7.3 图的遍历--广度优先搜索

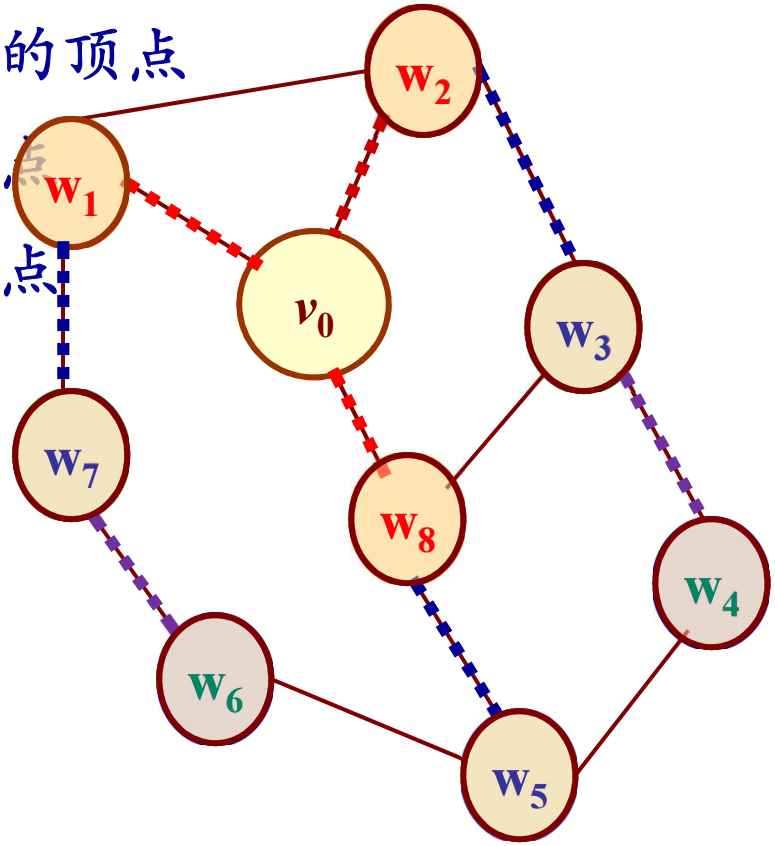
- 基本思想：

1. 初始时图中所有顶点均为未被访问
2. 从图中的某个未访问顶点v_0出发，并在访问此顶点之后依次访问v_0的所有未被访问过的邻接点，之后按这些顶点被访问的先后次序依次访问它们的未访问过的邻接点，直至图中所有和v_0有路径相通的顶点都被访问到。
3. 若此时图中**尚**有顶点未被访问，则另选图中一个**未曾被访问的顶点作起始点**，重复上述过程，直至图中所有顶点都被访问到为止。

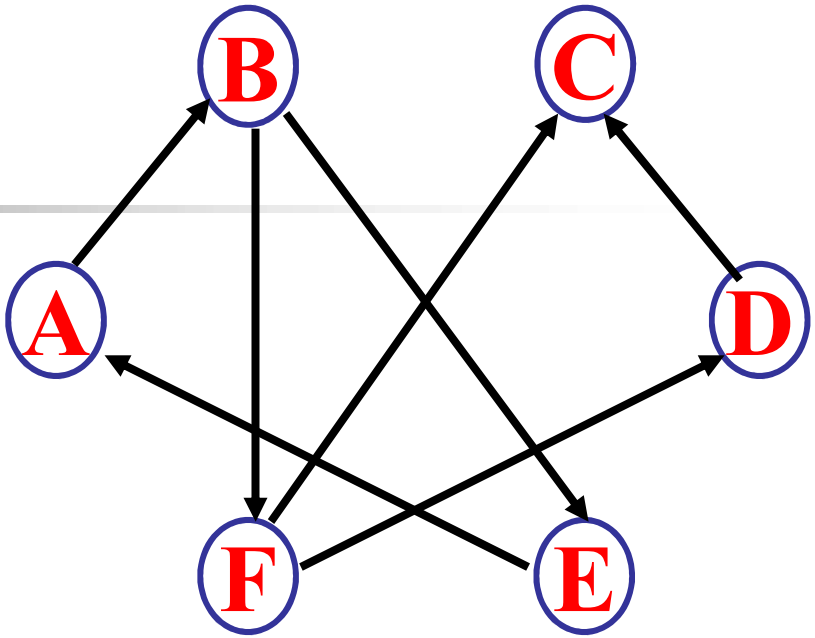
7.3 图的遍历--广度优先搜索

- 另一种解读----按照与出发点v_0路径长度递增的顺序访问顶点：

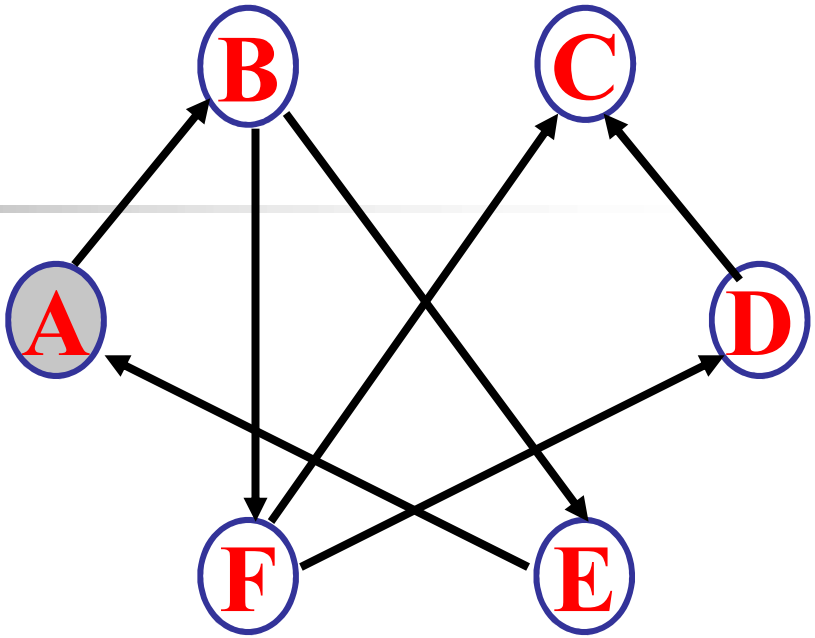
1. 首先访问与出发点v_0路径长度为1的顶点
2. 访问与出发点v_0路径长度为2的顶点
3. 访问与出发点v_0路径长度为3的顶点
4. ……



广度优先搜索

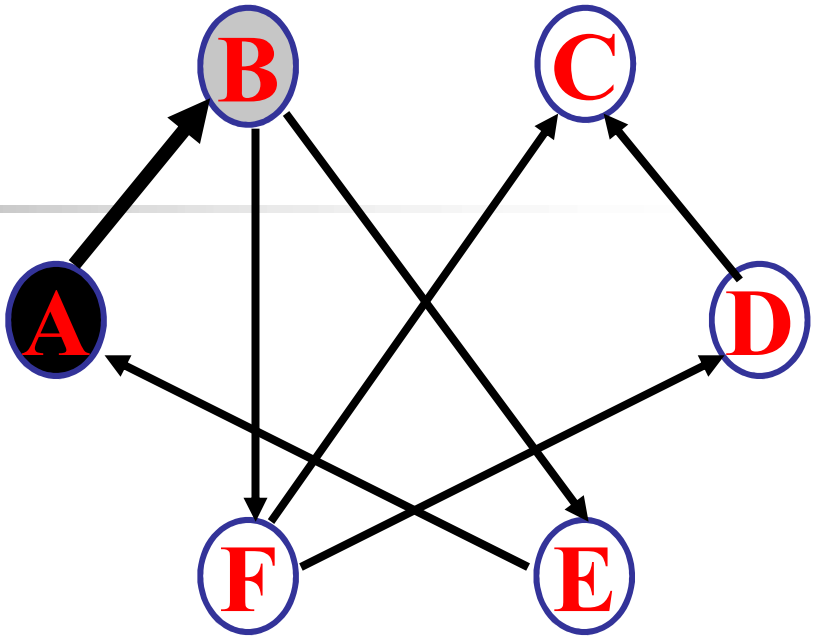


广度优先搜索



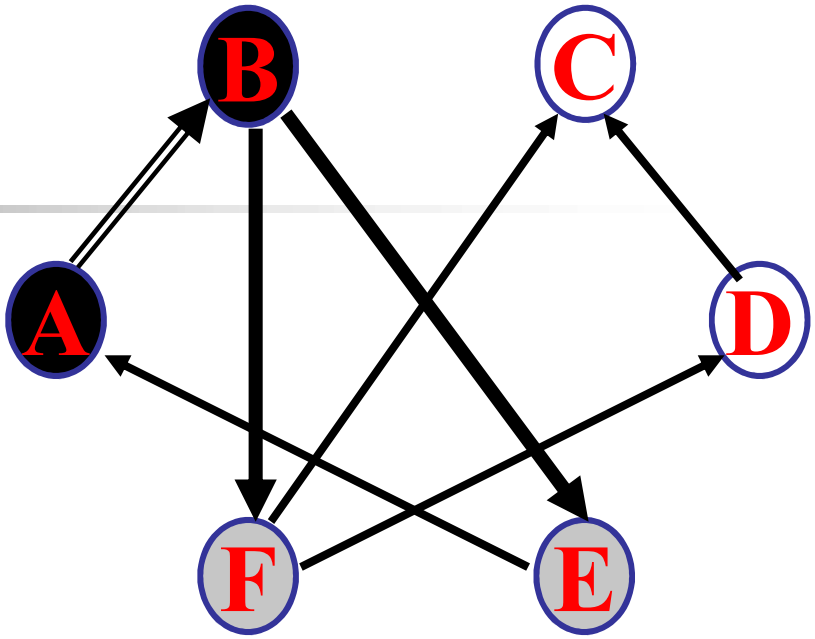
- A

广度优先搜索



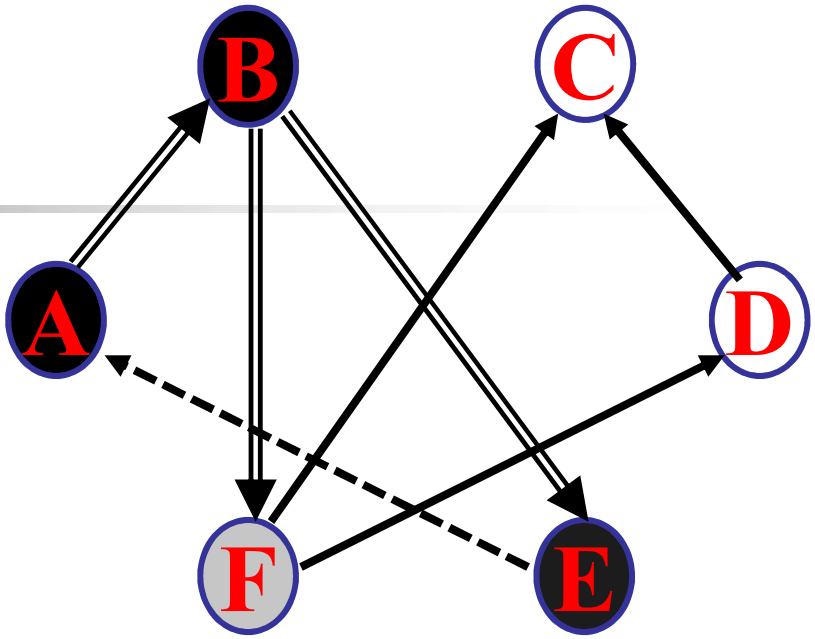
- A,B

广度优先搜索



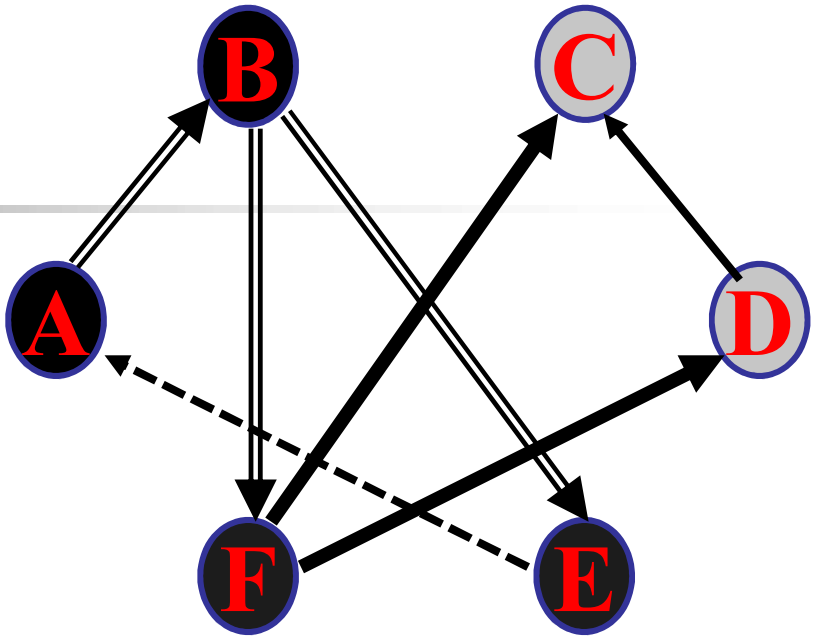
- A,B,E,F

广度优先搜索



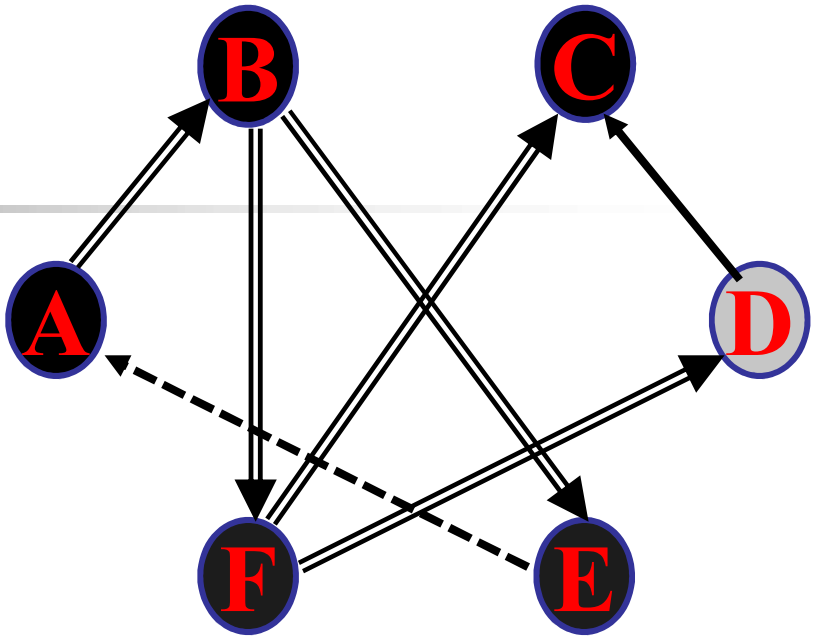
- A,B,E,F

广度优先搜索



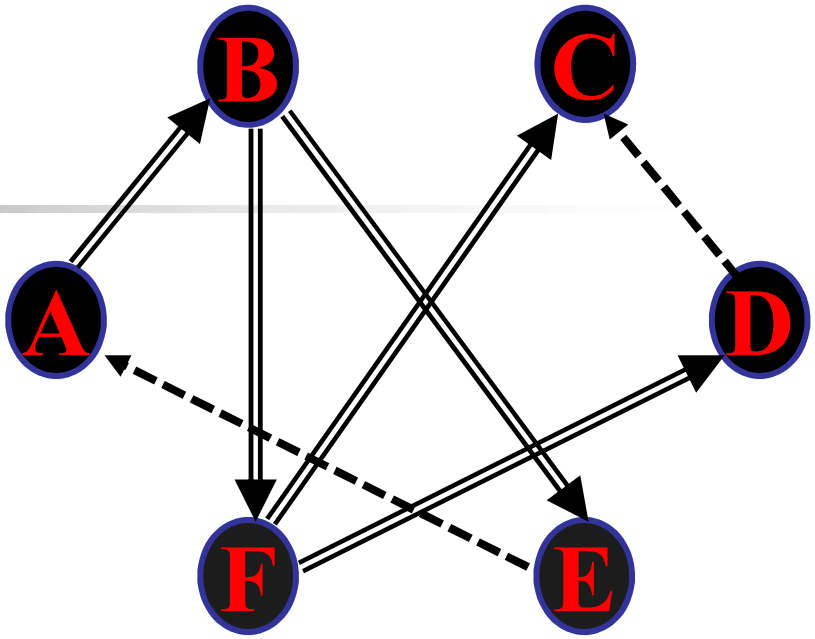
- A,B,E,F,C,D

广度优先搜索



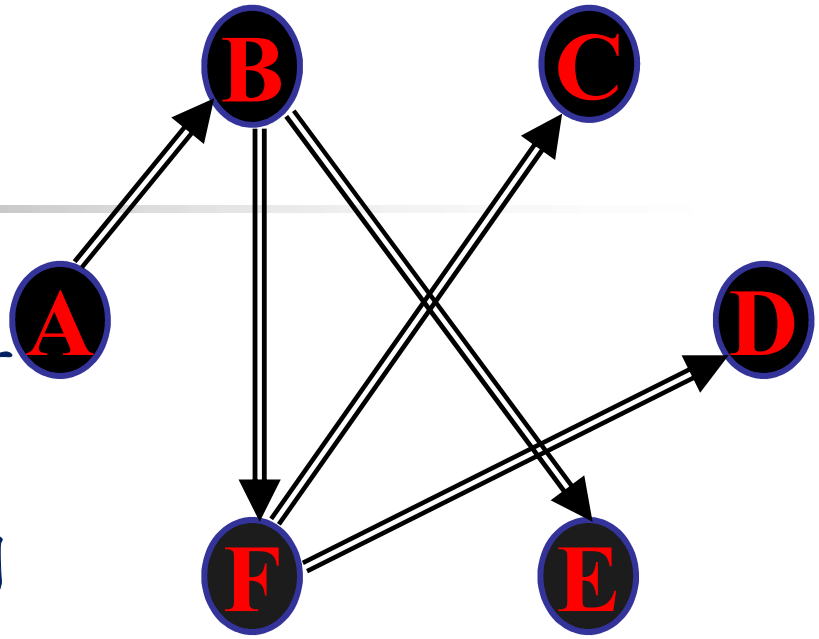
- A,B,E,F,C,D

广度优先搜索



- A,B,E,F,C,D

广度优先搜索

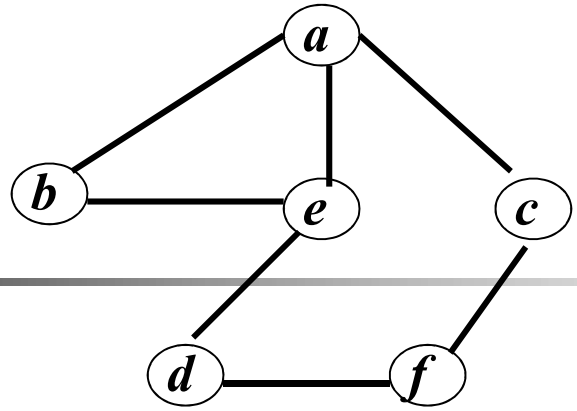


- **广度优先搜索生成树**：访问时经过的顶点和边构成的子图
- **广度优先搜索生成森林**：选用多个出发点做广度优先搜索，会产生多棵广度优先搜索生成树—构成广度优先搜索生成森林
- 对连通图，从起始点v到其余各顶点必定存在路径。按此路径长度递增次序访问

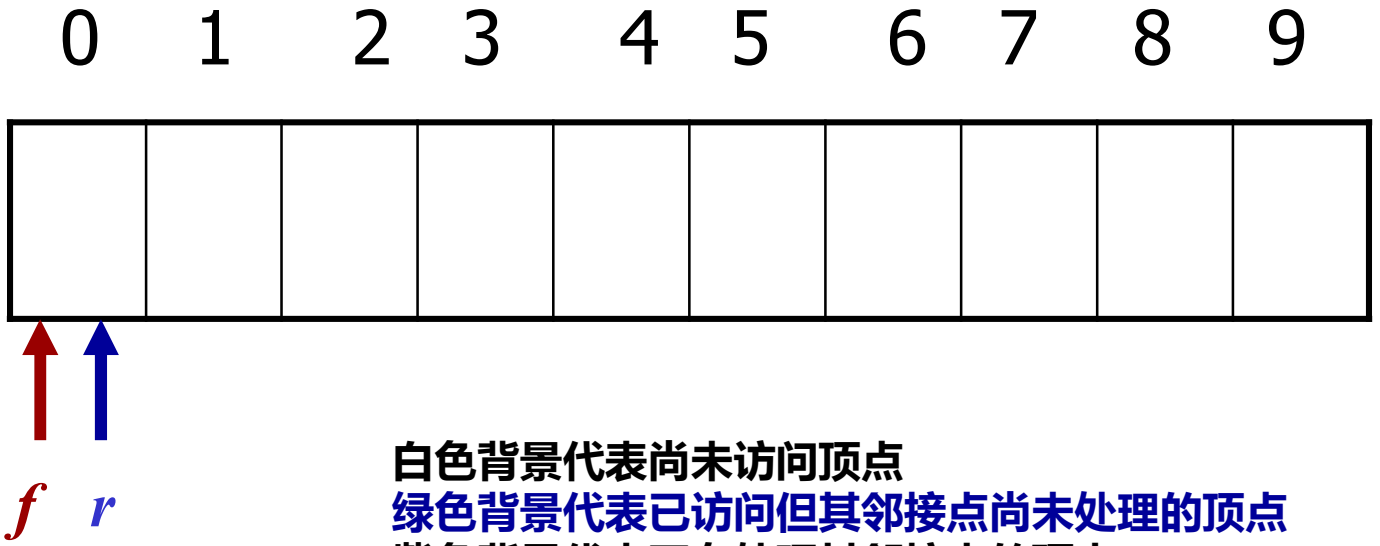
7.3 图的遍历--广度优先搜索

```
void BFSTraverse(Graphs G)
//对图G做广度优先搜索，图采用邻接表存放
{
  for (v=0; v<G.vexnum; ++v) //所有顶点的状态置为未被访问
        visited[v] = 0;
  for ( v=0;  v<G.vexnum;  ++v )
  //依次考察所有顶点，遇到一个未访问的顶点，就从该点出发做广度优先搜索
    if ( !visited[v]) BFS(G, v);
} // BFSTraverse
```

BFS(G,v):队列存放访问过但邻接点未处理的顶点

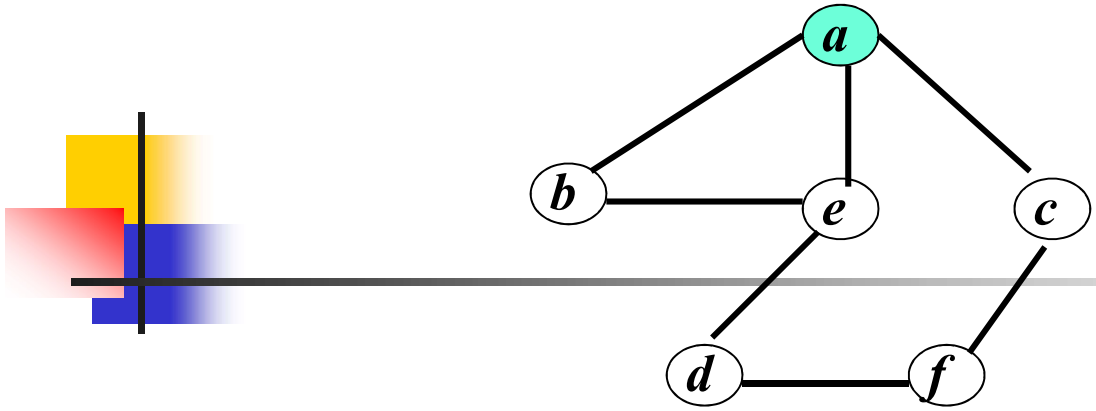


从a出发做广度优先搜索



白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

a

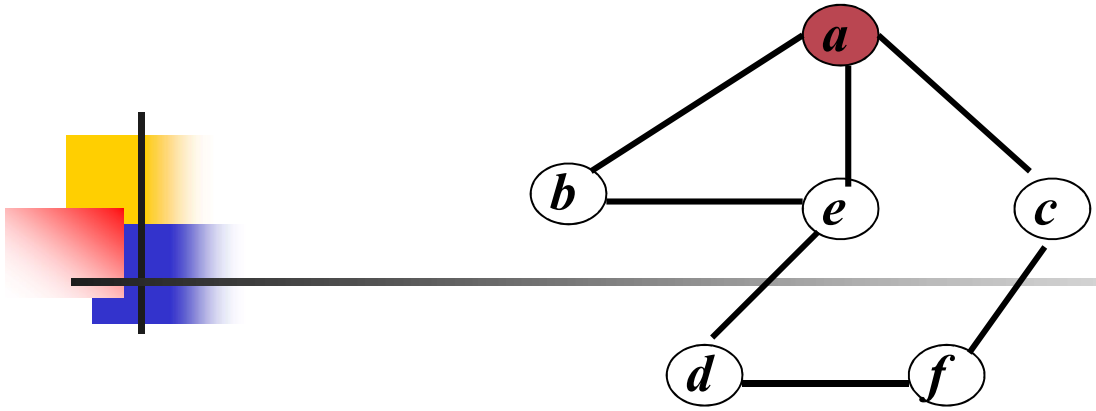
0	1	2	3	4	5	6	7	8	9
a									

f *r*

a

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

a

0 1 2 3 4 5 6 7 8 9

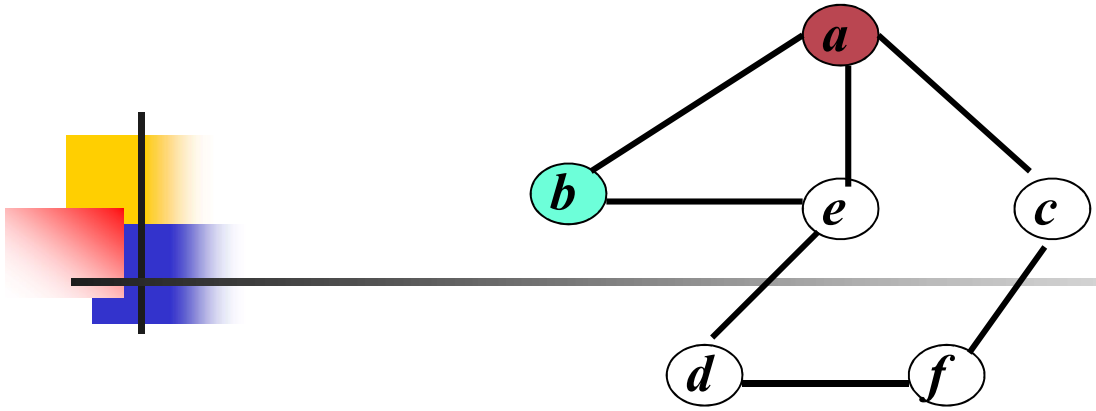
a									

f *r*

a

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

ab

0	1	2	3	4	5	6	7	8	9
a	*b*								

f *r*

ab

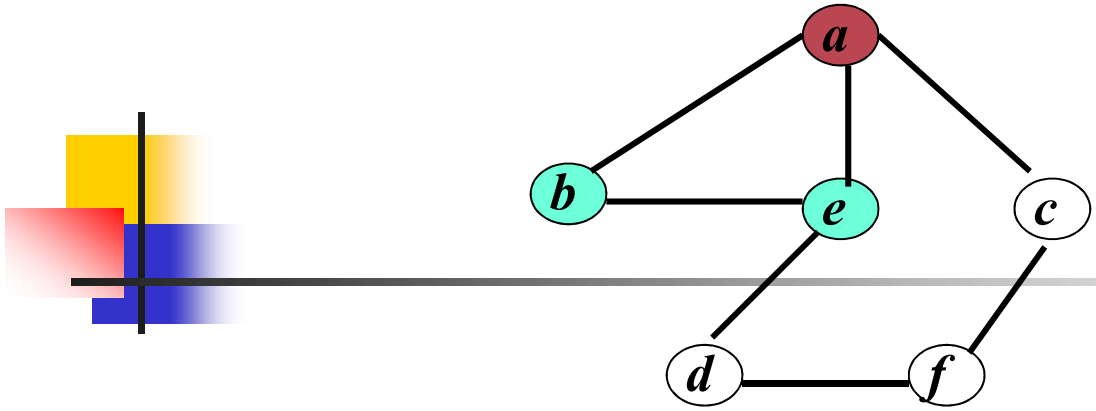
白色背景代表尚未访问顶点

绿色背景代表已访问但其邻接点尚未处理的顶点

紫色背景代表正在处理其邻接点的顶点

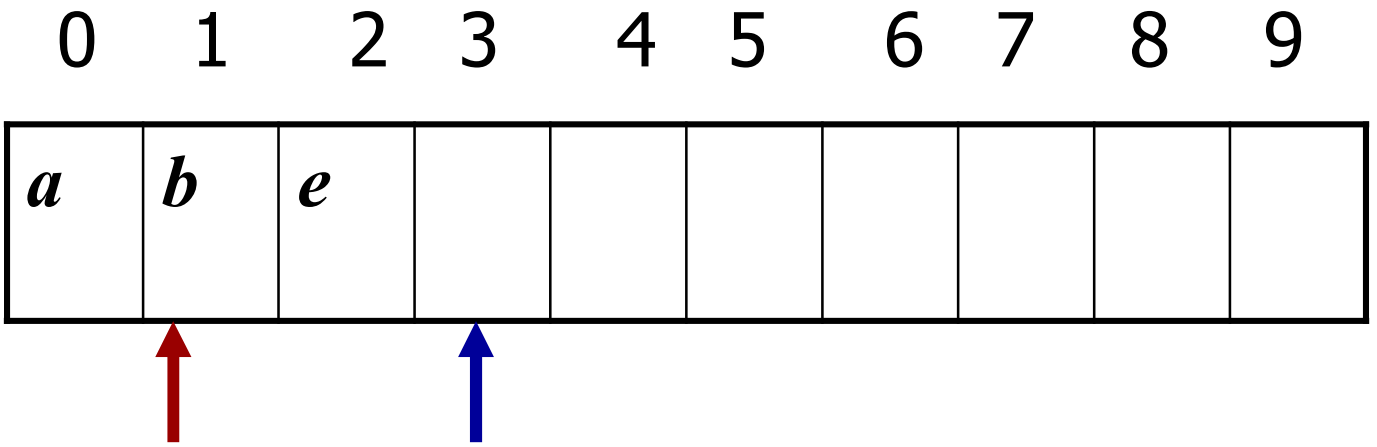
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

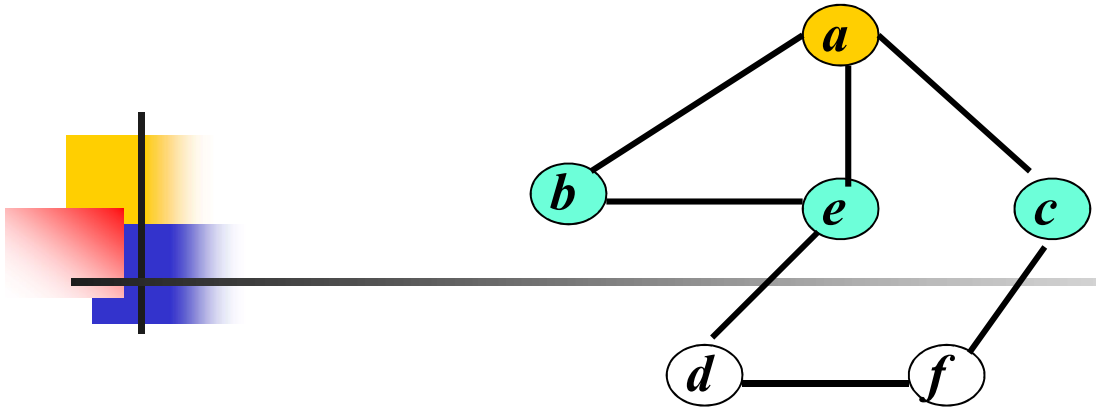
abe



abe

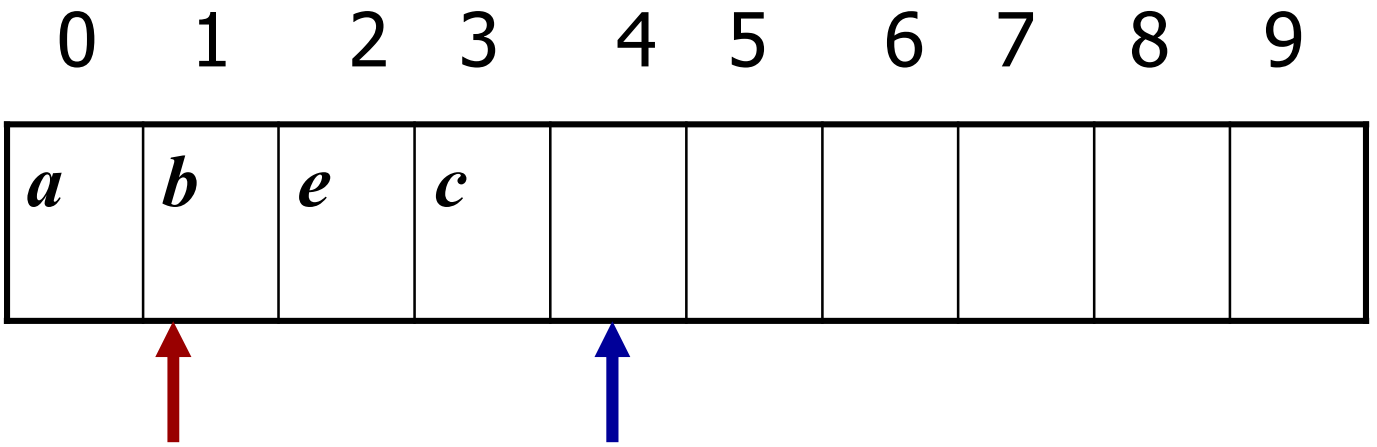
白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

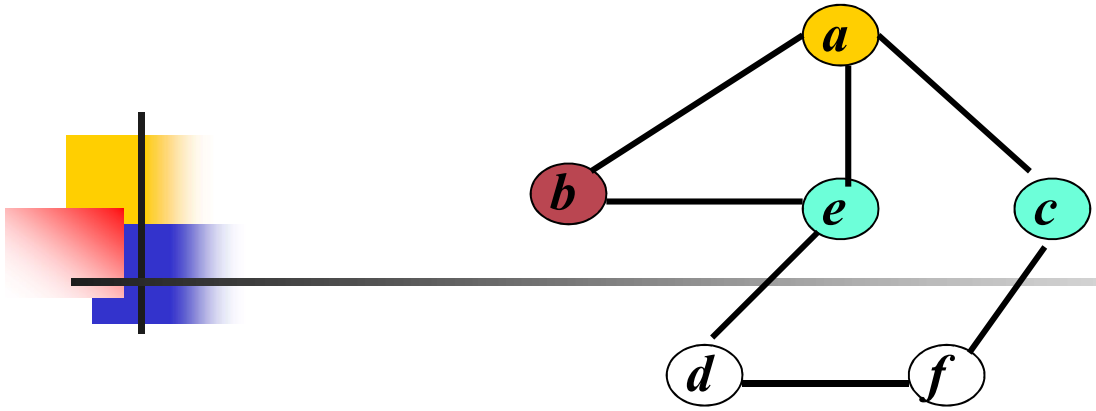
abec



abec

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

abec

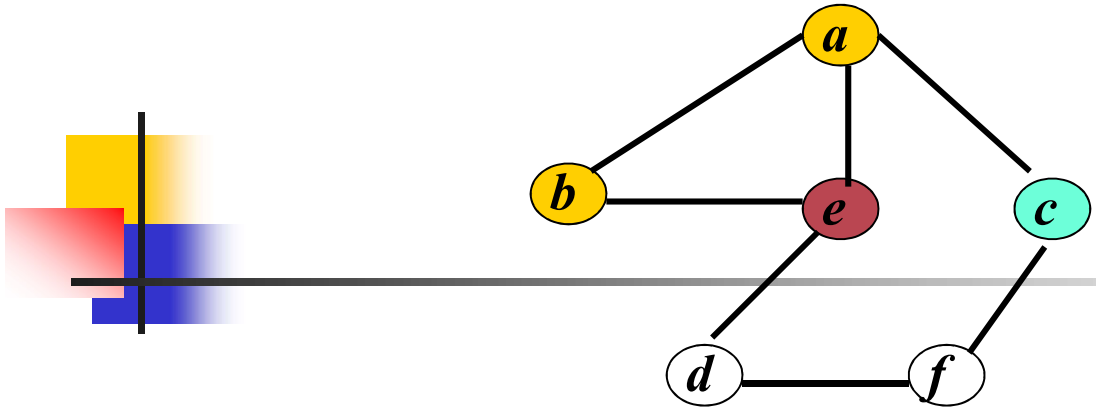
0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*						

f *r*

abec

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

abec

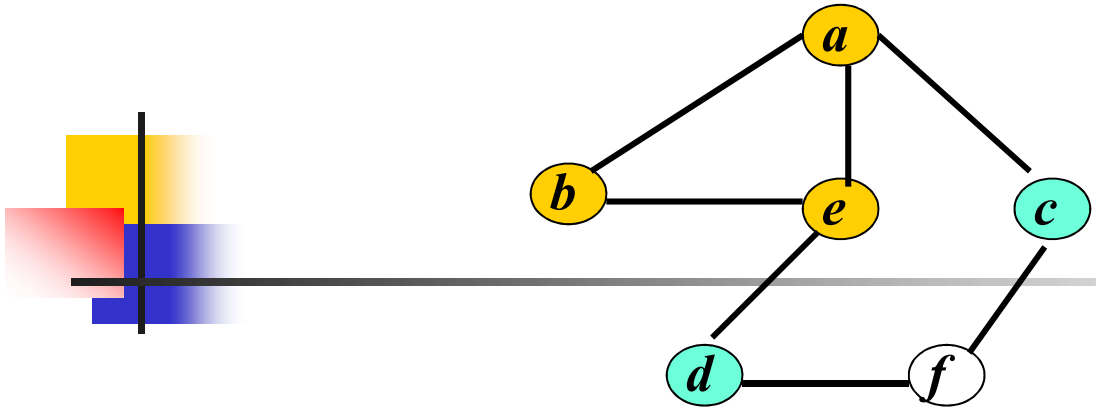
0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*						

f *r*

abec

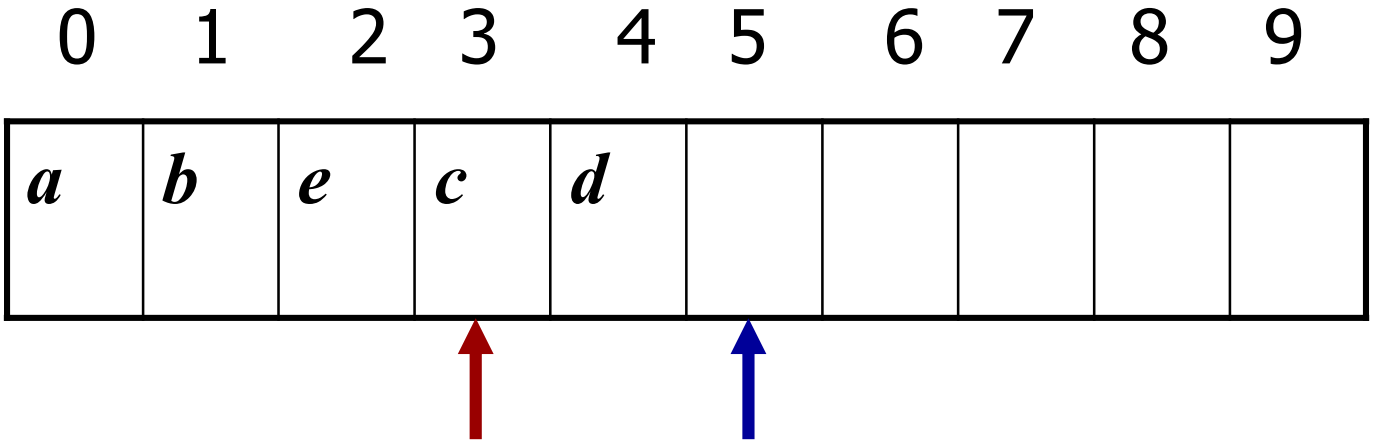
白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

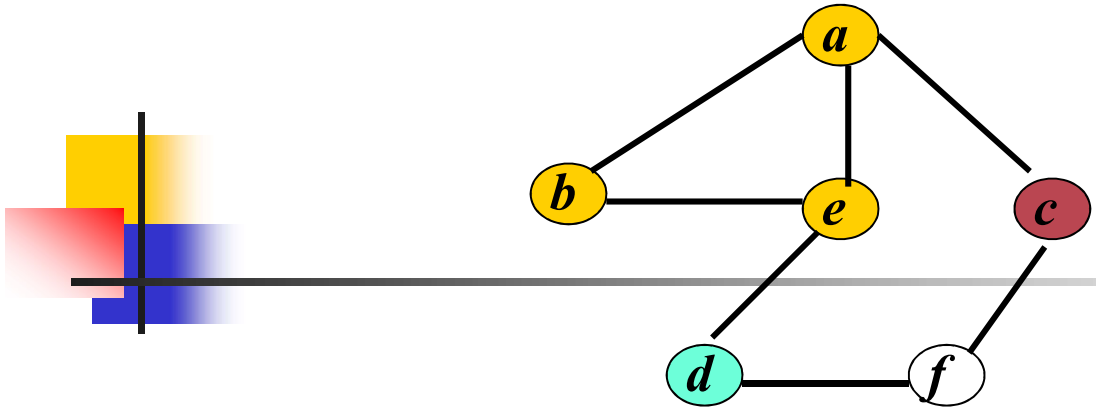
abecd



abecd

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

abecd

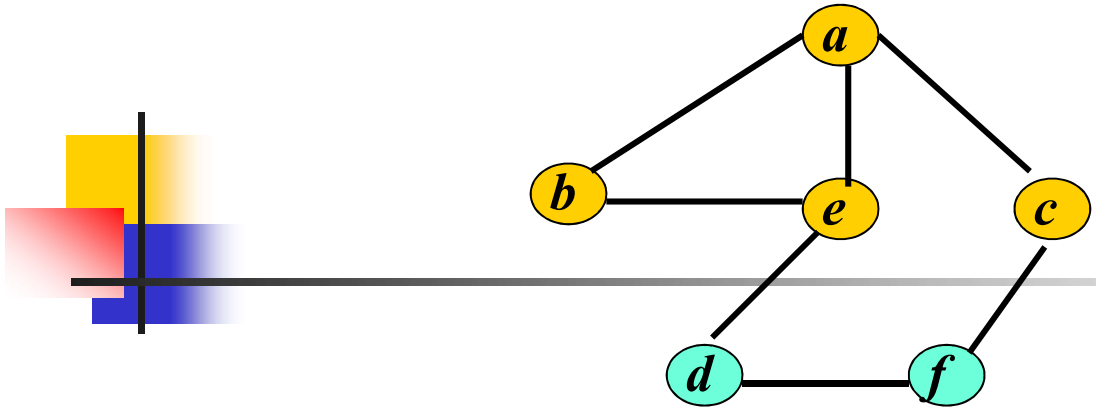
0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*	*d*					

f *r*

abecd

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

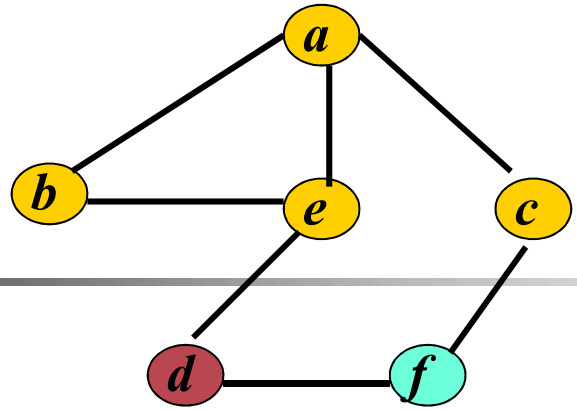
abecdf

0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*	*d*	*f*				

f *r*

abecdf

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点



从a出发做广度优先搜索

abecdf

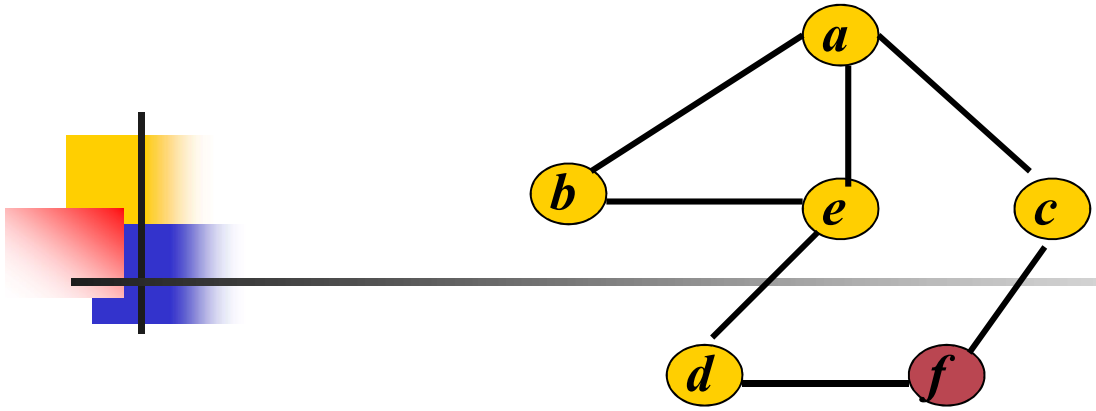
0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*	*d*	*f*				

f *r*

abecdf

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

abecdf

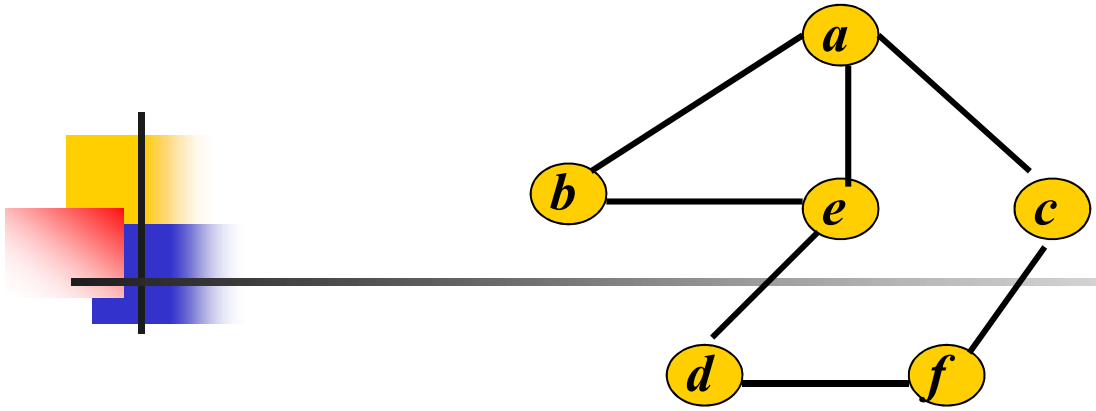
0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*	*d*	*f*				

f r

abecdf

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

BFS(G,v):队列存放访问过但邻接点未处理的顶点



从a出发做广度优先搜索

abecdf

0	1	2	3	4	5	6	7	8	9
a	*b*	*e*	*c*	*d*	*f*				

f r

abecdf

白色背景代表尚未访问顶点
绿色背景代表已访问但其邻接点尚未处理的顶点
紫色背景代表正在处理其邻接点的顶点
黄色背景代表邻接点已经处理结束顶点

```
void BFS(Graphs G,int v) //对图G从v出发做广度优先搜索
{ int q[MAXSIZE]; int f=r=0; //初始化一个空队列
   visited[v] = 1; printf("%d",v);//访问v
   q[r++]=v; //刚刚访问的顶点v入队
   while(f!=r)
   {
       w=q[f++]; //取队首顶点w（w已经访问过，现在要访问w的未被访问的邻接点）
       for(p=G.arc[w].firstarc; p!=NULL; p=p->link)
      {  k=p->vex;
        if(!visit[k]){visited[k]=1;  printf("%d",k);
                        if(r==MAXSIZE) exit(-2);
                        else q[r++]=k;}
      }
   }
} // BFSTraverse
```

将此算法改成循环队列

7.3 图的遍历——广度优先搜索应用

- 判断从顶点 i 到顶点 s 是否存在简单路径
- 判断一个图是否为连通图
- 求两个顶点之间的一条路径长度最短的路径

若两个顶点之间存在多条路径，则其中必有一条路径长度**最短**的路径。如何求得这条路径?