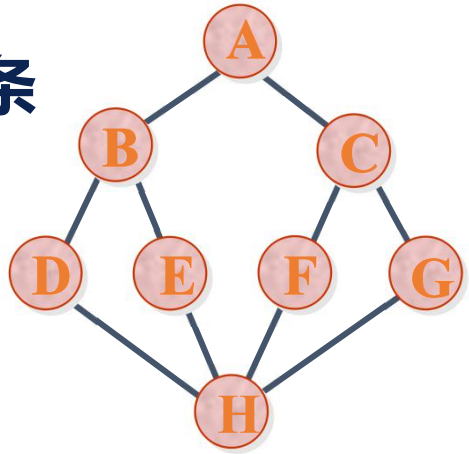
  - 若图中存在多条符合条件的路径，本算法是输出一条，还是输出全部？
  - 如何修改算法，变成判断是否有包含全部顶点的简单路径？
  - 如何修改算法，输出包含全部顶点的简单路径的条数？

图的遍历算法应用

题目：求距某顶点最短路径长度最长的一个顶点

每一个顶点与v之间可能有很多路径，必有一条最短。在所有顶点中找最短路径最长的顶点。

非带权图的路径长度是指此路径上边的条数

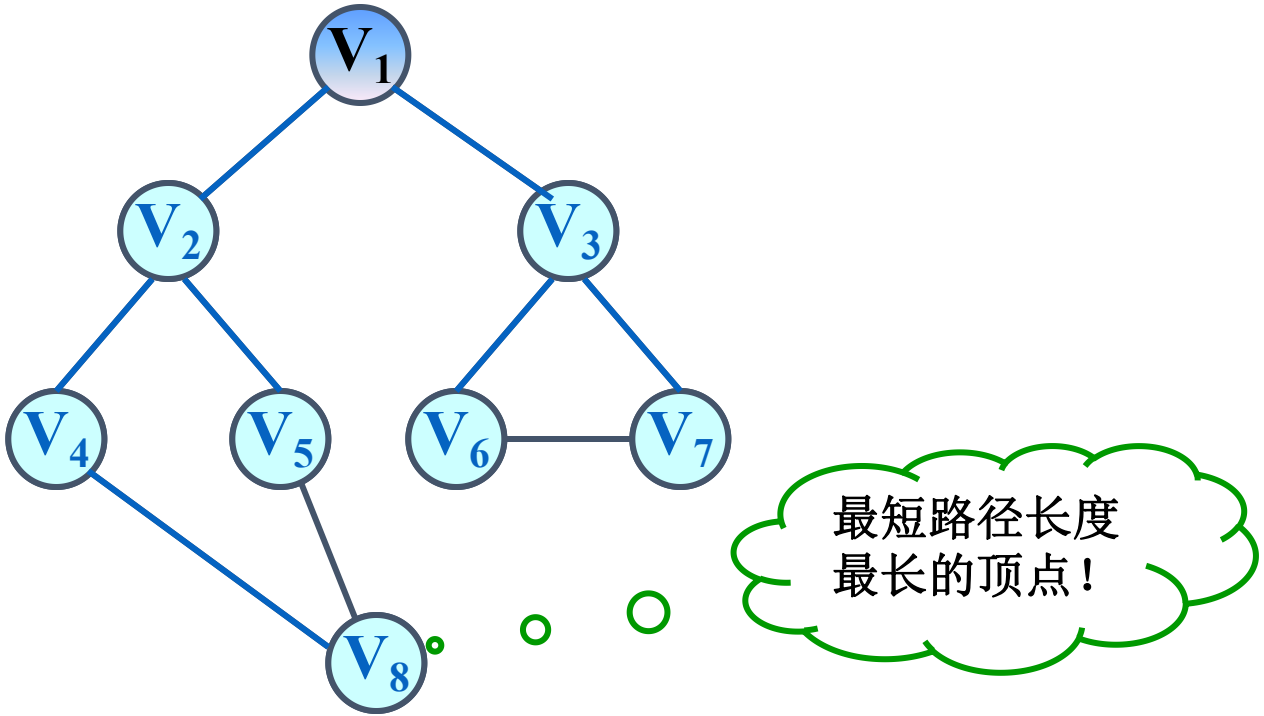


思考：类似DFS找最深顶点嘛？？

3、距离最远=树高最大/层高最大　转换为找最外层节点

图的遍历算法应用

BFS，队列中最后一个出队的顶点必定符合题意



图两种实现方式的简单比较

	A邻接矩阵	B邻接表
存储空间	$O(n+n^2)$	$O(n+e)$
图的创建算法	$T(n)=O(e+n^2)$	$T(n)=O(n+e)$
遍历	$O(n+n^2)$	$O(n+e)$

注：在输入边/弧时，还有一种情况即输入的为顶点本身的信息，而非顶点编号，届时查找时，也是输入顶点本身信息。此时算法复杂度不同。

稀疏图邻接表优，稠密图邻接矩阵优，邻接表求入度不方便。更多比较大家在算法实践时可自行总结。

查找

下午上机

- 图创建+输出图　p20页图例作为输入(邻接表/邻接矩阵)
- 深度优先遍历（邻接矩阵/邻接表）p20页图例作为输入
- 广度优先遍历（邻接矩阵），p20页图例作为输入