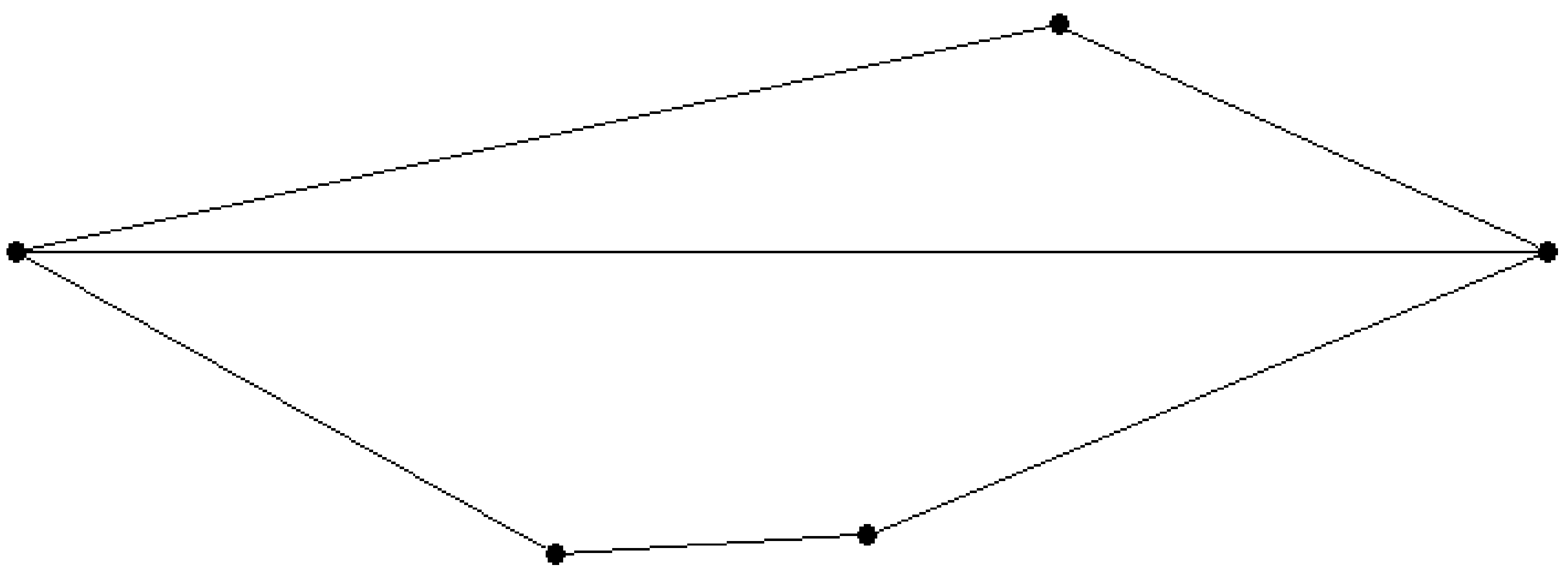
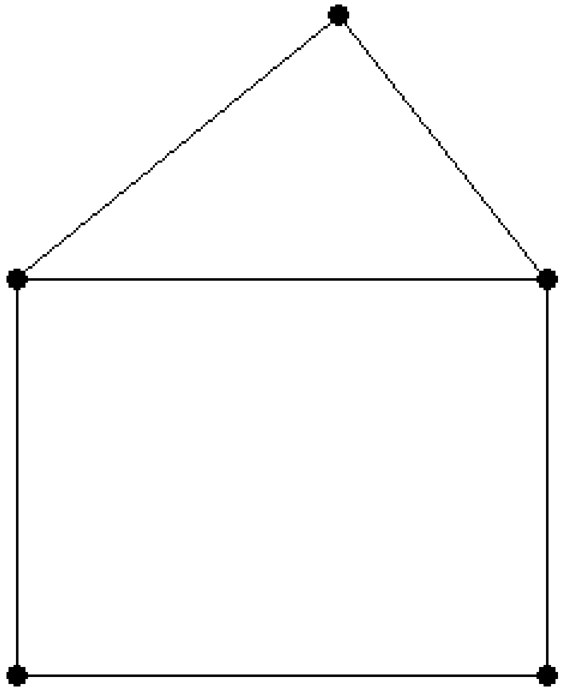
图论的常用算法及应用



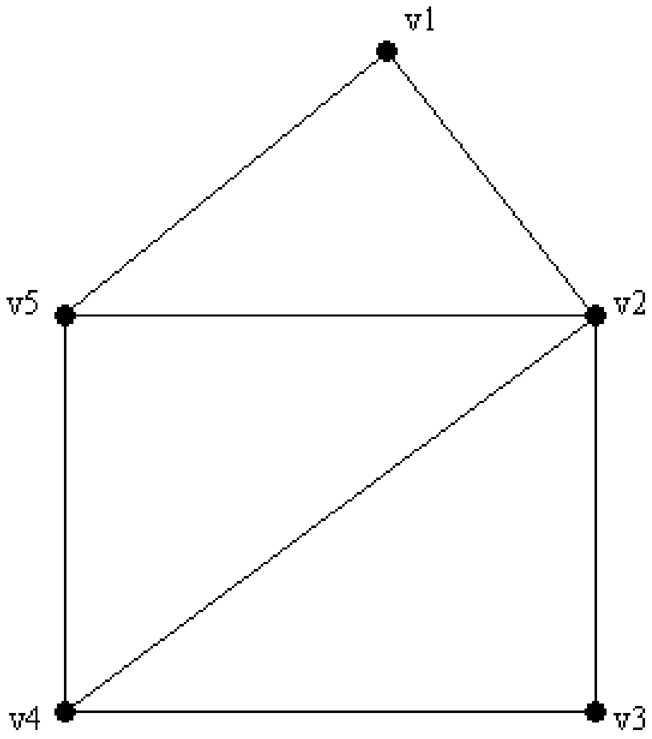
基本概念和性质 1

- 顶点集
- 边集
- 邻接与非邻接
- 点的度数
- 完全图与补图
- 同构

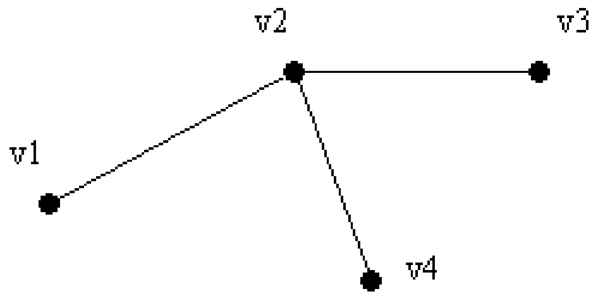


基本概念和性质 2

- 路径　$v_1, v_2, \cdots, v_n$
- 开路
- 回路
- 真路
- 环
- 连接
- 连通与非连通

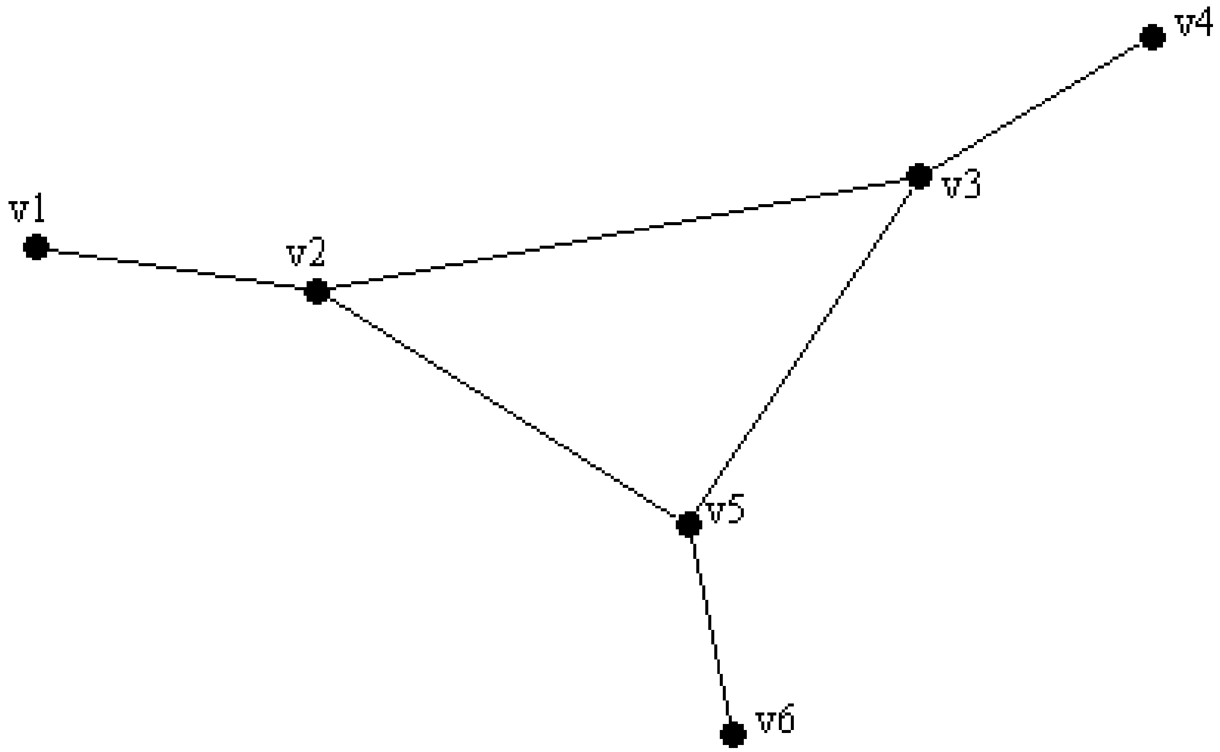


基本概念和性质 3



- 割点
- 割边
- 割边连接的两点是割点
- 两个割点之间的边是割边

一个反例



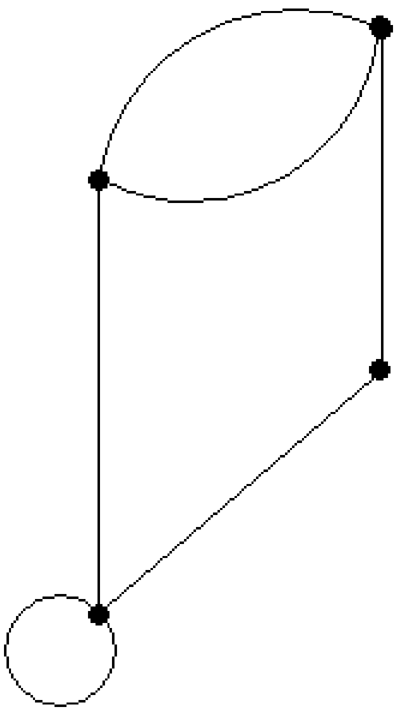
最短路径

- 路径 $v_1, v_2, \cdots, v_n$
- 任意两点的最短路径长度小于 n

  $v_{i1}, v_{i2}, v_{i3}, \cdots, v_{im}$ $m>n$
- 任一环的长度不大于 n

其他一些图

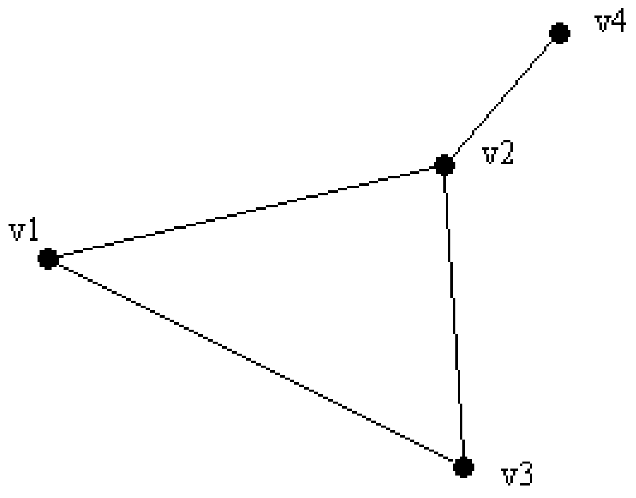
- 多重图
- 伪图
- 有向图
- 带权图
- 欧拉图
- 哈密顿图
- 平面图



图的矩阵表示 1

- 图的邻接矩阵

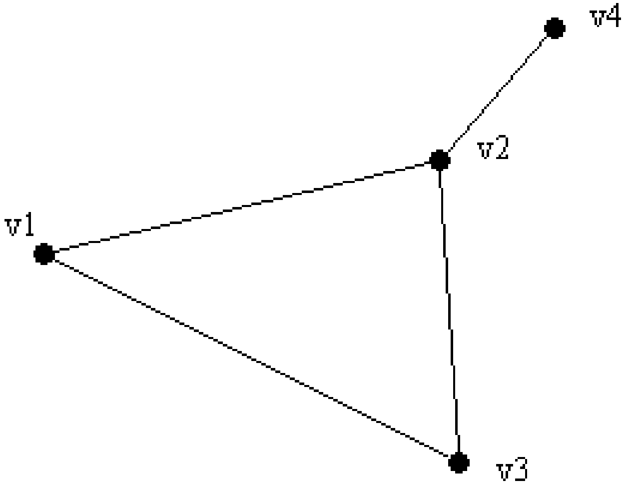
$$A=\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



图的矩阵表示 2

- 图的邻接向量矩阵

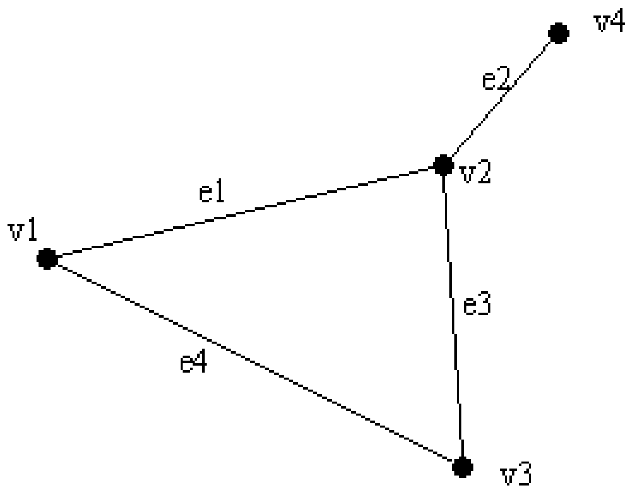
$$A=\begin{pmatrix} ٢ & ٣ & ٠ & ٠ \\ ١ & ٣ & ٤ & ٠ \\ ١ & ٢ & ٠ & ٠ \\ ٢ & ٠ & ٠ & ٠ \end{pmatrix}$$



图的矩阵表示 3

- 图的关联矩阵

$$A=\begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



例题 1

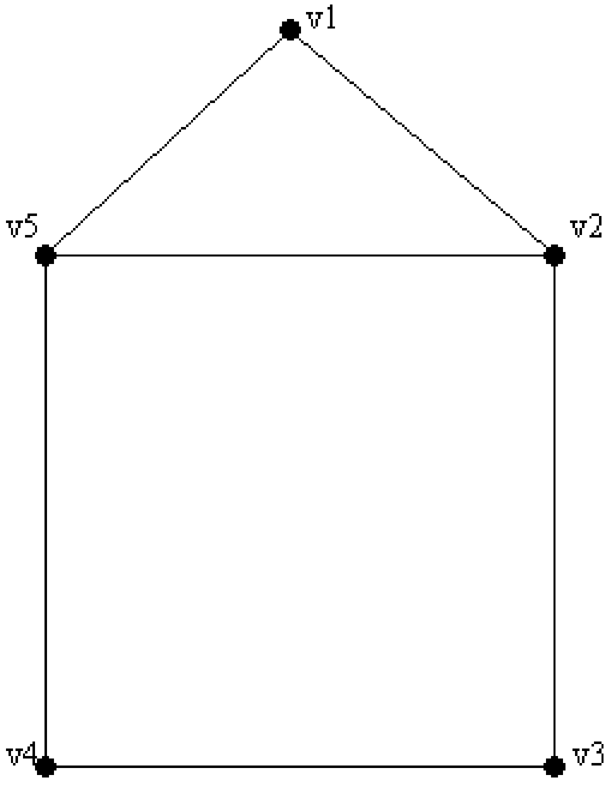
- 给出一张无向简单图的邻接矩阵 A 以及正整数 k，求任意两点之间长度为 k 的路径个数。

例题 1　样例

K=5

求从 v1 到 v3

长度为 5 的路径个数

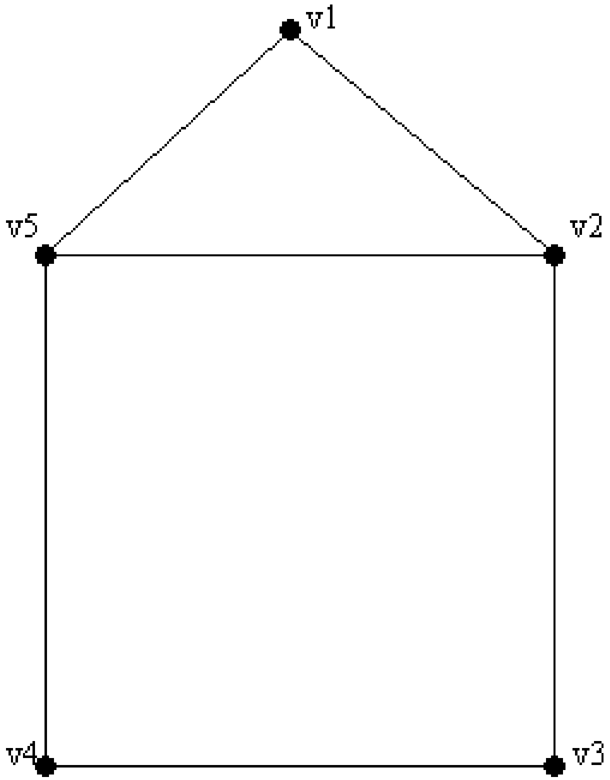


例题 1 求解 1

利用图的邻接矩阵

$A_k(i,j)$ 表示从 i 到 j，

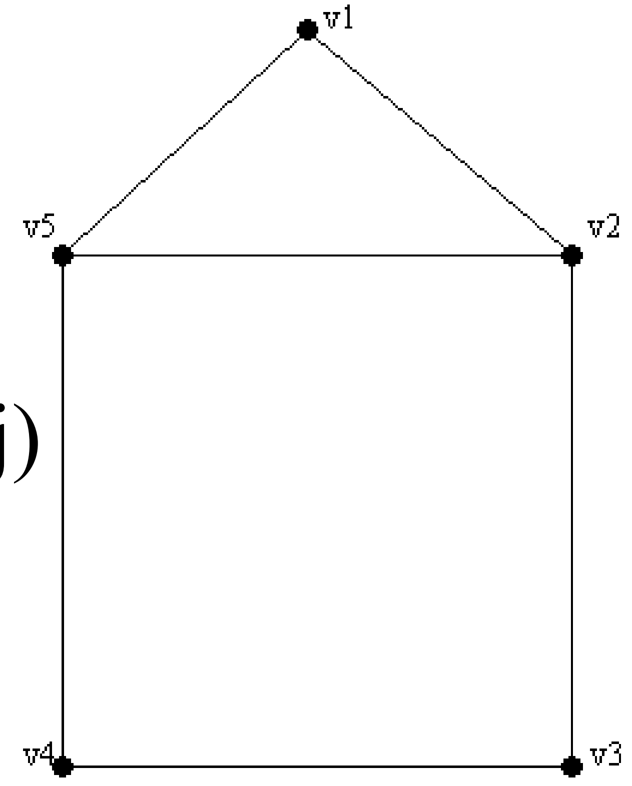
长度为 k 的路径个数



例题 1　求解 2

递推公式：

$$A_k(i,j)=\sum_{l=1}^{n}A_{k\text{-}1}(i,l)\,A_1(l,j)$$



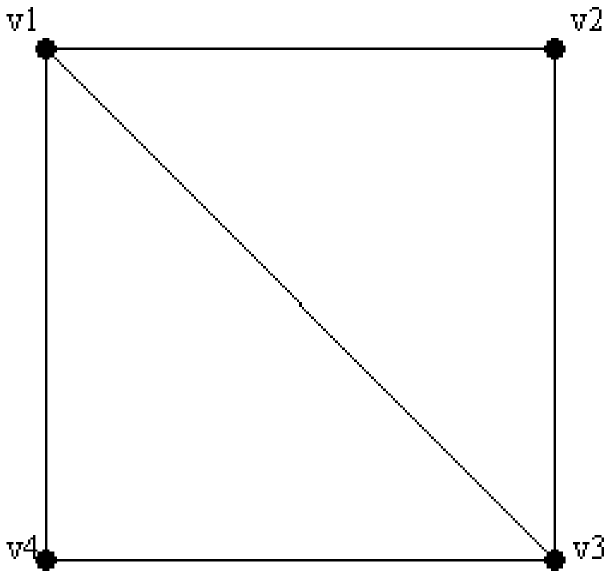
例题 1 程序

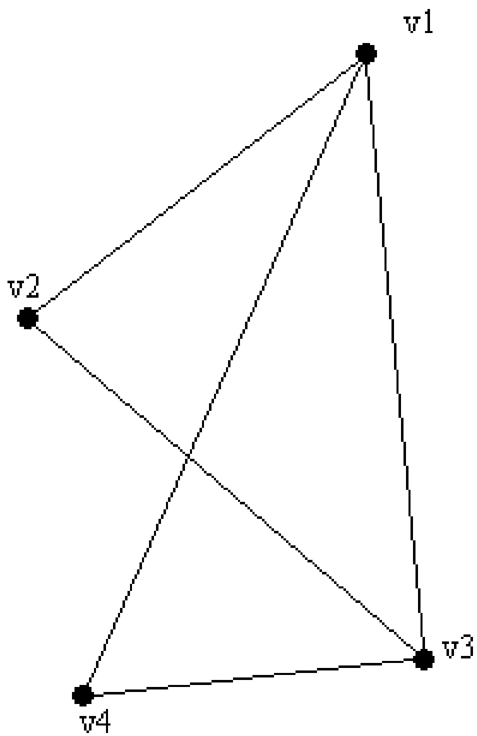
```
for k:=2 to 5 do
    for i:=1 to n do
        for j:=1 to n do
            for l:=1 to n do
  A[k,i,j] := A[k,i,j] +
A[k-1,i,l] * A[l,l,j];
```

例题 2

- 判定图的同构
- 给出两张图的邻接矩阵，判断他们是否同构

例题 2 样例





例题 2　解法 1

- 建立一一映射
- 将图 1 的顶点与图 2 的顶点进行映射

- 一个一维数组 m
- m[i] 表示图 1 中的顶点 i 映射到图 2 中的顶点 m[i]

例题2 程序1

- var
- A,B: array[1..10,1..10] of integer;
- m: array[1..10] of integer;
- n: integer;

例题 2　程序 2

```
function judge: boolean;
var
   i,j: integer;
begin
   judge:=false;
   for i:=1 to n do
       for j:=1 to n do
           if A[i,j] <> B[m[i],m[j]] then exit;
   judge:=true;
end;
```

例题 2　解法 2

- 如何产生这样的一一映射呢？
- 如何产生排列？

例题 2　循环法

```
for m[1]:=1 to 4 do
for m[2]:=1 to 4 do if m[1] <> m[2] then
for m[3]:=1 to 4 do
if (m[3] <> m[2]) and (m[3] <> m[1]) then
for m[4]:=1 to 4 do
   if (m[4] <> m[3]) and (m[4] <> m[2])
      and (m[4] <> m[1]) then
      调用 judge 且判断；
```

例题 2　用递归法模拟循环

- **变量定义：**

```
var
    m:array[1..10] of integer;
    used: array[1..10] of integer;
    n: integer;
```

例题 2　递归子程序

```
procedure work(d:integer);
var
   i:integer;
begin
for i:=1 to n do if used[i] = 0 then
     begin
          used[i] := 1;
          m[d] := i;
          work(d+1);
          used[i] := 0;
     end;
end;
```

例题 2　递归终止条件

```
if d = n+1 then
    begin
        调用 judge 判断；
      exit;
    end;
```

例题 3　最短路径

- 给出一张有向带权图
- 以及两个顶点 a,b　求从 a 到 b 的最短路径长度

- 数据规模：
- 1000 个顶点
- 给出 2000 条边的三元组形式

例题 3　最短路径 算法

- 迪卡斯特拉算法（标号法）
- 适用范围，算法复杂度

- Floyd — Warshall 算法
- 适用范围，算法复杂度

- Bellman — Ford 算法
- 适用范围，算法复杂度

Bellman — Ford 程序解答 1

- 变量定义
- const
- 　　　oo = 15000;
- var
- 　　s,e,l: array[1..2000] of integer;
- 　　dist: array[1..1000] of integer;
- 　　a,b,i,j,k,n,m: integer;

Bellman — Ford 程序解答 2

- for i:=1 to n do
- dist[i] := oo;
- dist[a] := 0;
- for i:=1 to n-1 do
- for j:=1 to m do
- if dist[s[j]] +1[j]<dist[e[j]] then
- dist[e[j]] := dist[s[j]] + 1[j];

Bellman — Ford 思考题

- 如何进一步提高算法的效率
- 如何判断一个图存在负圈
- 应用举例

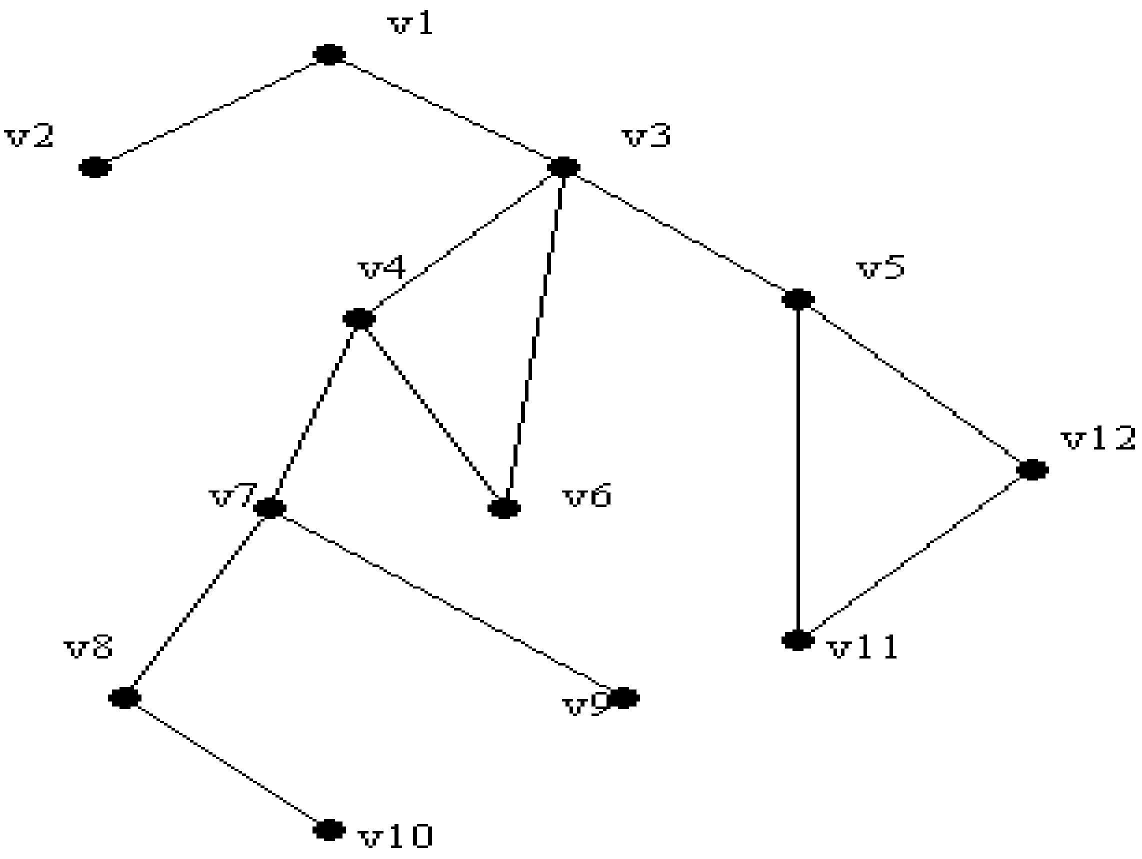
两个基本算法

- 深度优先搜索（Depth First Search）
- 广度优先搜索（Breadth First Search）

深度优先搜索

- 特点
- 复杂度
- 算法和数据结构的实现
- 应用

深度优先搜索的搜索顺序



深度优先搜索算法程序

- var
- n:integer;
- visited: array[1..100] of integer;
- data: array[1..100,1..100] of integer;

深度优先搜索算法程序

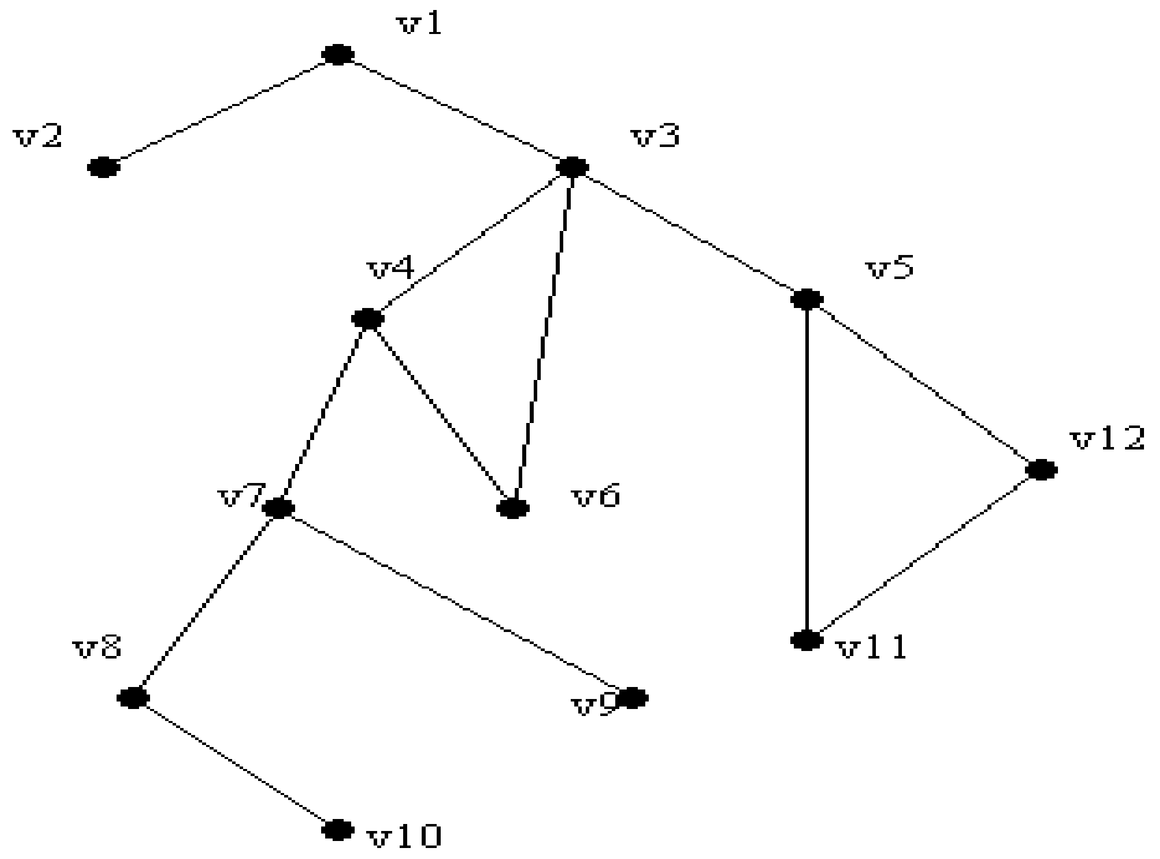
```
procedure dfs(which: integer);
var
   i: integer;
begin
  visited[which] := 1;
  for i:=1 to n do
 if (visited[i] = 0) and (data[which][i])

     then dfs(i);
end;
```

广度优先搜索

- 特点
- 复杂度
- 算法和数据结构的实现
- 应用

广度优先搜索的搜索顺序



广度优先搜索算法程序

- var
- n:integer;
- visited: array[1..100] of integer;
- data: array[1..100,1..100] of integer;

- queue: array[1..100] of integer;
- qh,ql:integer;

广度优先搜索算法程序 1

```
procedure bfs(which: integer);
var
   i: integer;
begin
    queue[1] := which;
    ql:=1;
    qh:=1;
    visited[which] :=1;
```

广度优先搜索算法程序 2

```
while qh<=ql do
    begin
        for i:=1 to n do
            if (visited[i] = 0) and (data[queue[qh]][i])
  then
            begin
                inc(ql);
                queue[ql] := i;
                visited[i] := 1;
            end;
        inc(qh);
    end;
end;
```

例题 4

- 给出一张无向简单图，求此图的割点

- 经典做法
- 简单做法

例题 4　经典做法

- 使用深度优先搜索
- 在搜索过程中计算每个顶点的两个值
- dfn　和　low
- 如何计算

例题4　简单做法1

- 删除某个节点
- 判断是否仍然连通

例题4　简单做法2

- 问题的转换
- 1．是否真的需要删除节点？
- 2．是否真的需要求出所有连通分量？

例题 4　简单做法 3

- 问题的答案
- 1．　否，只要将 visited 数组初始赋 1
- 2．　否，只要 dfs 一个分量

例题 4　解答程序

```
function judge(which: integer) : boolean
var
    i:integer;
begin
    for i:=1 to n do
        visited[i] := 0;
    visited[which] := 1;
    if which = 1 then dfs(2); else dfs(1);
    judge := true;
    for i:=1 to n do
        if visited[i] == 0 then exit;
    judge := false;
end.
```

思考题：

- 如何求割边
- 如何求有向图的强连通分量