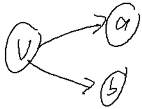
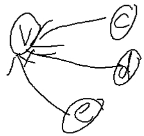
图

出度



V出度=2

入度



V入度=3

V的度=V入度+V出度=5

无向完全图 $e=\frac{n(n-1)}{2}$

有向完全图 $e=n(n-1)$

连通图 任意 V_i 到 V_j 有路径的无向图

强连通图 V_i 到 V_j 和 V_j 到 V_i 有边的有向图

生成树 连通图所有点的极小连通分量 $e=n-1$

如果生成树 减一条边 就不连通

加一条边 就有回路

存储

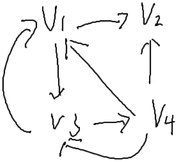
① 邻接矩阵

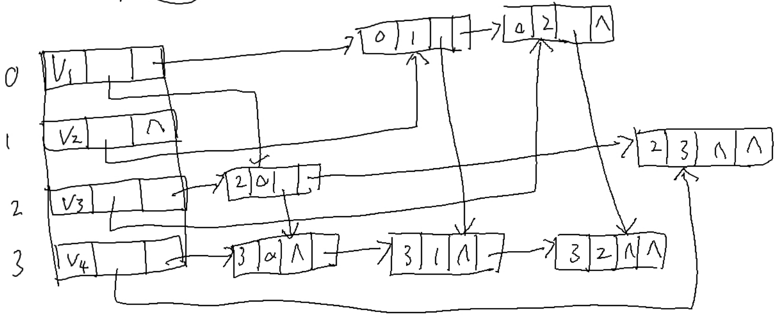
```
typedef struct Cell{
        VRType adj;
}Cell, Matrix[MAX_VSIZE][MAX_VSIZE];

typedef struct Graph{
        VexType vexs[MAX_VSIZE];
        Matrix arcs;
        int vexnum, arcnum;
}Graph;
```

② 邻接表

```
typedef struct ArcNode{
        int adjvex;
        ArcNode *next;
}ArcNode;
typedef struct VNode{
        VexType elem;
        ArcNode *first;
}VNode, List[MAX_VSIZE];
typedef struct Graph{
        List vexs;
        int vexnum, arcnum;
}Graph;
```

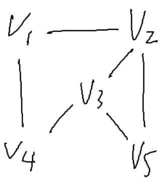
③ 十字链表

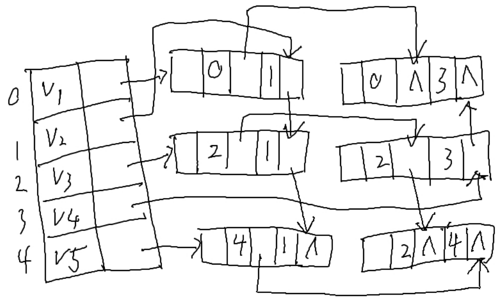





```
typedef struct ArcNode{
    int tvex,hvex;
    ArcNode *hlink,*tlink;
}ArcNode;
typedef struct VexNode{
    VexType elem;
    ArcNode *in,*out;
}VexNode;
typedef struct Graph{
    VexNode vexs[MAX_VSIZE];
    int vexnum,arcnum;
}Graph;
```

④ 邻接多重表






```
typedef struct ENode{
    VisitIf mark;
    int ivex, jvex;
    ENode *ilink, *jlink;
}ENode;
typedef struct VNode{
    VexType elem;
    ENode * first;
}VNode;
typedef struct Graph{
    VNode list[MAX_VSIZE];
    int vexnum, arcnum;
}Graph;
```

深度优先搜索 DFS

时间复杂度 $\left\{\begin{array}{ll}\text{邻接表} & O(V+E) \\ \text{邻接矩阵} & O(V^2)\end{array}\right.$

空间复杂度 $O(V)$

```
void DFSTraverse(Graph G){
    for(v=0; v<G.vexnum; v++)  visited[v]=false;
    for(v=0; v<G.vexnum; v++)
        if(!visited[v])
            DFS(G,v);
}

void DFS(Graph G, int v){
    visit(v);
    visited[v]=true;
    for(w=First(G,v); w>=0; w=Next(G,v,w))
        if(!visited[w])
            DFS(G,w);
}
```

广度优先搜索 BFS

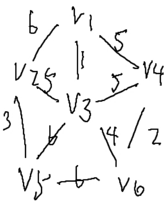
时间复杂度 $\begin{cases} \text{邻接表} & O(V+E) \\ \text{邻接矩阵} & O(V^2) \end{cases}$

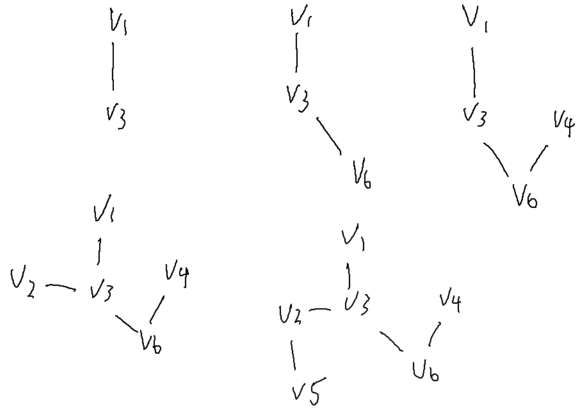
空间复杂度 $O(V)$

```
void BFSTraverse(Graph G){
    for(v=0; v<G.vexnum; v++) visited[v]=false;
    Queue Q;
    for(v=0; v<G.vexnum; v++)
        if(!visited[v]){
            visit(v); visited[v]=true;
            Q.push(v);
            while(!Q.empty()){
                u=Q.pop();
                for(w=First(G,u); w>=0; w=Next(G,u,w)){
                    if(!visited[w]){
                        visit(w); visited[v]=True;
                        Q.push(w);
                    }
                }
            }
        }
}
```

最小生成树 { Prim 算法 / Kruskal 算法

Prim算法





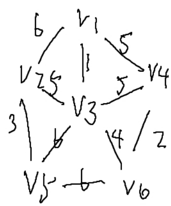
V_2	V_3	V_4	V_5	V_6	U	S
V_1 6	V_1 1	V_1 5	∞	∞	$\{V_1\}$	V_3
V_3 5	0	V_1 5	V_3 6	V_3 4	$\{V_1, V_3\}$	V_6
V_3 5	0	V_6 2	V_3 6	0	$\{V_1, V_3, V_6\}$	V_4
V_3 5	0	0	V_3 6	0	$\{V_1, V_3, V_4, V_6\}$	V_2
0	0	0	V_2 3	0	$\{V_1, V_2, V_3, V_4, V_6\}$	V_5
0	0	0	0	0	$\{V_1, V_2, V_3, V_4, V_5, V_6\}$	

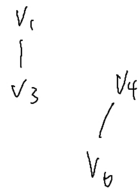
邻接矩阵的Prim实现

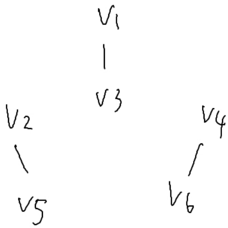
```
typedef struct {   //辅助数组
    VexType adj;
    VRType cost;
} closedge[MAX_VEX_NUM];

void Prim(Graph G, VexType u){
    k=Locate(G,u);  //u在G中的下标
    for(i=0; i<G.vexnum; i++)   //初始化closedge
      if(i!=k)   closedge[i]={u, G.arcs[k][i].adj};
    closedge[k].cost=0;
    for(i=1; i<G.vexnum; i++){
        k=min(closedge);
        // closedge[k].adj 到 G.vexs[k] 是最小生成树的边
        closedge[k].cost=0;
        for(j=0; j<G.vexnum; j++)
            if(G.arcs[k][j].adj < closedge[j].cost)
                closedge[j]={G.vexs[k], G.arcs[k][j].adj};
    }
}
```

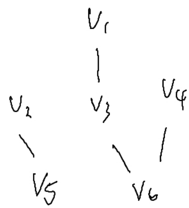
$O(n^2)$

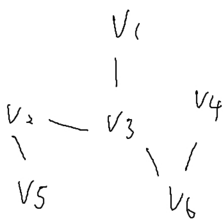
Kraskal算法 $O(e\log e)$











拓扑排序

算法步骤：①选择没有前驱的结点输出
②删除该点和以它为尾的弧
③重复①② 直至所有点被删去，如果没删完，说明有环

可用拓扑排序判断有向图是否有环

```
bool Topo(Graph G){          //邻接表 G
  //indegree[0...vexnum]  记录各点入度
  FindInDegree(G, indegree);   //求各点入度，初始化 indgree
  stack S;
  for(i=0; i<G.vexnum; i++)
      if(indegree[i]==0)   S.push(i);
  count=0;
  while(!S.empty()){
      i=S.pop();   // 输出 G.vexs[i].elem
      count++;
      for(p=G.vexs[i].first; p; p=p->next){
          k=p->adj;
          if((--indegree[k])==0)
              S.push(k);
      }
  }
  if(count<G.vexnum)   return false;  //有环
  else  return true;
}
```

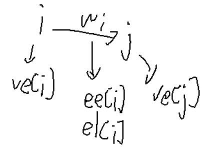
关键路径

$ve \rightarrow ve[j] = \max\{ve[i] + w_i\}$ i为所有指向j的点

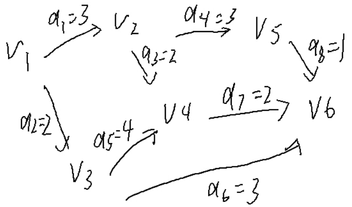
$vl \rightarrow vl[i] = \min\{vl[j] - w_i\}$

$ee \rightarrow ee[i] = ve[i]$

$el \rightarrow el[i] = vl[j] - w_i$



ee = el 为关键活动



	ve	vl
v_1	0	0
v_2	3	4
v_3	2	2
v_4	6	6
v_5	6	7
v_6	8	8

	ee	el
a_1	0	1
a_2	0	0
a_3	3	4
a_4	3	4
a_5	2	2
a_6	2	5
a_7	6	6
a_8	6	7

关键活动 $\{a_2, a_5, a_7\}$

关键路径 $V_1 \xrightarrow{a_2} V_3 \xrightarrow{a_5} V_4 \xrightarrow{a_7} V_6$

关键路径实现

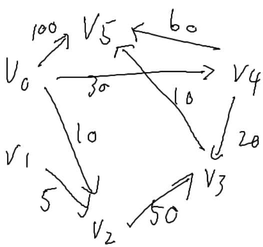
```
bool Topo(Graph G, Stack &T){
    InitInDegree(G, indegree);  Stack S; count=0;
    ve[G.vexnum] = {0};
    for(i=0; i<G.vexnum; i++)
        if(indegree[i]==0)  S.push(i);
    while(!S.empty()){
        i=S.pop();  T.push(i);  count++;
        for(p= G.vexs[i].first; p; p=p->next){
            k = p->adj;
            if((--indegree[k])==0)  S.push(k);
            if(ve[i] + p->cost  > ve[k])
                ve[k]= ve[i] +p->cost;
        }
    }
    if(count < G.vexnum)  return false;
    else  return true;
}
```

// 第一步，求拓扑排序，保存到栈T中

```
void CriticalPath(Graph G){
  if(!Topo(G, T))   return -1;
   vl[G.vexnum] = {ve[G.vexnum-1]};
   while(!T.empty()){
        i = T.pop();
        if(p = G.vexs[i].first ; p ; p = p->next){
            k = p->adj;   dut = p->cost;
            if( vl[k] - dut < vl[i])
                vl[i] = vl[k] - dut;
        }
    }

    for(i=0; i< G.vexnum; i++){
        for(p=G.vexs[i].first; p; p=p->next){
            k = p->adj;   dut = p->cost;
            ee = ve[i] ; el = vl[k] - dut;
            if(ee == el)  //是关键活动
                printf(i, k, dut, ee, el);
        }
    }
}
```

最短路径 单源点 Dijkstra 算法
各个点之间 Floyd 算法

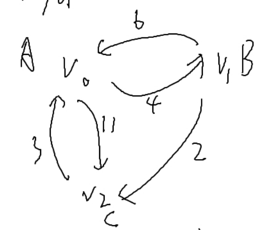
Dijkstra



$$\begin{bmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

	1	2	3	4	5
V_1	∞	∞	∞		∞
V_2	10 (V_0, V_2)				
V_3	∞	60 (V_0, V_2, V_3)	50 (V_0, V_4, V_3)		
V_4	30 (V_0, V_4)	30 (V_0, V_4)			
V_5	100 (V_0, V_5)	100 (V_0, V_5)	90 (V_0, V_4, V_5)	60 (V_0, V_4, V_3, V_5)	
V_j	V_2	V_4	V_3	V_5	
S	$\{V_0, V_2\}$	$\{V_0, V_2, V_4\}$	$\{V_0, V_2, V_3, V_4\}$	$\{V_0, V_2, V_3, V_4, V_5\}$	

Floyd



$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

D

$$D^{(-1)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 0 & 4 & 11 \\ 1 & 6 & 0 & 2 \\ 2 & 3 & \infty & 0 \end{array} \qquad D^{(0)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline & 0 & 4 & 11 \\ & 6 & 0 & 2 \\ & 3 & 7 & 0 \end{array} \qquad D^{(1)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline & 0 & 4 & 6 \\ & 6 & 0 & 2 \\ & 3 & 7 & 0 \end{array} \qquad D^{(2)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline & 0 & 4 & 6 \\ & 5 & 0 & 2 \\ & 3 & 7 & 0 \end{array}$$

P

$$P^{(-1)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & & AB & AC \\ 1 & BA & & BC \\ 2 & CA & & \end{array} \qquad P^{(0)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline & & AB & AC \\ & BA & & BC \\ & CA & CAB & \end{array} \qquad P^{(1)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline & & AB & ABC \\ & BA & & BC \\ & CA & CAB & \end{array} \qquad P^{(2)} \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline & & AB & ABC \\ & BCA & & BC \\ & CA & CAB & \end{array}$$

$D^{(i)}$ $P^{(i)}$ 表示各点经过 i点到其他点的距离和路径

$i=-1$ 表示不经过任何点