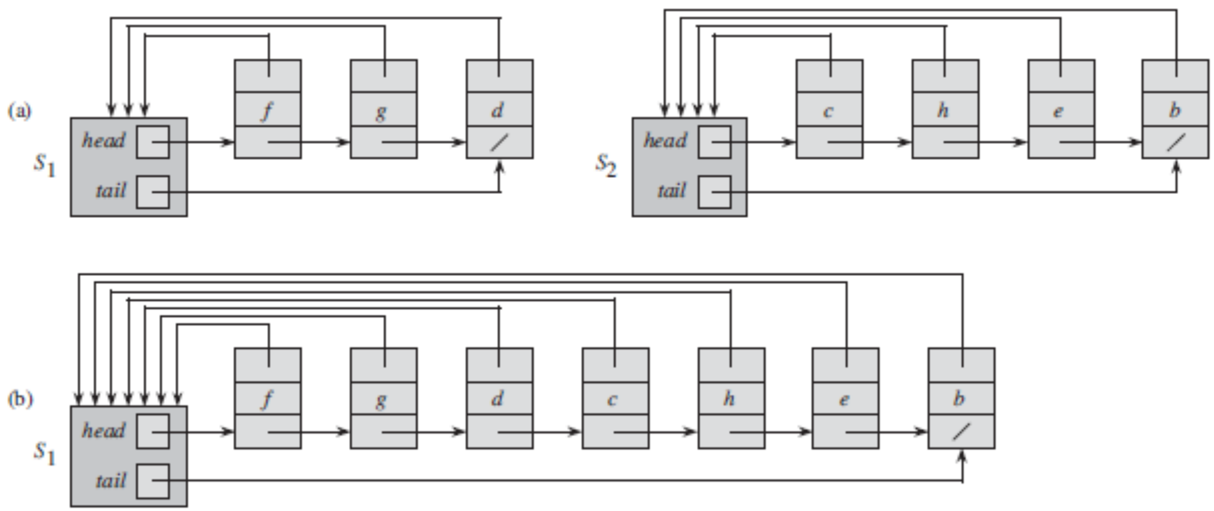
- 作业讲解
  - TC第21.1节练习2、3
  - TC第21.2节练习1、3、6
  - TC第21.3节练习1、2、3
  - TC第21章问题1

TC第21.1节练习2

- Two vertices are in the same connected component if and only if they are in the same set.
  - in the same connected component → in the same set
  - in the same set → in the same connected component

TC第21.2节练习1

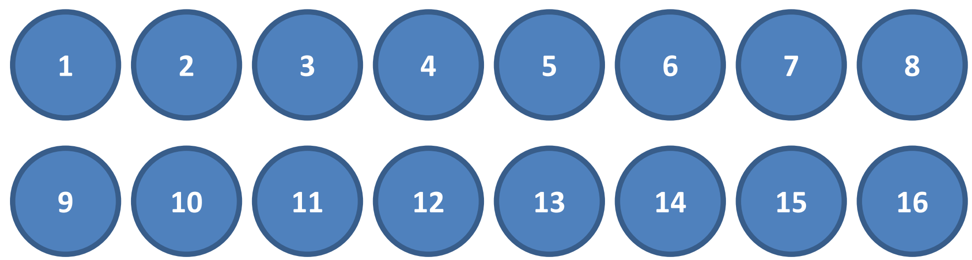
- UNION之后，需要维护哪些指针？



TC第21.3节练习1

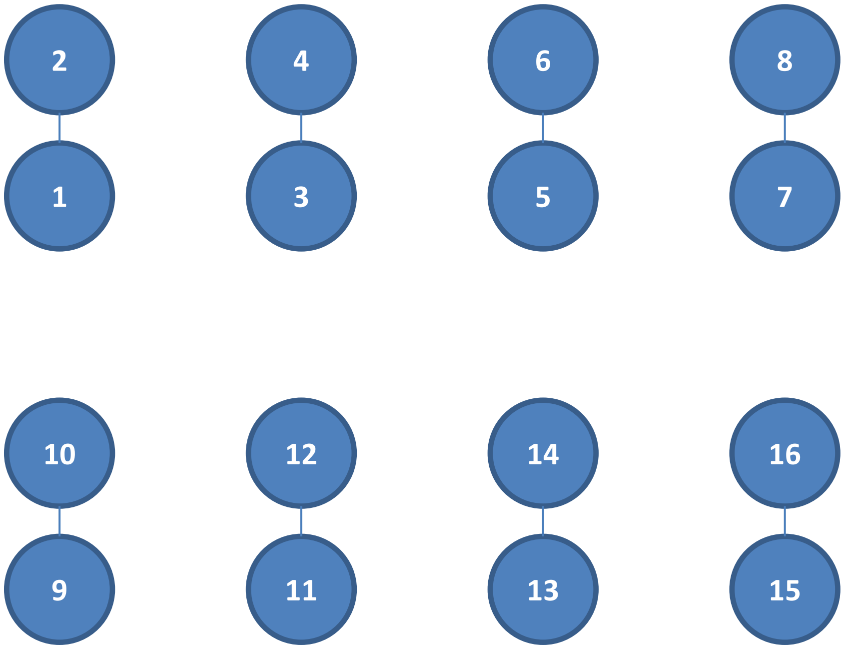
- for i=1 to 16
  - MAKE-SET(x_i)

如果x_i和x_j等长时，把x_i连接到x_j



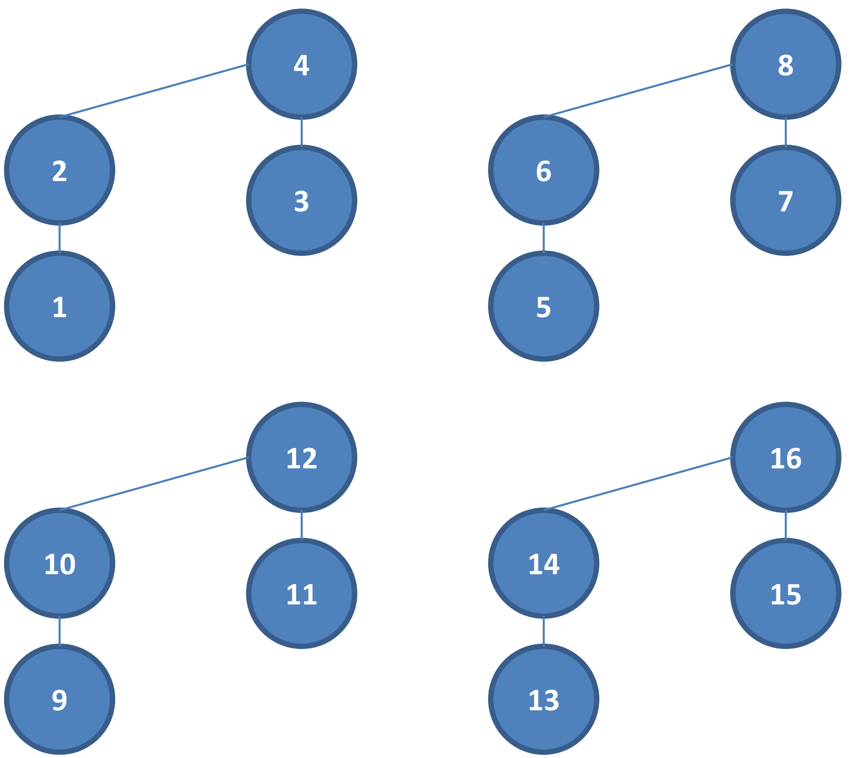
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})



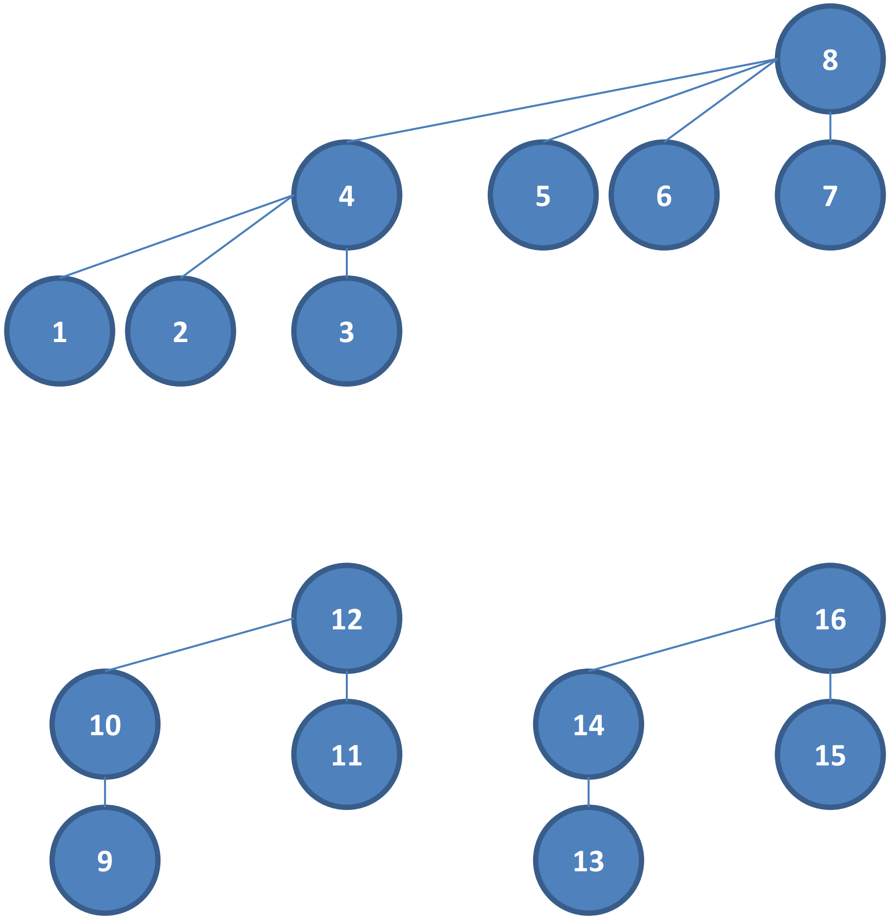
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})



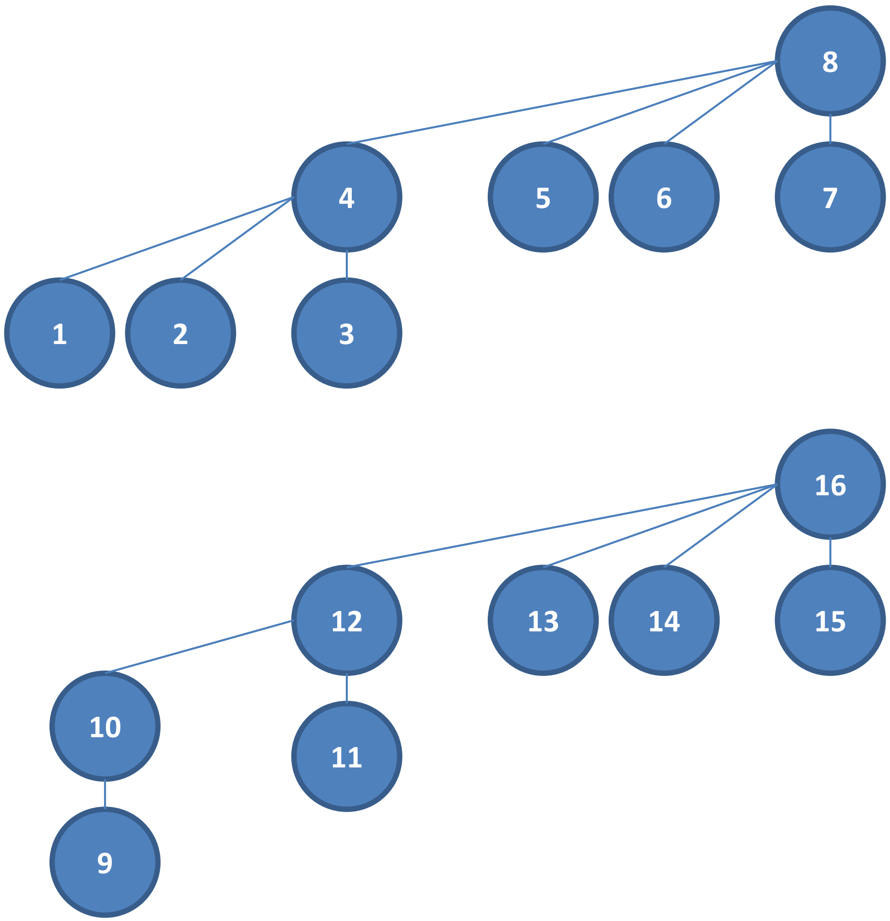
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)



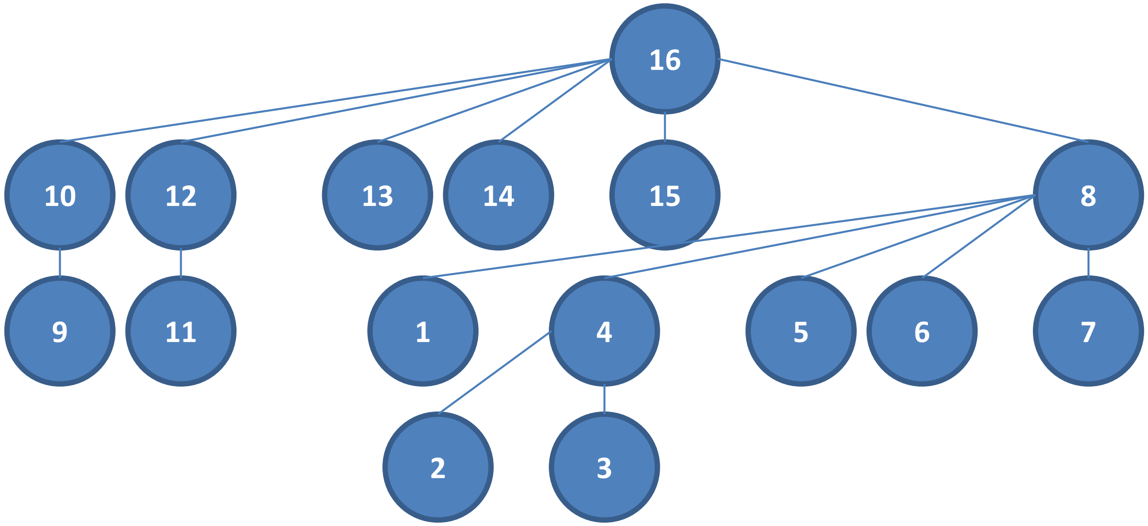
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})



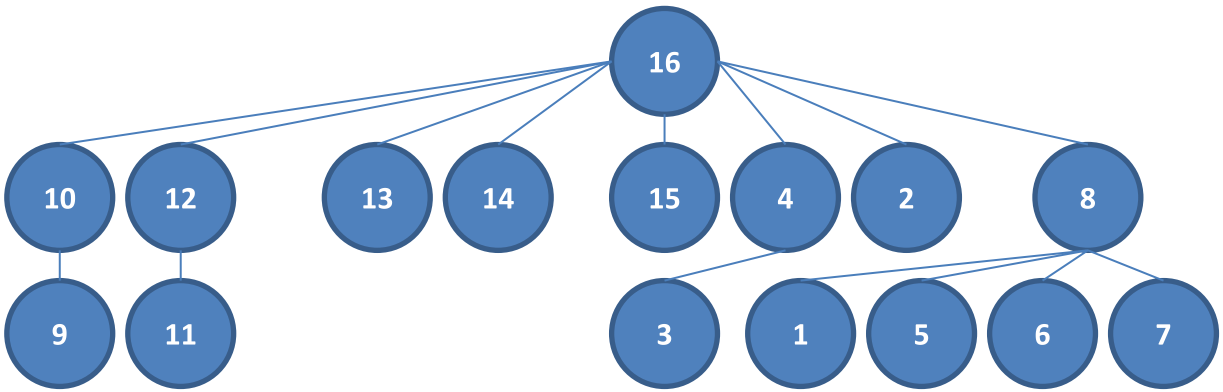
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})
- UNION(x_1, x_{10})



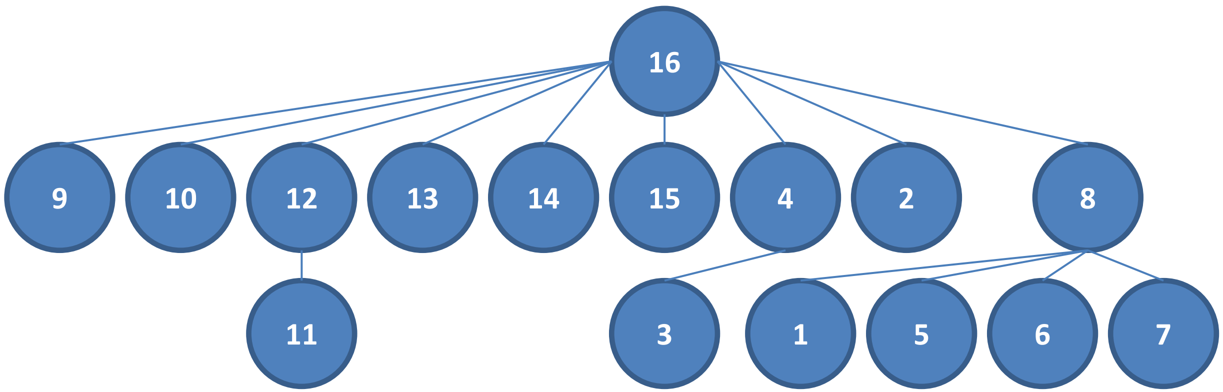
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})
- UNION(x_1, x_{10})
- FIND-SET(x_2)



TC第21.3节练习1 (续)

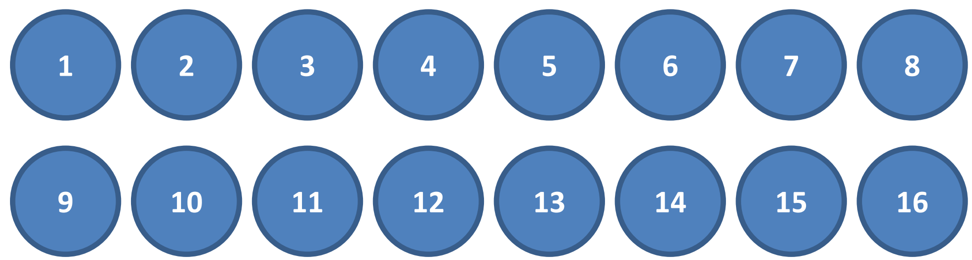
- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})
- UNION(x_1, x_{10})
- FIND-SET(x_2)
- FIND-SET(x_9)



TC第21.3节练习1 (续)

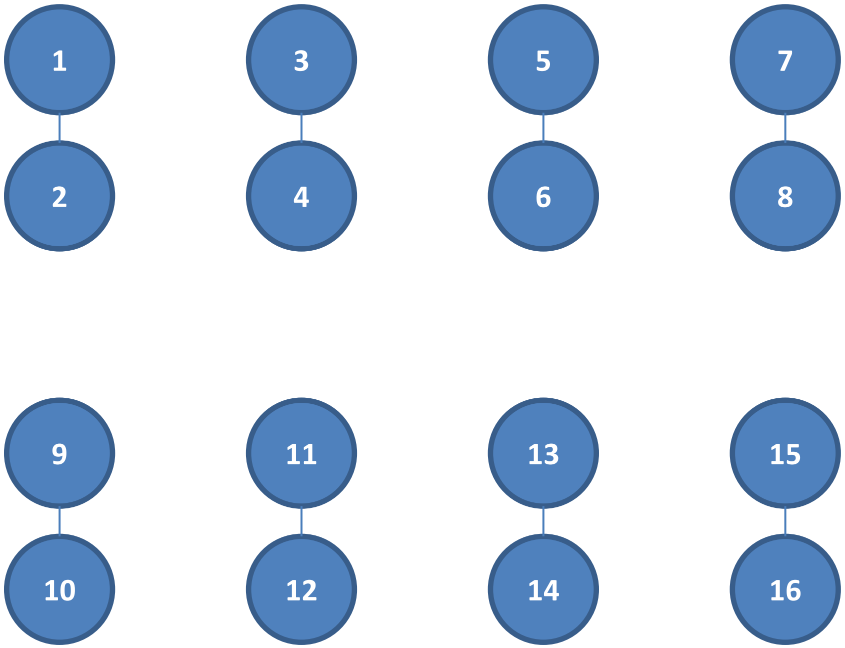
- for i=1 to 16
  - MAKE-SET(x_i)

如果x_i和x_j等长时，把x_j连接到x_i



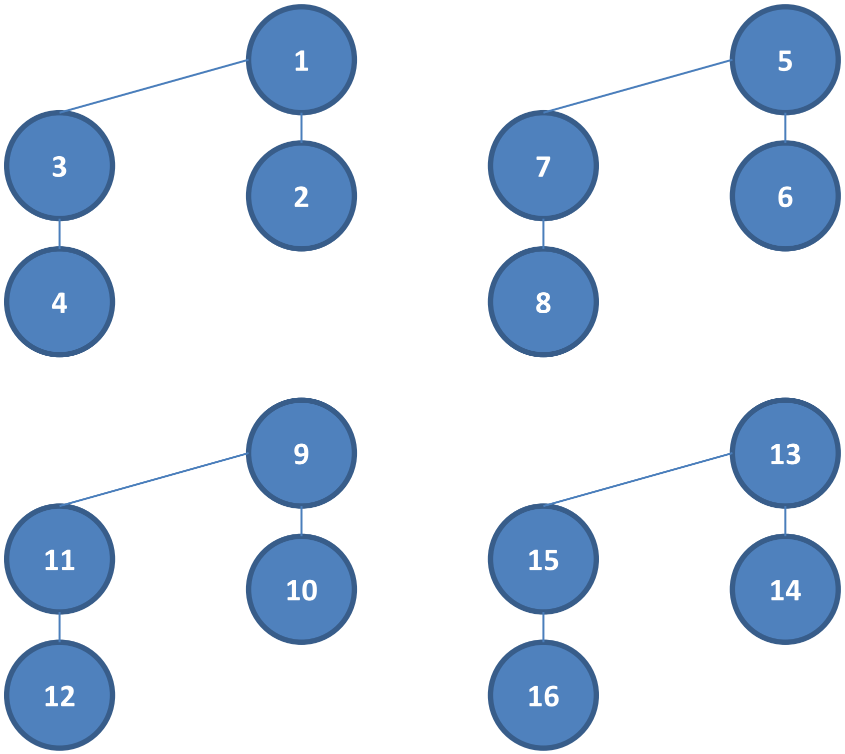
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})



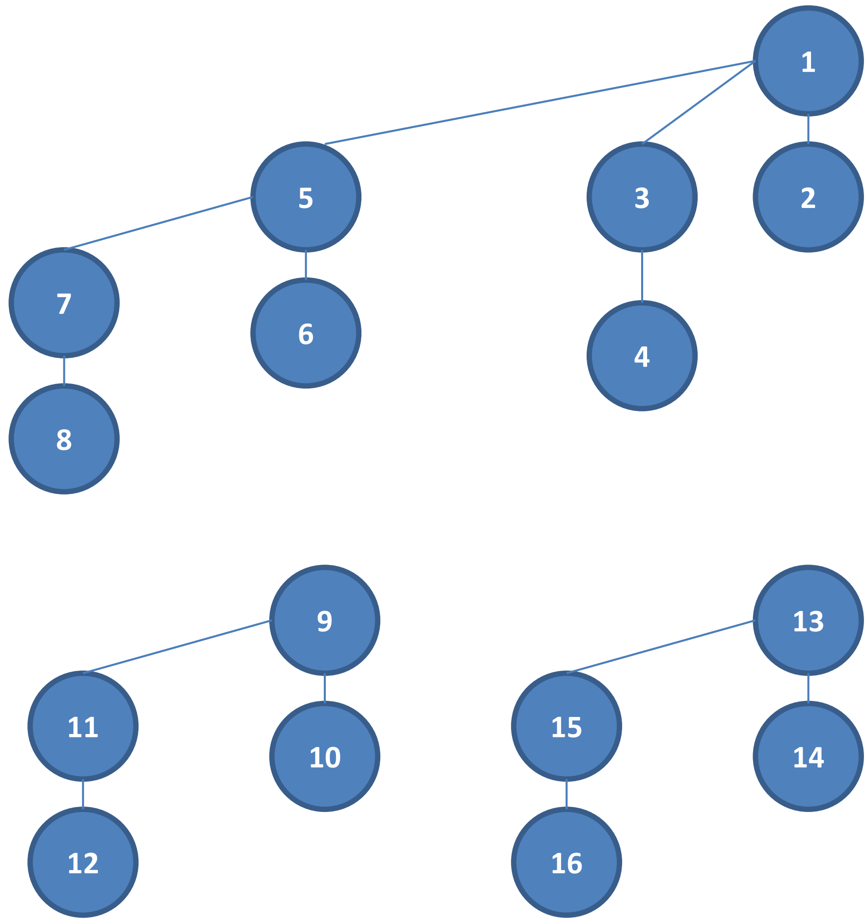
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})



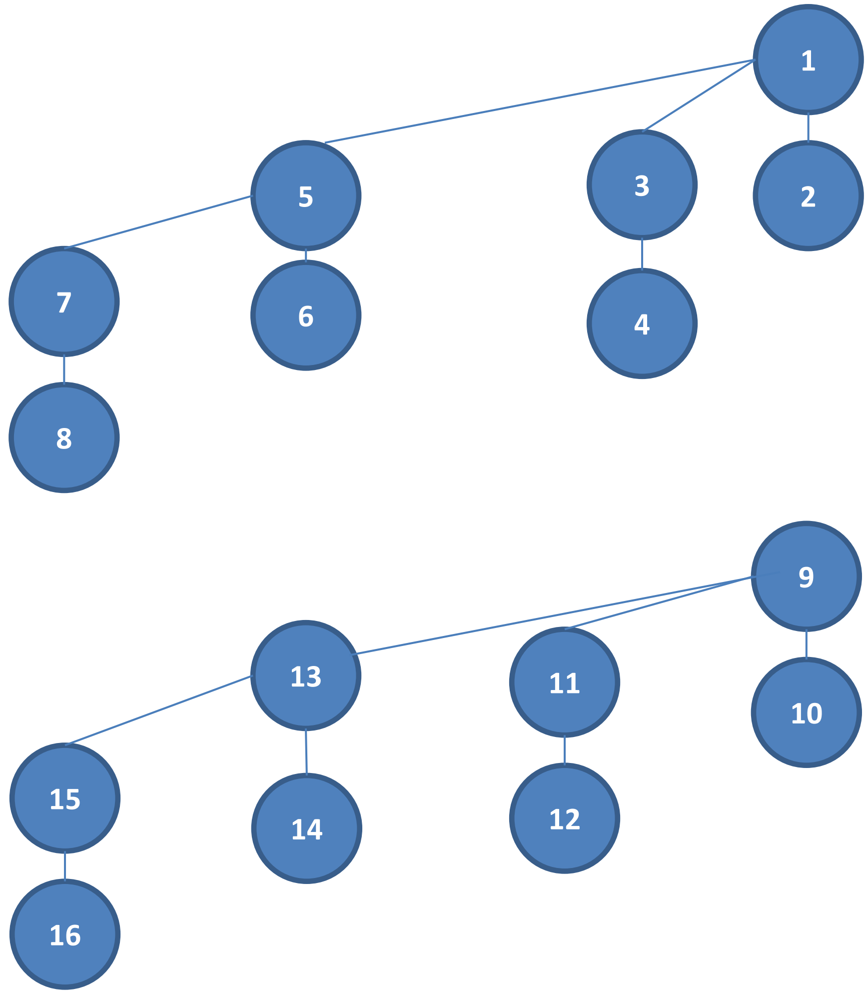
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)



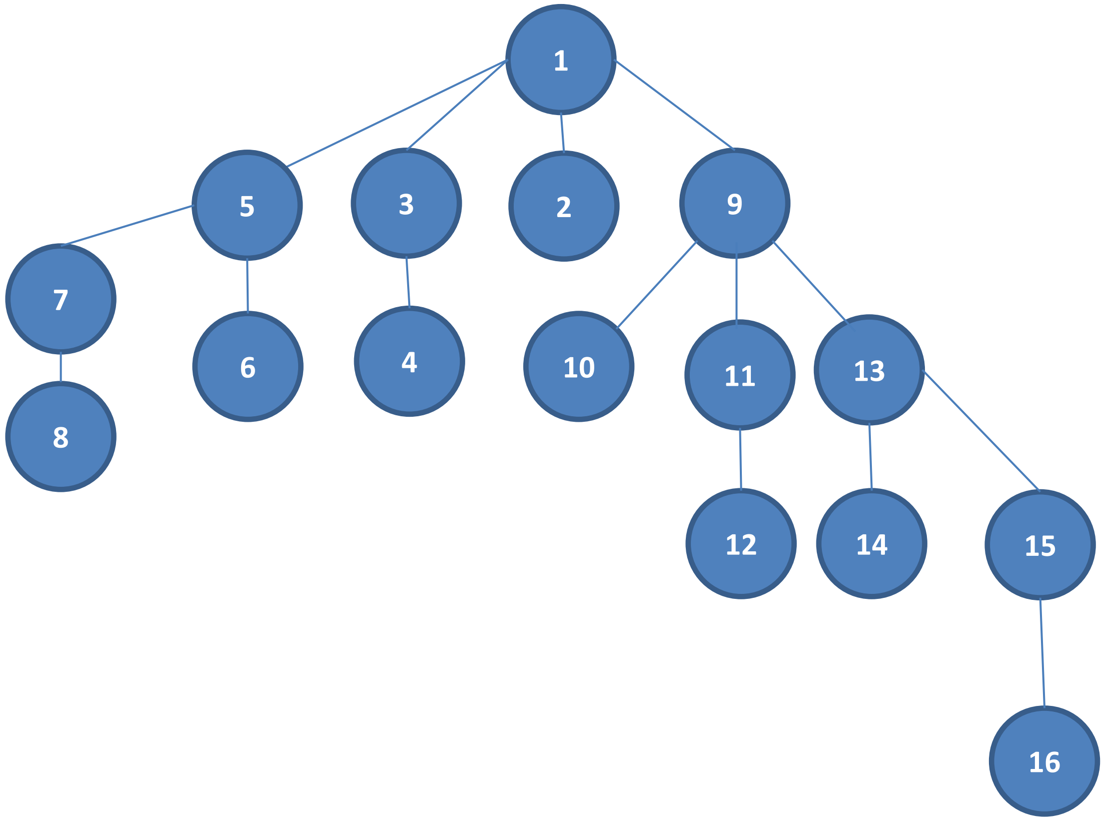
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})



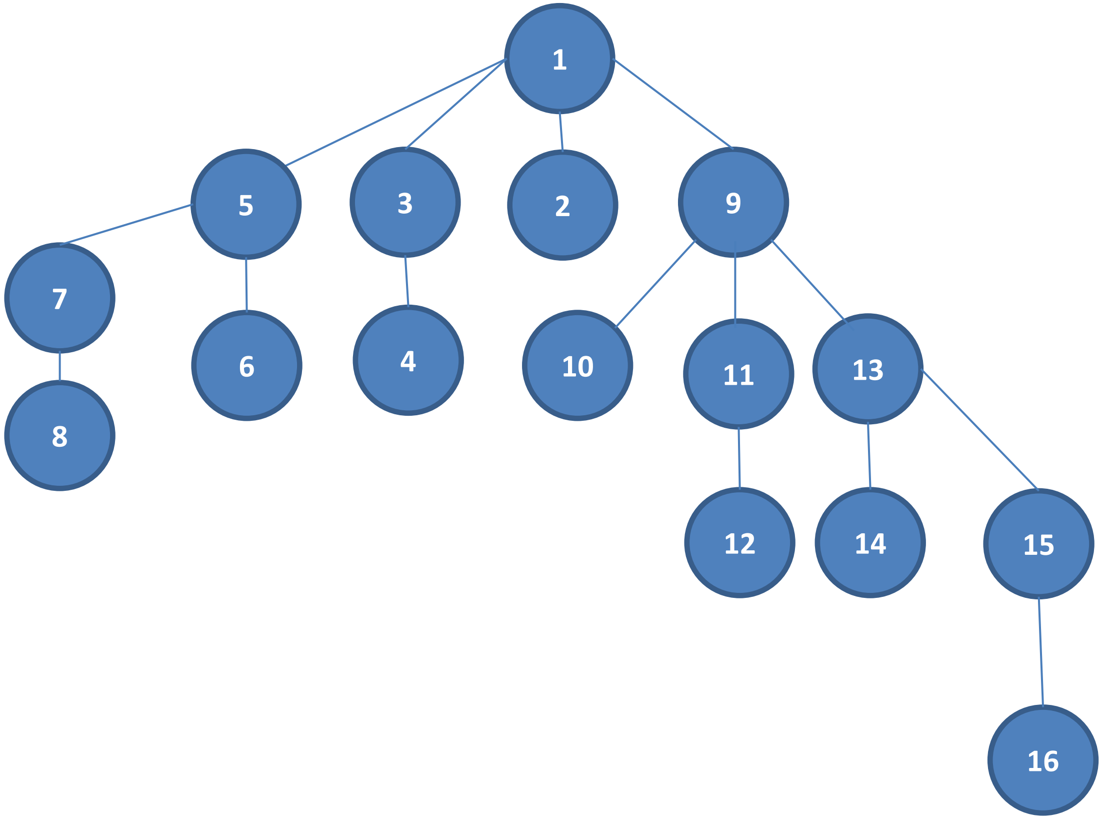
TC第21.3节练习1 (续)

- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})
- UNION(x_1, x_{10})



TC第21.3节练习1 (续)

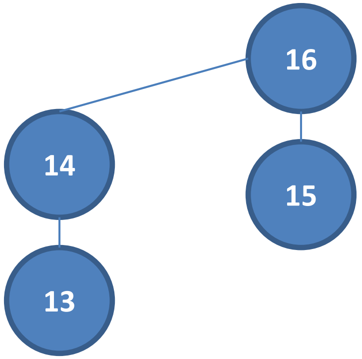
- for i=1 to 16
  - MAKE-SET(x_i)
- for i=1 to 15 by 2
  - UNION(x_i, x_{i+1})
- for i=1 to 13 by 4
  - UNION(x_i, x_{i+2})
- UNION(x_1, x_5)
- UNION(x_{11}, x_{13})
- UNION(x_1, x_{10})
- FIND-SET(x_2)
- FIND-SET(x_9)



TC第21.3节练习2

- FIND-SET(x)

```
y=x;
while (y!=y.p) //先找根
   y=y.p;
whie (x!=x.p) //再压缩路径
   x.p=y;
   x=x.p;
```



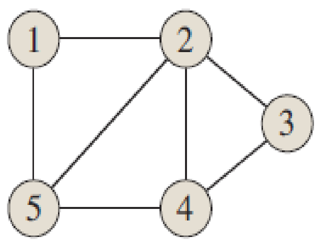
- 这段程序有什么问题？

```
temp=x.p;
x.p=y;
x=temp;
```

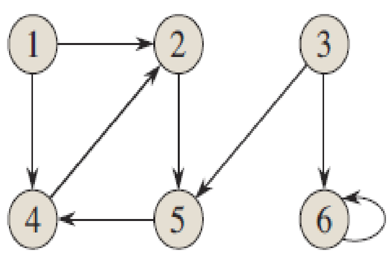
- 教材讨论
  - TC第22章

问题1：图的计算机表示

- 图的计算机表示 = 矩阵的计算机表示



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

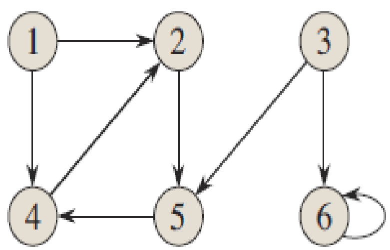
问题1：图的计算机表示 (续)

- 在计算机中存储矩阵的最常用方法：two-dimensional array
  你能从时间和空间的角度，分析这种方法的优缺点吗？

```
[ 10 20  0  0  0  0 ]
[  0 30  0 40  0  0 ]
[  0  0 50 60 70  0 ]
[  0  0  0  0  0 80 ]
```

问题1：图的计算机表示 (续)

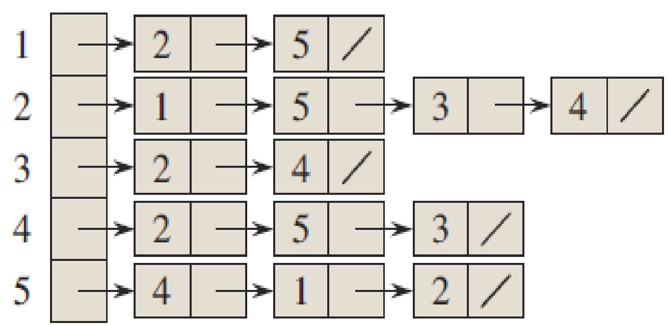
- 实际问题中的图往往是稀疏的，对应的矩阵称作稀疏矩阵
- 稀疏矩阵的存储空间有可能缩小



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

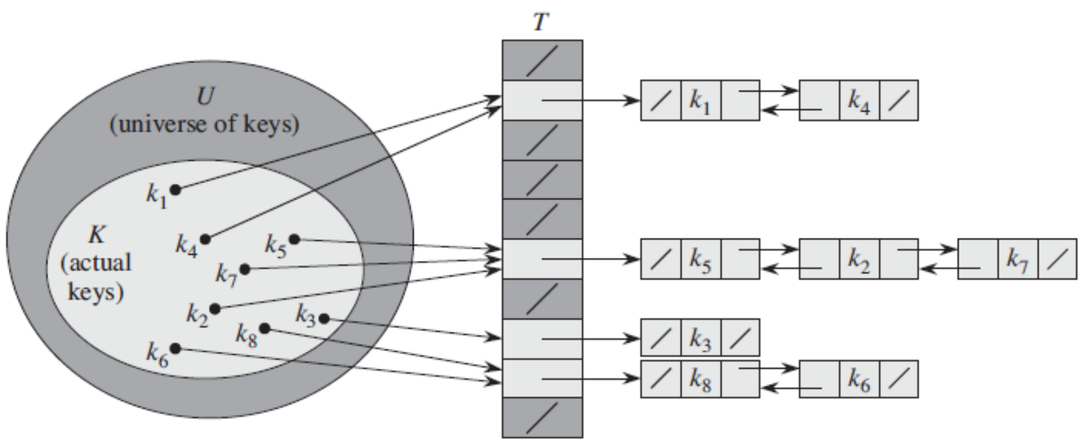
问题1：图的计算机表示 (续)

- List of lists
  - Stores one list per row, where each entry stores a column index and value.
  - Entries are kept sorted by column index.
- 它适合这些应用场景吗？
  - 频繁的元素增删改
  - 随机访问
  - 矩阵运算（加法、乘法）



问题1：图的计算机表示 (续)

- Dictionary of keys
  - Represents non-zero values as a dictionary.
  - Maps (row, column)-tuples to values.
- 它适合这些应用场景吗？
  - 频繁的元素增删改
  - 随机访问
  - 矩阵运算（加法、乘法）



问题1：图的计算机表示 (续)

- Coordinate list
  - Stores a list of (row, column, value) tuples.
  - Entries are sorted (by row index, then column index).
- 它适合这些应用场景吗？
  - 频繁的元素增删改
  - 随机访问
  - 矩阵运算（加法、乘法）

```
{
  {1, 2, 3},
  {1, 4, 1},
  {2, 3, 1},
  {3, 1, 2},
  {3, 4, 1},
  {4, 1, 2}
}
```

问题1：图的计算机表示 (续)

- 总体而言，这些表示方法
  - 较适合频繁的元素增删改
  - 较不适合矩阵运算

问题1：图的计算机表示 (续)

- Yale format
  - A: array of non-zero element values.
  - IA: array of index of first nonzero element of row *i*.
  - JA: array of column index of each A element.
- 它适合这些应用场景吗？
  - 频繁的元素增删改
  - 随机访问
  - 矩阵运算（加法、乘法）

```
[ 10 20  0  0  0  0 ]
[  0 30  0 40  0  0 ]
[  0  0 50 60 70  0 ]
[  0  0  0  0  0 80 ]
```

```
A  = [ 10 20 30 40 50 60 70 80 ]
IA = [ 0  2  4  7 8 ]
JA = [  0  1  1  3  2  3  4  5 ]
```

问题1：图的计算机表示 (续)

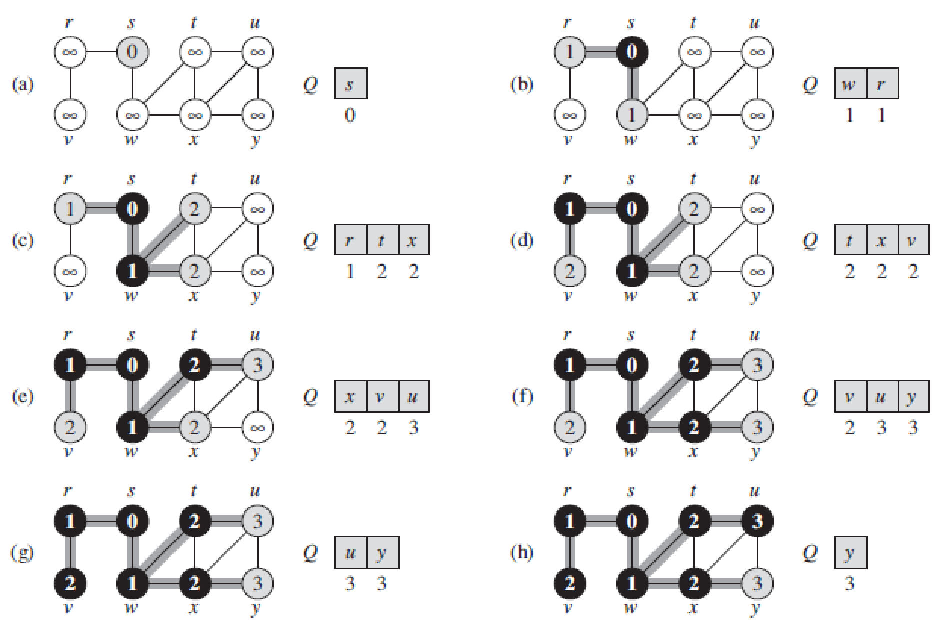
- 怎样才能同时做到
  - 高效的构建矩阵
  - 高效的矩阵运算

问题1：图的计算机表示 (续)

- 怎样才能同时做到
  - 高效的构建矩阵
  - 高效的矩阵运算
- 结合使用
  1. Use dictionary of keys, list of lists, or coordinate list to construct the matrix.
  2. Once the matrix is constructed, it is typically converted to Yale format or similar formats for more efficient matrix operations.

问题2：图的搜索

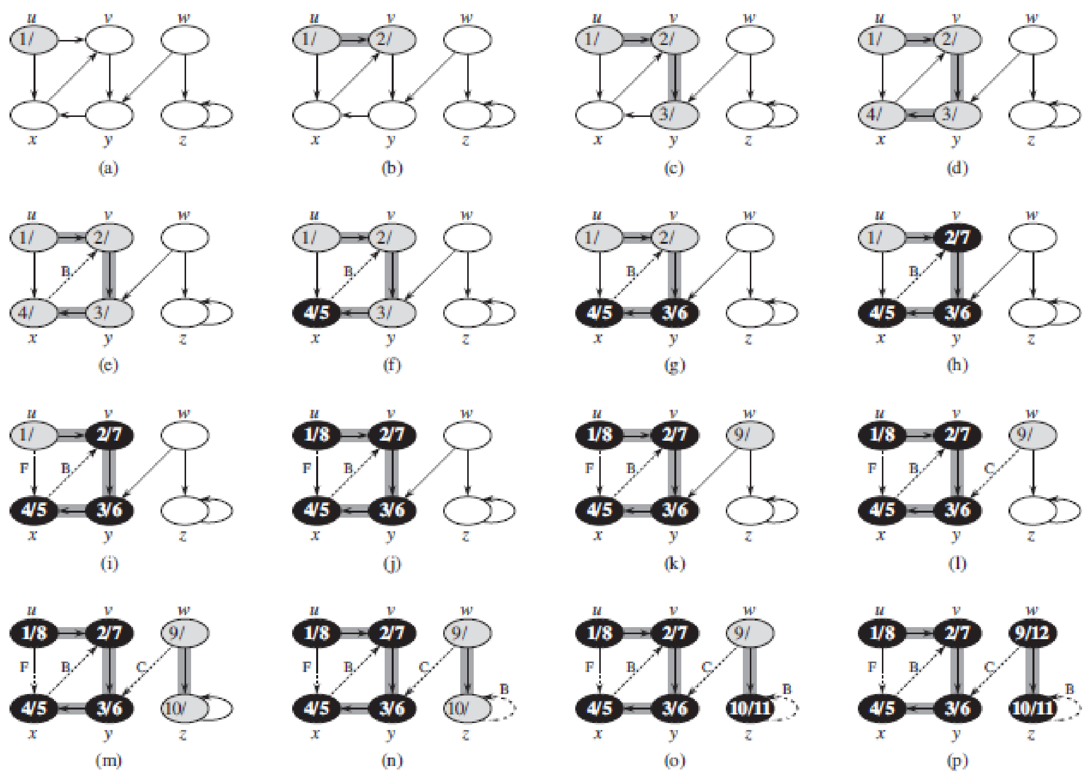
- 广度优先搜索 (BFS)
  - 顶点的三种颜色分别表示什么意思？
  - d、π分别具有什么含义？
  - BFS-tree是如何构建的？为什么一定是tree？
  - 仅就遍历任务而言，BFS有什么缺点？




```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

问题2：图的搜索 (续)

- 深度优先搜索 (DFS)
  - 顶点的三种颜色分别表示什么意思？
  - 为什么DFS需要两种timestamp，而BFS只有一种？



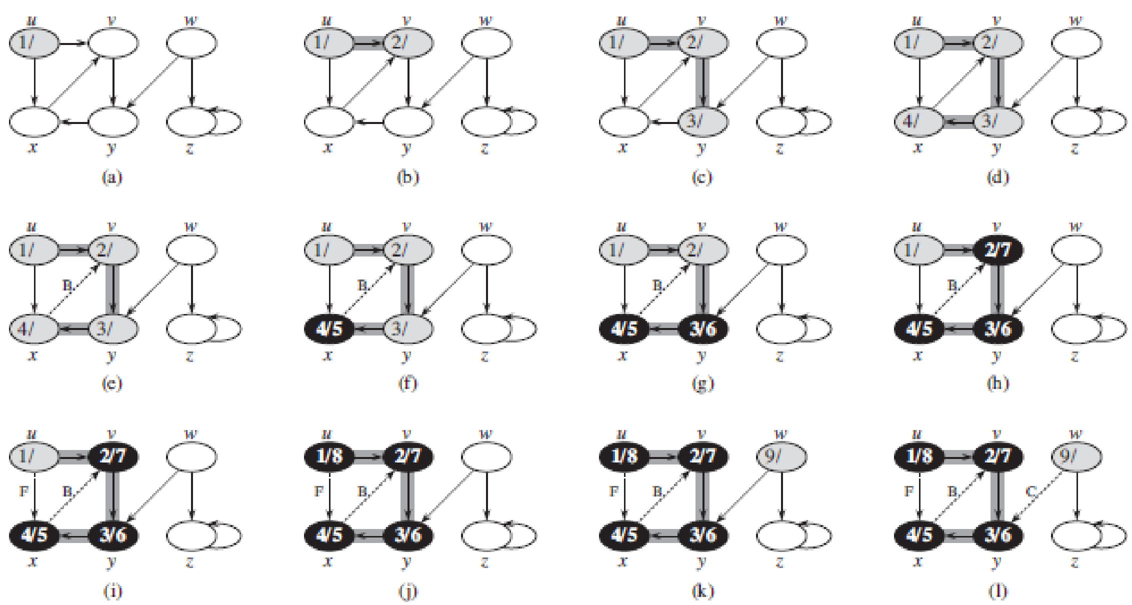
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

问题2：图的搜索 (续)

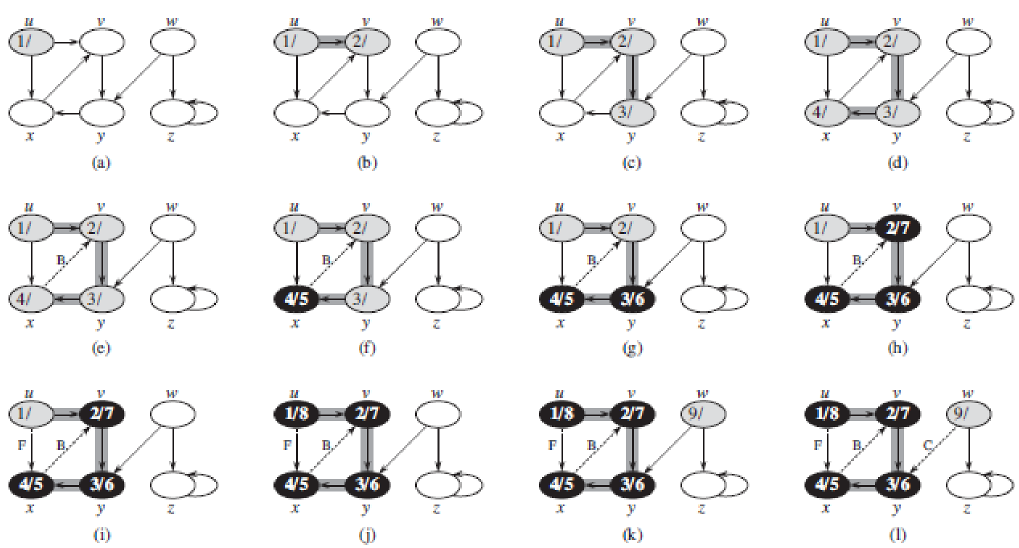
- 深度优先搜索 (DFS)
  - 你能解释后裔关系的三种充要条件吗？

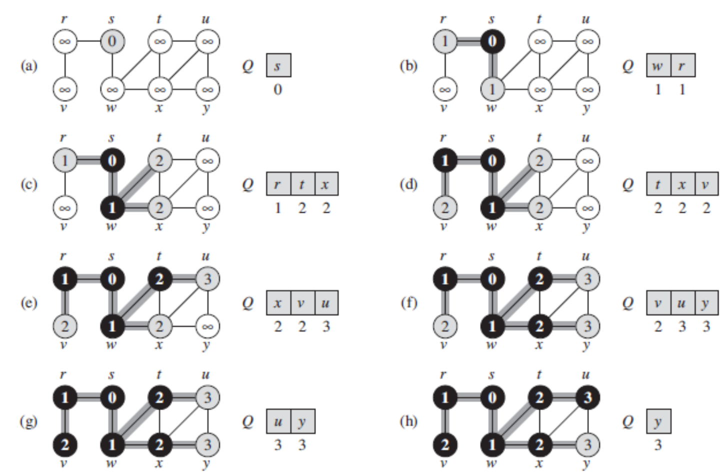
    v is a descendant of u in the depth-first forest if and only if
    - v is discovered during the time in which u is gray.
    - u.d < v.d < v.f < u.f.
    - At the time u.d, there is a path from u to v consisting entirely of white vertices.



问题2：图的搜索 (续)

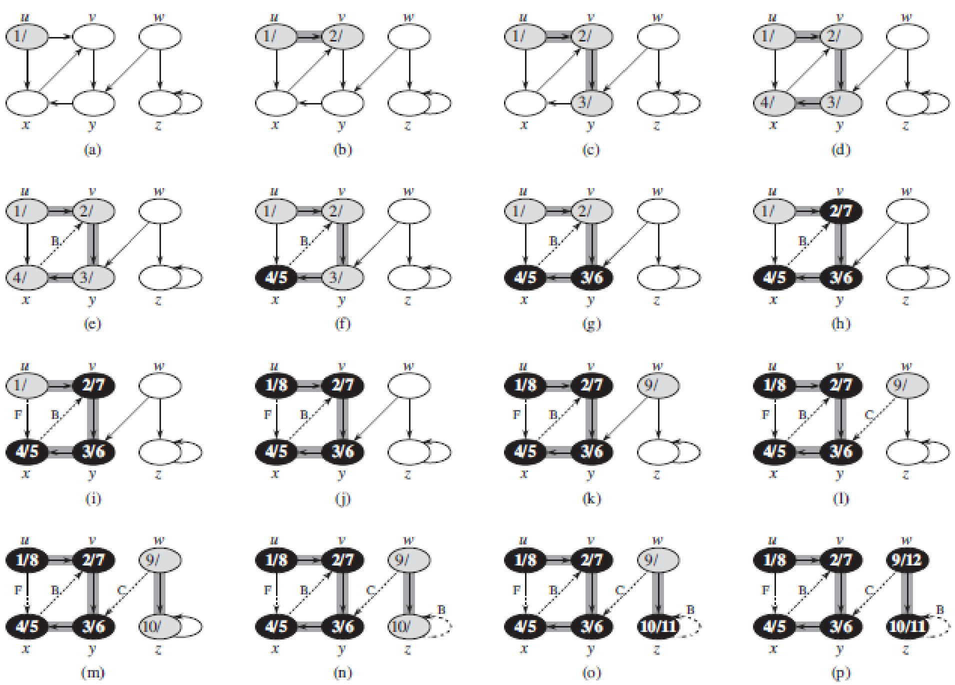
- 深度优先搜索 (DFS)
  - 你理解这四种边了吗？如何识别它们？
    - Tree edges
    - Back edges
    - Forward edges
    - Cross edges
  - 为什么无向图中只有前两种？

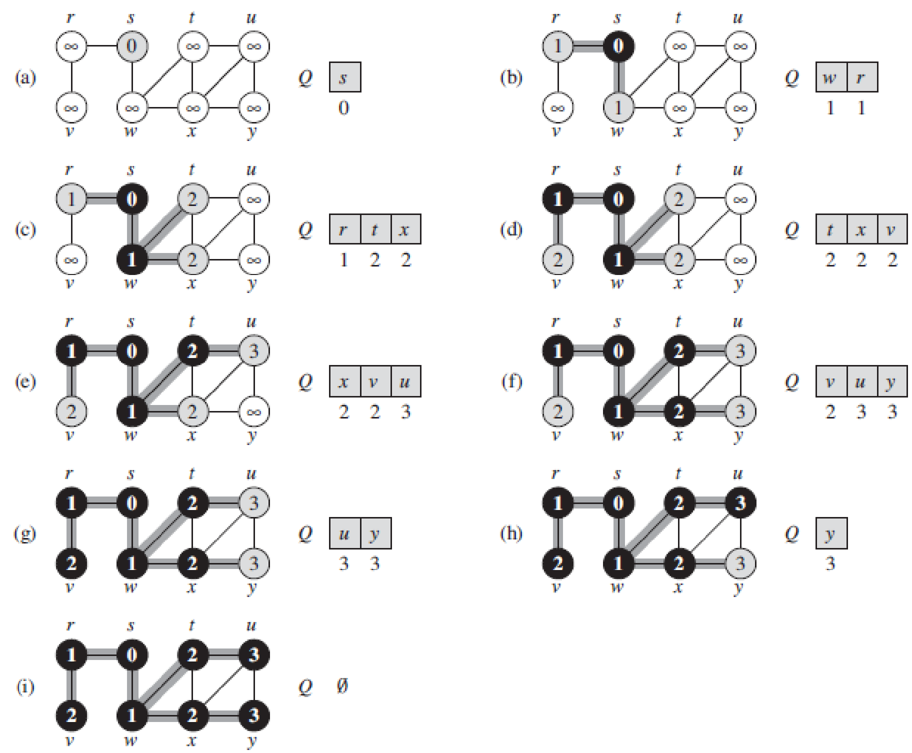




问题2：图的搜索 (续)

- 你能比较BFS和DFS的优缺点吗？

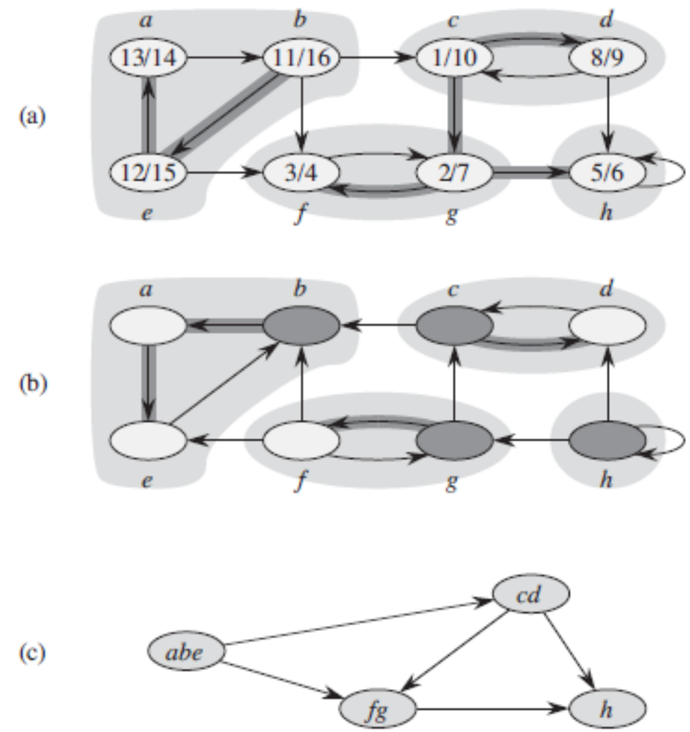




问题2：图的搜索 (续)

- 你理解这个算法的原理了吗？

```
STRONGLY-CONNECTED-COMPONENTS(G)
1   call DFS(G) to compute finishing times u.f for each vertex u
2   compute G^T
3   call DFS(G^T), but in the main loop of DFS, consider the vertices
        in order of decreasing u.f (as computed in line 1)
4   output the vertices of each tree in the depth-first forest formed in line 3 as a
        separate strongly connected component
```



问题2：图的搜索 (续)

There are two types of professional wrestlers: "babyfaces" ("good guys") and "heels" ("bad guys"). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n + r)$-time algorithm that determines whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. If it is possible to perform such a designation, your algorithm should produce it.

问题2：图的搜索 (续)

Let $G = (V, E)$ be a connected, undirected graph. Give an $O(V + E)$-time algorithm to compute a path in G that traverses each edge in E exactly once in each direction.

问题2：图的搜索 (续)

- 你能走出迷宫吗？