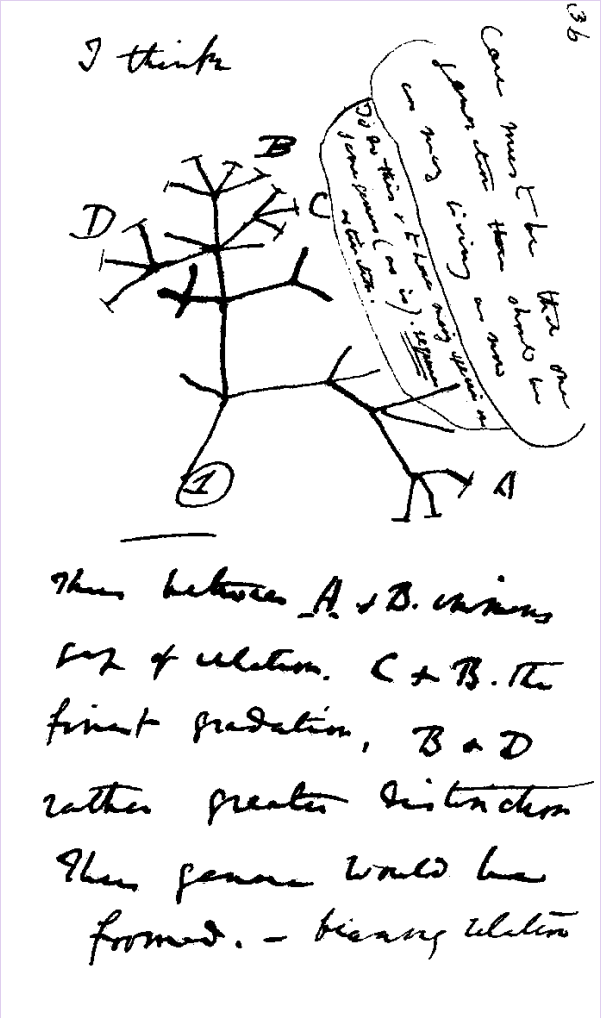
树

School of Computer
Wuhan University

Example — C.Darwin《物种起源》手稿

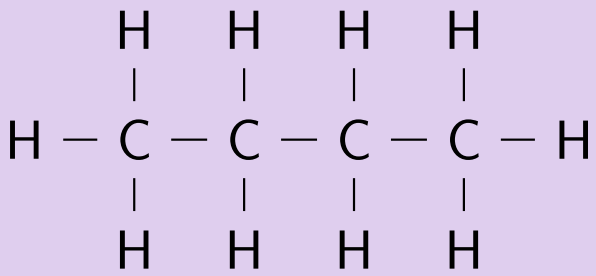
Evolutionary tree (1837)



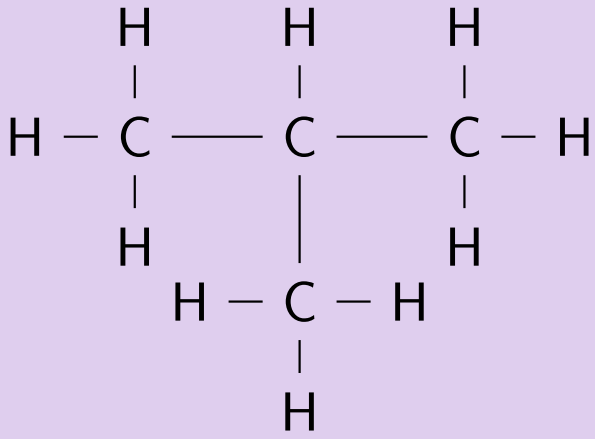
树的起源

树的概念最早是由Arthur Cayley 在研究饱和碳氢化合物 C_nH_{2n+2} 的同分异构体提出的.

饱和碳氢化合物与树



(a) 丁烷



(b) 异丁烷

1. C_4H_{10}在同构意义下有2个;
2. C_5H_{12}在同构意义下有3个.

无向树的定义

Definition

- 连通并无简单回路的无向图称为无向树 (Undirected tree)，简称为树；
- 设T为树，$\deg(v) = 1$，称v为树叶(Leaf)；否则称为树枝(Branch)；
- 如果无向图的每个连通分支都是树，称这样的图为森林(Forest).

Example

- 以太网(Ethernet)：每对结点是可达的，但是不能有回路；
- 广域网：允许有回路，不是树.

树的等价定义

Theorem

1. 连通且无简单回路(Definition);
2. 无简单回路且$m = n - 1$;
3. 连通且$m = n - 1$;
4. 无简单回路，增加一条边则有简单回路;
5. 连通，删除一条边则不连通，即每条边都是割边;
6. 每对结点有唯一的一条基本路径.

Remark

树是连通与非连通的临界状态!

定理的证明(1/3)

Proof.

① $\Longrightarrow$ ②: 归纳法

1. $n = 1$时, $n = 0$, ②成立;
2. 设$n = k$时, ②成立;
3. 考虑树$T'(k+1, m')$, 则T'一定有一个叶结点, 否则每个结点的度数均$\geqslant 2$时, 一定存在一个简单回路, 这与T'是树矛盾;
4. 设$\deg(v) = 1$, 则$T' - \{v\}$还是树, 其结点数为k, 边数为$m-1$, 由归纳假设得 $m - 1 = k - 1$,

无简单回路且$m = n - 1 \Longrightarrow$ 连通且 $m = n - 1$

② $\Longrightarrow$ ③: 反证法

1. 设T不连通, 并且有两个连通分支T_1和T_2;
2. 则$T_1(n_1, m_1)$和$T_2(n_2, m_2)$都是树:
$$m = m_1 + m_2 = n_1 - 1 + n_2 - 1 = n - 2 < n - 1$$
与条件矛盾.

□

定理的证明(2/3)

连通且$m = n - 1 \Longrightarrow$无简单回路，增加一条边则有简单回路

Proof.

③ $\Longrightarrow$ ④：归纳法

1. $n = 1$时，$m = 0$，④成立；
2. 设$n = k$时，图$T(k, k-1)$无简单回路且加上一边后简单回路.
3. 考虑图$T'(k+1, k)$，则T'一定有结点v_j，$\deg(v_j) = 1$，否则每个结点的度数均$\geqslant 2$时:

$$2k = \sum_{i=1}^{k+1} \deg(v_i) > 2k$$

矛盾；

4. 这样$T' - \{v_j\}$也满足条件③，所以，在T上增加一条边，如果该边的两个端点都不是v_j，则等价于在$T' - \{v_j\}$上增加了一条边，由归纳假设将形成简单回路；
5. 如果该边为$v_i v_j$，则由于T'是连通图，所以存在v_i到v_j的路径P，这样$P \cup (v_j, v_i)$是简单回路.

□

定理的证明(3/3)

Proof.

④ $\Longrightarrow$ ⑤: 反证法

1. 设T不连通，并且有两个连通分支T_1和T_2；则在T_1中选择一结点u，在T_2中选择一结点v，增加边uv即是割边. 根据割边的等价定理，uv 一定不再任何简单回路上. 矛盾.
2. 同样，删除一条边后则不连通，否则原图一定有简单回路.

⑤ $\Longrightarrow$ ⑥: 反证法

1. 否则若有简单回路，删除该回

每对结点有唯一的一条路径 $\Longrightarrow$ 连通且无简单回路

⑥ $\Longrightarrow$ ①: 反证法

1. 连通性：因为每对结点都有一条路径；
2. 无简单回路：否则将存在一对结点有两条不同路径.

□

生成树

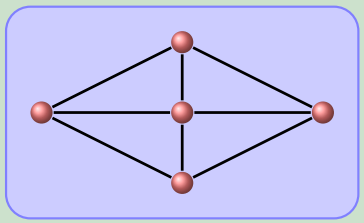
Definition

设G是无向连通图，如果G的一个生成子图是无向树，则称该子图为图G的一颗生成树(Spanning tree).

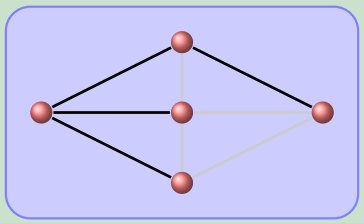
Theorem

任意一个连通图至少有一颗生成树.

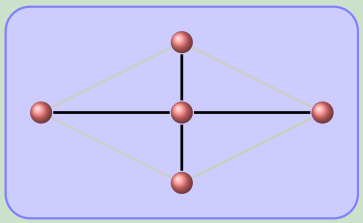
Example



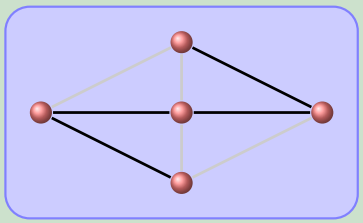
(a) 原图



(b) 生成树1



(c) 生成树2



(d) Hamilton路径

图的遍历

遍历算法

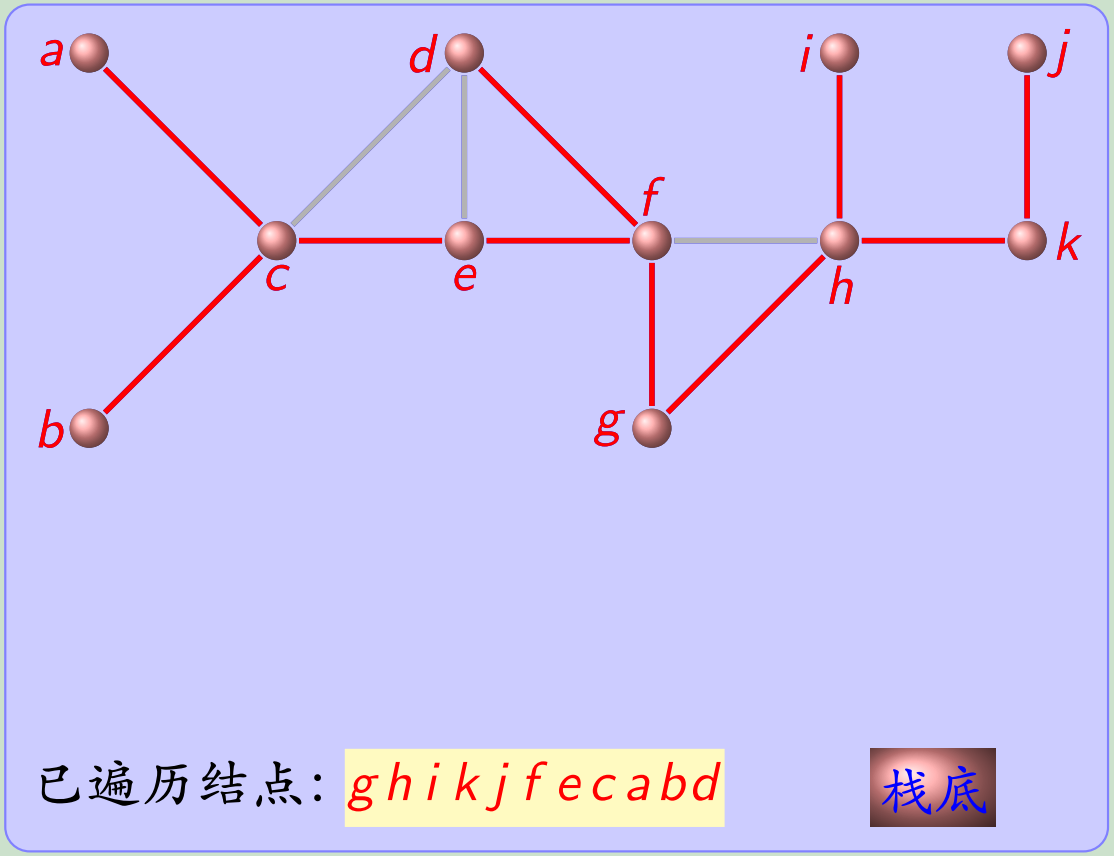
- Depth-first search(DFS): 选择一个结点作为始点，沿以该结点作为始点某一条路径作为遍历的方向，直到该路径上的结点都被访问后才进行回溯(Backtracking)；重复上述过程直到图中的每个结点都被访问过为止；
- Breadth-first search(BFS): 选择一个结点作为始点，依次访问与该结点相邻的每个结点，再对这些相邻的结点按上述方式遍历直到所有的结点都被访问过为止；
- Dead loop(死循环问题): 由于图中有回路，必须对已经访问过的结点标记，在遍历过程中对已有访问标记的结点不再访问；
- Backtracking: 在非递归实现中，必须记录下还没有展开的结点，如: Stack(FILO) for DFS, Queue(FIFO) for BFS;
- 对非连通图，必须对每个连通分支用上述算法才能遍历所有的结点.

DFS Algorithm

```
1   function DFS(Start, Goal)
2   {
3     push(Stack, Start);
4     while Stack is not empty do {
5       var Node := Pop(Stack);
6       if Visited(Node) then
7         continue;    //C style continue
8       if Node = Goal then
9         return Node;
10      setVisited(Node);
11      for Child in Expand(Node) do
12        if notVisited(Node) then
13          push (Stack, Child);
14      }
15  }
```

DFS与生成树的生成

DFS产生生成树的动态演示



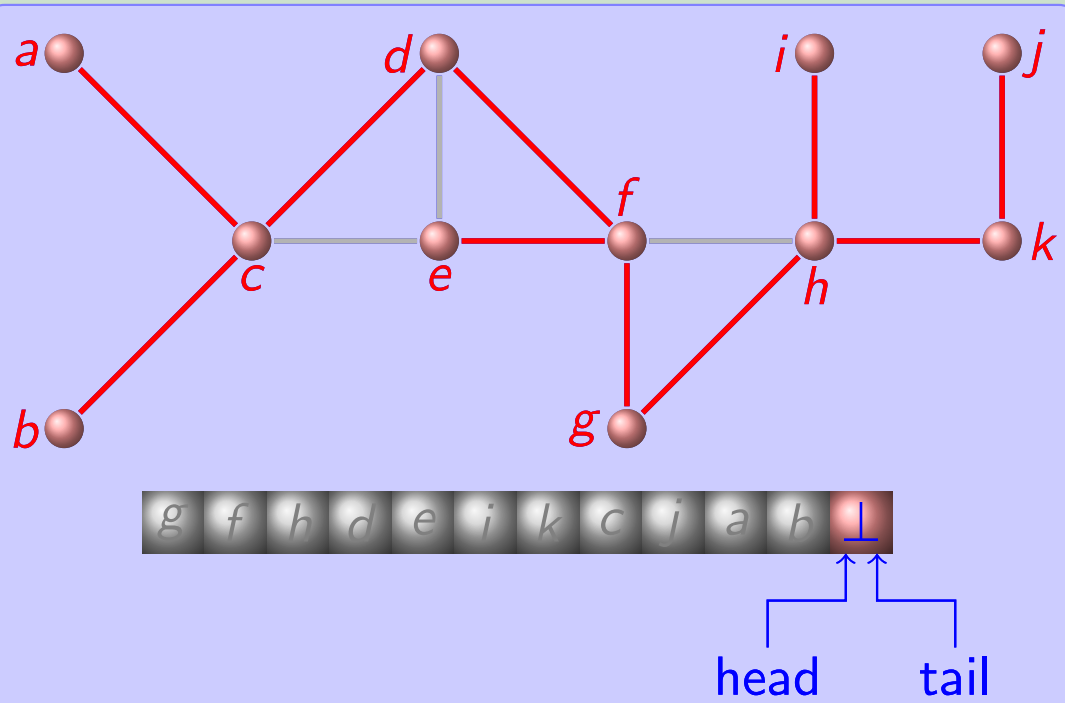
(a)

BFS Algorithm

```
1   function BFS(Start, Goal)
2   {
3     setVisited(Start);
4     enqueue(Queue,Start);
5     while notEmpty(Queue) {
6       Node := dequeue(Queue);
7       if Node = Goal then
8         return Node;
9       for each Child in Expand(Node) {
10        if notVisited(Child) then {
11          setVisited(Child);
12          enqueue(Queue, Child);
13          }
14      }
15    }
16  }
```

BFS与生成树的生成

BFS产生生成树的动态演示



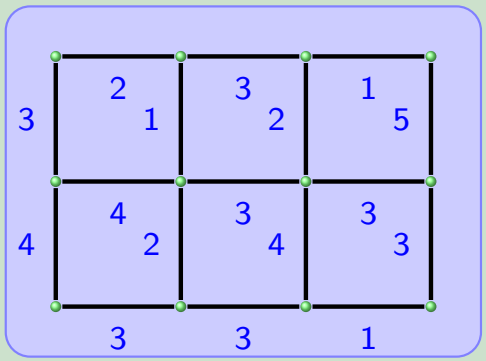
已遍历结点: g h f e d i k c j a b

最小生成树

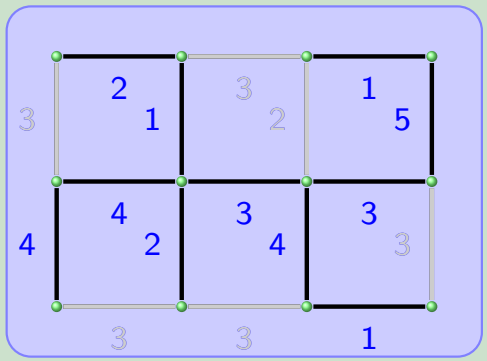
Definition

- 设$G = \langle V, E, \omega \rangle$是简单连通赋权图，其中：$\omega : E \longrightarrow \mathbb{R}^+$，设$T$为$G$的生成树，$T$所有的边的权值之和称为$T$的权值；
- 如果T在G的所有生成树中其权值最小，称为最小生成树(Minimum Spanning Tree, MST)；
- 最小生成树一定存在. 应用于管网设计的优化.

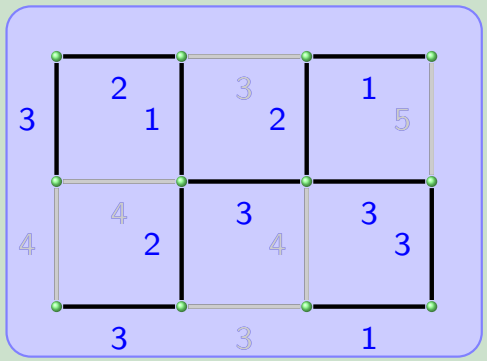
Example



(a) 原图



(b) 权值为30的生成树



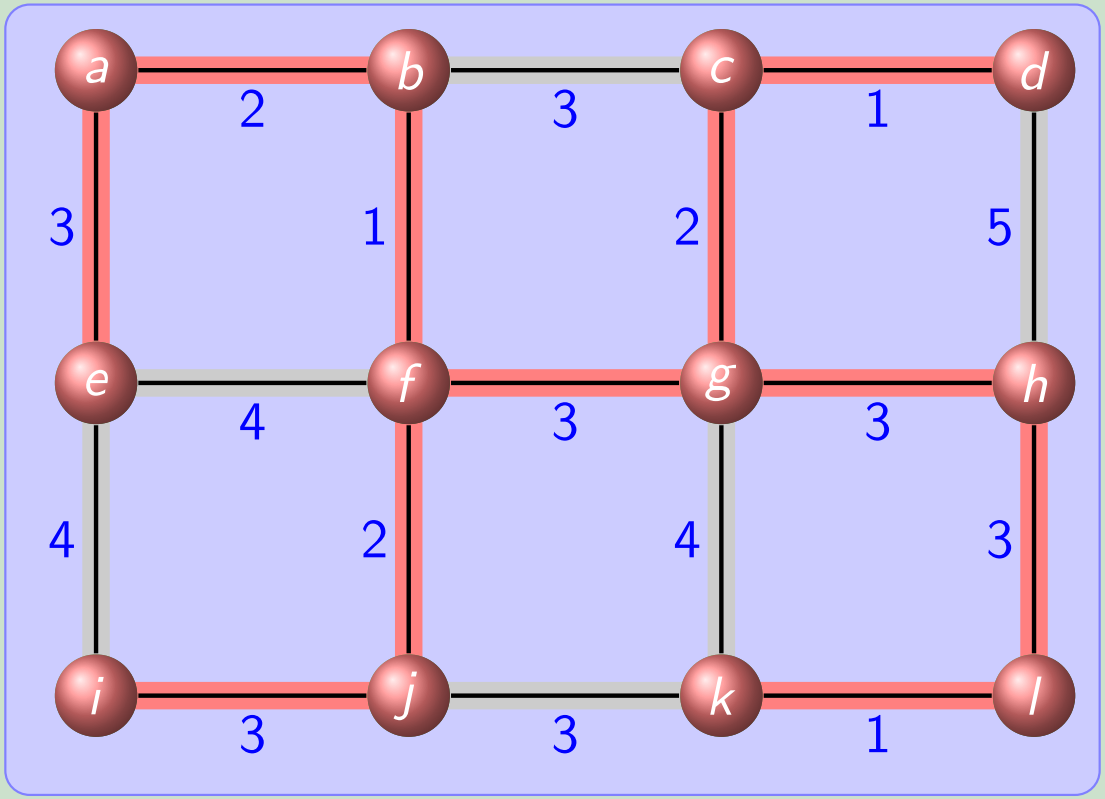
(c) MST(权值为24)

Prim's Algorithm

Prim's Algorithm — 求最小生成树

1. 初始化: 设$G = \langle V, E, \omega \rangle$是无向连通赋权图;
   $T :=$由E中的最小权值边导出的子图;
2. for $i := 1$ to $n - 2$ do {
   $e :=$ 和T相邻并$T \cup \{e\}$不产生简单回路的最小权值边;
   $T := T + \{e\}$;
   }
3. 当上述循环终止所得到的图即是MST.

Prim's Algorithm动态演示

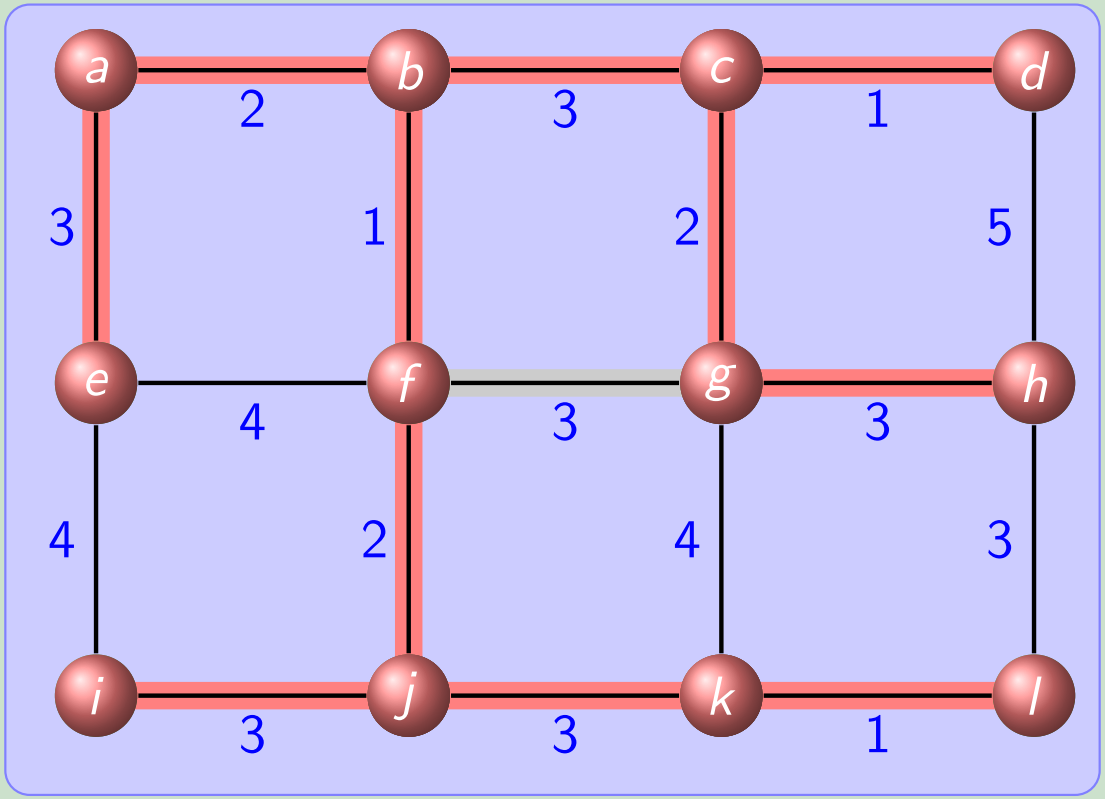


Kruskal's Algorithm

Kruskal's Algorithm — 求最小生成树

1. 初始化：设$G = \langle V, E, \omega \rangle$是无向连通赋权图；
   $T := \varnothing$；
   $E := \{e_1, e_2, \ldots, e_m\}(\omega(e_1) \leqslant \omega(e_2) \leqslant \cdots \leqslant \omega(e_m))$；
   $i := 1$；
2. if （$T \cup \{e_i\}$ 不成简单回路）
   $T := T \cup \{e_i\}$；
3. if （T 中有$n-1$条边）then Stop;
   else $\{i := i + 1$; goto ②. $\}$

Kruskal's Algorithm动态演示



Kruskal's Algorithm的正确性

问题

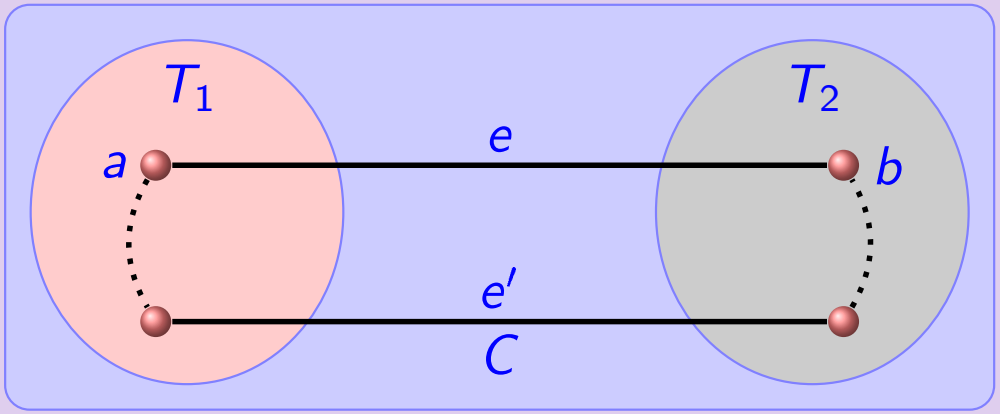
1. 如果算法能在$n-1$步终止，此时的生成树是否MST?
2. 算法不能在$n-1$步之前加上剩下任意的权值比最后一步添加的边要大的边都成简单回路?

Kruskal's Algorithm的正确性

Lemma

设T是G的MST，C是G上的任意一个简单回路，则该回路的最大权值边e一定不在T中.

Sketch of Proof(反证法)



设e在T中，则$T-\{e\}$将T分解为两个连通分支T_1和T_2，如上图所示；由于a和b可以通过回路C的另一段路径相连接，所以存在$e' \in C$, $e' \neq e$, e'是T_1和T_2的割边，这样$(T-\{e\}) \cup \{e'\}$也是一颗生成树，并且其权值比T要小($\because \omega(e') \leqslant \omega(e)$)，这与$T$是MST矛盾.

Kruskal's Algorithm的正确性

Answer for Problem

1. 如果算法能在$n-1$步终止，此时的生成树是否MST?
   反证法：设T是Krustal's Algorithm output的树，T'是MST
   $T \neq T'$，则存在$e \in T' \wedge e \notin T$，而$T \cup \{e\}$一定有简单回路$C$，并且该回路经过$e$，这样$e$一定是回路中的最大权值边，否则算法不会将$e$排除，而根据引理回路上的最大权值边$e$一定不在MST上，这与$e$是MST上的边矛盾.

Kruskal's Algorithm的正确性

Answer for Problem

2. 算法在$n-1$步之前加上剩下任意的权值比最后一步添加的边要大的边都成回路?
设算法在$n-1$步时加上任何权值比$n-2$步所加边要大的边时都成回路，此时因为T中仅有$n-2$条边，所以T做为G的生成子图一定不连通，设$T = T_1 \cup T_2$, $T_1 \cap T_2 = \varnothing$, 所以存在$e \in E$, $T \cup \{e\}$没有回路，由于$\omega(e)$比算法在$n-2$步所选的边的权值要小(因为T加上比该边权值要大的边均成回路)，所以算法不能排除选择边e, 这与边e没有被选矛盾.

有向树的定义

Definition

设$T = \langle V, E \rangle$是有向图，称之为有向树(根树，Rooted tree)，iff，其满足下述三条件：

1. 存在一个结点其入度为0，称为树根(Root)；
2. 其它所有结点的入度均为1；
3. 从树根到其他每个结点都有一条基本路径.

Example

1. 机构的组织结构；
2. Computer File System；
3. 面向对象程序设计的对象继承关系；
4. XML和HTML语言的结构.

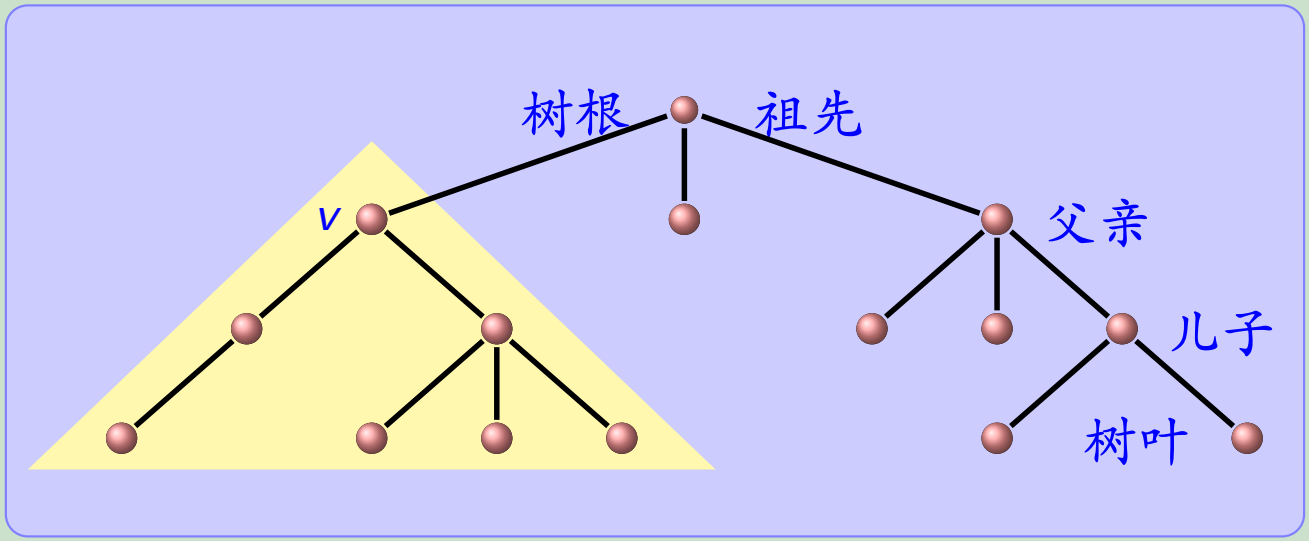
相关概念

Definition

设$T = \langle V, E \rangle$是有向树:

1. 父亲(Parent)，儿子(child)，祖先(Ancestor)，后裔(Descendant)，树叶(Leaf)；
2. 子树(Subtree)：由v及其后裔导出的子图；
3. 深度(Height)：由树根到树叶的最长基本路径的长度.

Example



相关性质

Theorem

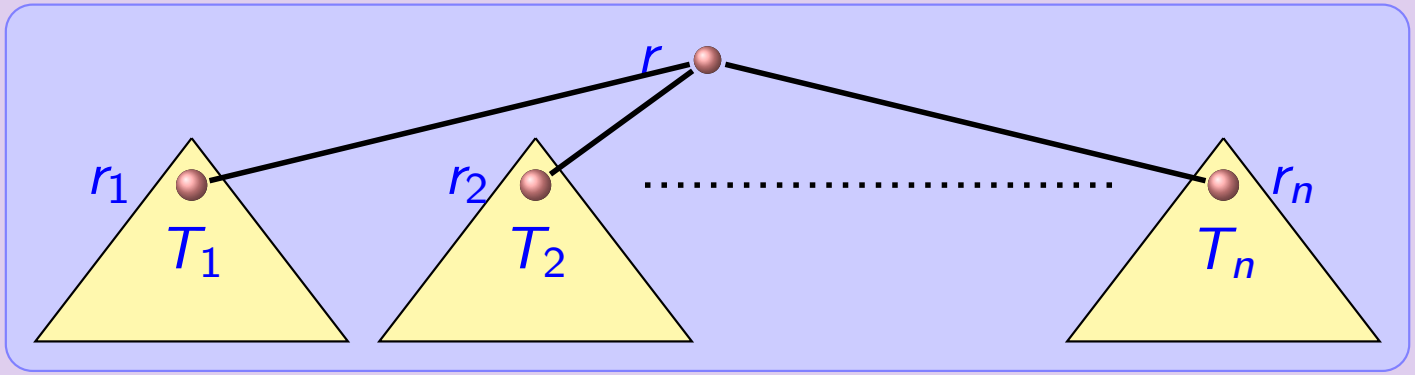
设$T = \langle V, E \rangle$是有向树，则：

1. 从树根到树中的每个结点有唯一的一条路径；
2. 有向树没有回路；
3. $\sum_{i=1}^{n} \deg^+(v_i) = \sum_{i=1}^{n} \deg^-(v_i) = m = n - 1$；
4. 有向树的底图是无向树；
5. 有向树的子树也是树.

有向树的递归定义

Definition

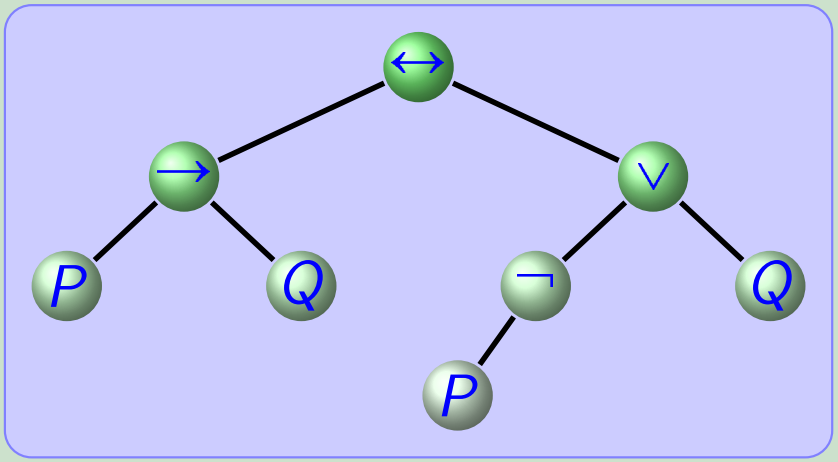
1. 归纳基础：孤立点是有向树；
2. 设$T_1(r_1), T_2(r_2), \ldots T_n(r_n)$是有向树，$r$是孤立点，则下图也是有向树：



3. 按上述规则在有限步生成的都是有向树.

递归定义与树结构

Example (命题公式和程序语言表达式的递归定义)



$$(P \to Q) \leftrightarrow (\neg P \vee Q)$$

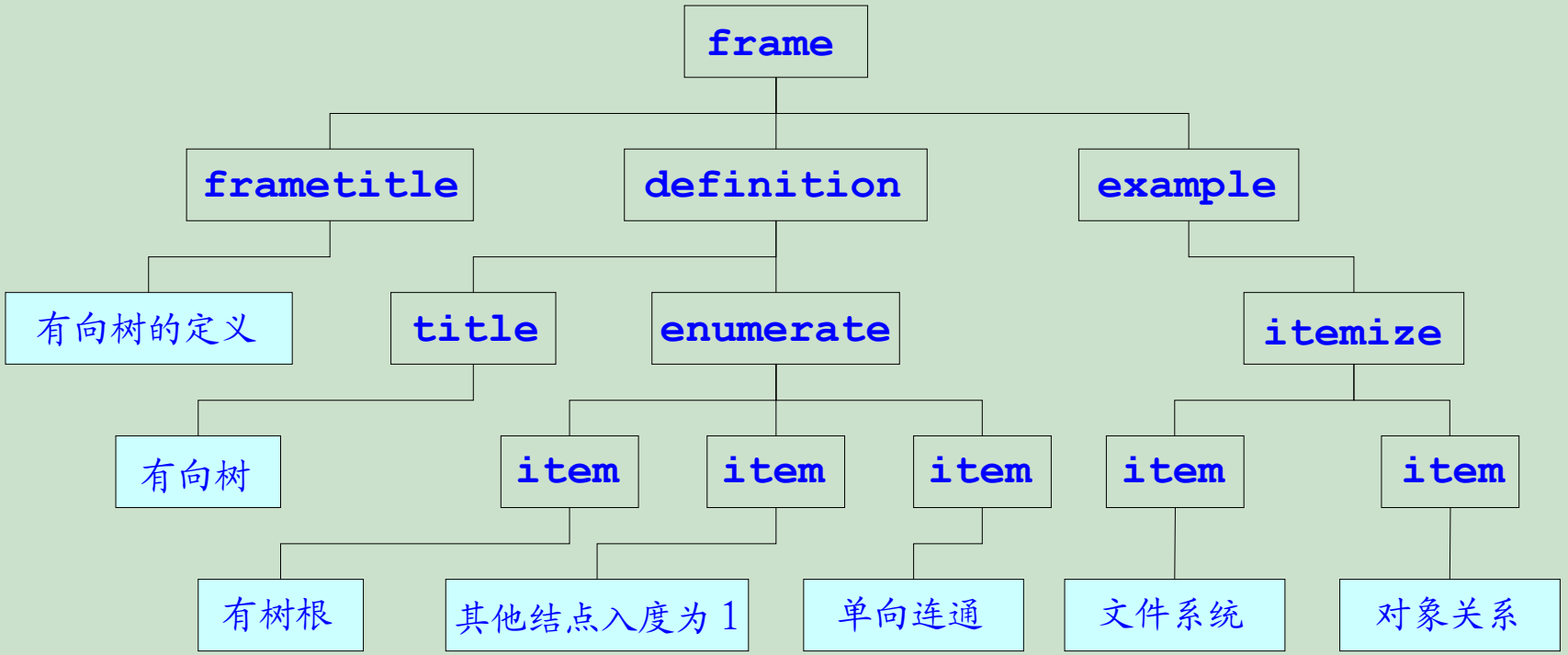
XML语言的树结构(1/2)

XML语言

```
<frame>
  <frametitle> 有向树的定义  </frametitle>
  <definition>
    <title> 有向树 </title>
    <enumerate>
      <item> 有树根 </item>
      <item> 其他结点入度为1 </item>
      <item> 单向连通 </item>
    </enumerate>
  </definition>
  <example>
    <itemize>
      <item> 文件系统 </item>
      <item> 对象关系 </item>
    </itemize>
  </example>
</frame>
```

XML语言的树结构(2/2)

Example (XML语言的树结构)



特殊的树

Definition

设$T = \langle V, E \rangle$是有向树:

1. k元树: 每个结点的儿子之数$\leqslant k$; 当$k = 2$时称为二元树, 也称二叉树(Binary Tree);
2. 每个树枝均有k个儿子的树称为完全k元树(Complete k-ary tree);
3. 每个树叶的深度相同的完全k元树称为完美k元树(Perfect k-ary tree);
4. 树枝的每个儿子的排列次序有意义的树称为有序树(Ordered tree);
5. 树枝的每个儿子的位置有意义的树称为位置树.

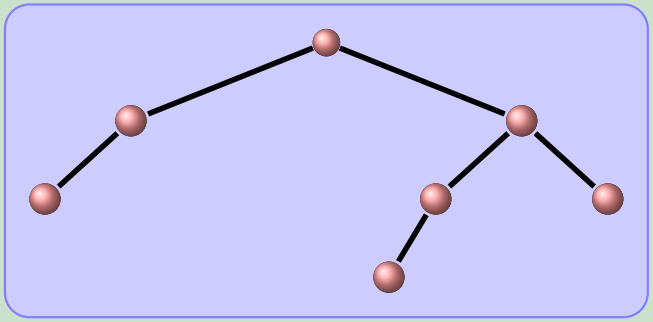
Proposition

设T是k元树, n是其结点数, h是深度, 则:

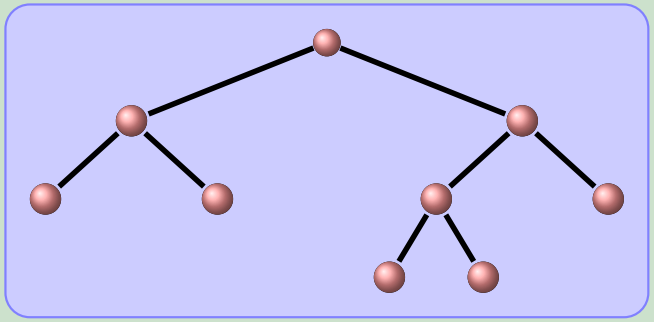
$$h + 1 \leqslant n \leqslant \frac{k^{h+1} - 1}{k - 1}$$

Example (1/2)

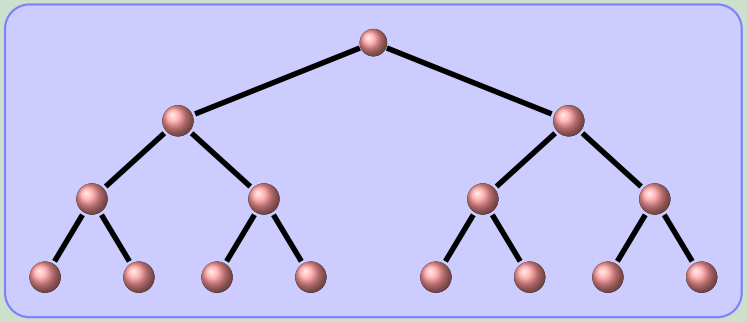
Example



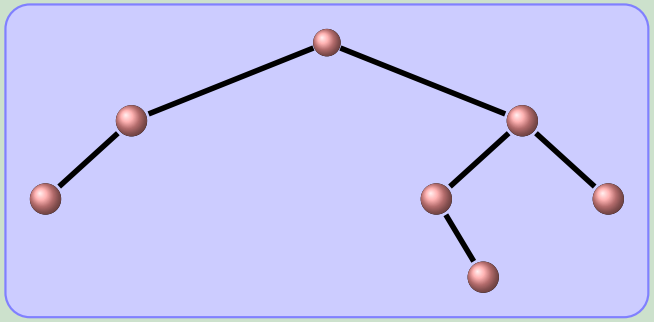
(1)二叉树



(2)完全二叉树



(3)完美二叉树

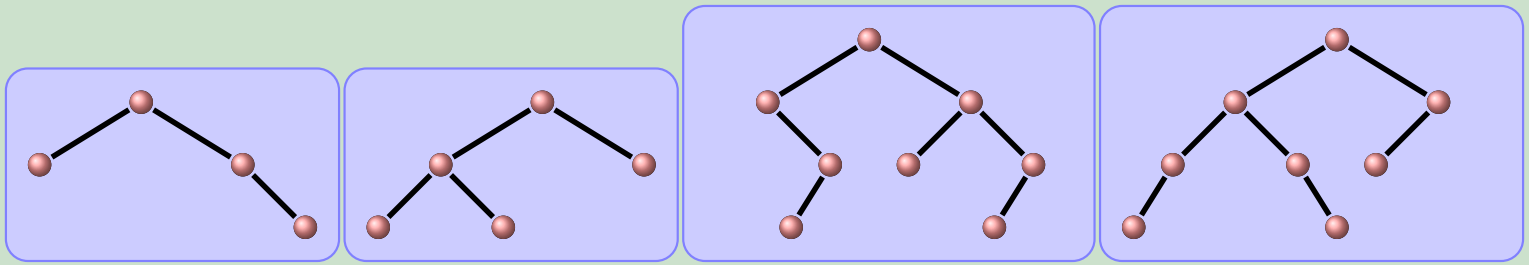


(4)位置二叉树

Example (2/2)

Example

设T是二叉树，如果T的每个结点的左子树与右子树的深度之差的绝对值$\leqslant 1$(约定：空树的深度为-1)，称这样的树为平衡树(balanced)；设$N(h)$为深度为h的平衡二叉树中的结点最小数目，$f_1, f_2, f_3, \ldots$是Fibonacci序列：



1. 试说明上图中哪些是平衡二叉树；
2. 试证明：$N(0) = 0, N(1) = 2, N(2) = 4$；
3. 试证明：$N(h) = 1 + N(h-1) + N(h-2)\ (h \geqslant 2)$；
4. 试证明：$N(h) = f_{h+2} - 1\ (h \geqslant 2)$；
5. 试证明：有n个结点的平衡二叉树满足：$h = \mathrm{O}(\lg n)$.

Solution(1/2)

Solution

1. $1, 2, 4$是平衡二叉树;
2. 深度为2至少有条深度为2的树叶，需要3个结点，但是要成平衡二叉树，树根的两个儿子的必须都存在，这样至少还要一个结点;
3. 设T_h是一颗深度为h并结点数最小平衡二叉树，设深度最长的树叶在其树根的左子树上，设T_l和T_r分别是树根的左右子树，则T_l是一颗深度为$h-1$并且结点数最小的平衡二叉树，否则，如果用结点数比该平衡树还要小的深度为$h-1$的树替换T_l后能够得到一个结点数比T_h还要少的深度为$h-1$的平衡二叉树，这与T_h是这样的树矛盾，同样T_r是颗深度为$h-2$并且结点数最小的平衡二叉树，这样$N(h) = 1 + N(h-1) + N(h-2)$.

□

Solution(2/2)

Solution

4. - Fibonacci序列: $f_0 = 0; f(1) = 1, \ldots, f_n = f_{n-1} + f_{n-2}, \ldots$;
   - $N(0) + 1 = f_3 = 2, N(1) + 1 = f_4 = 3, N(2) + 1 = f_3 + f_4 = f_5 = 5$;
   - $N(h) + 1 = (N(h-1) + 1) + (N(h-2) + 1) = f_{h+1} + f_h = f_{h+2}$;

   故: $N(h) = f_{h+2} - 1 \ (h \geqslant 2)$;

5. 用归纳法证明: $\forall n \geqslant 5,\ f_n > \left(\frac{3}{2}\right)^n$;

   设T是n个结点深度为h的平衡二叉树, 则:

$$n \geqslant N(h) = f_{h+2} - 1 > \left(\frac{3}{2}\right)^{h+2} - 1$$

   所以:

$$n + 1 > \left(\frac{3}{2}\right)^{h+2}$$

   两边取对数即得:

$$h < \log_{3/2}(n+1) - 2 = \mathrm{O}(\lg n)$$

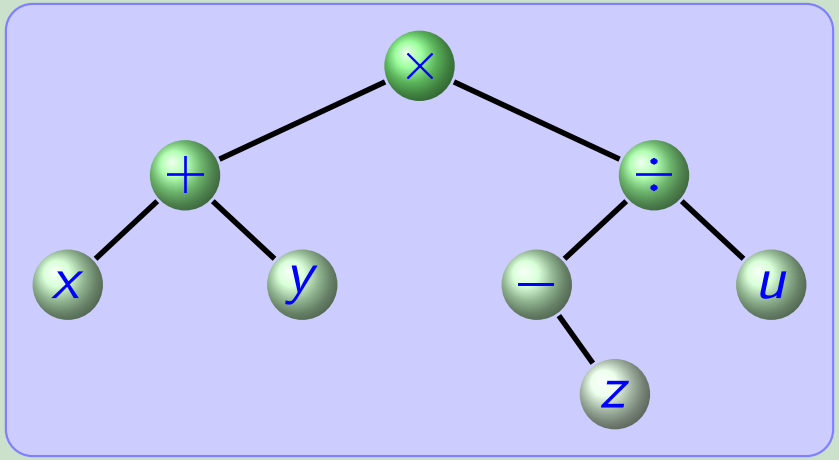
□

Tree Traversal

```
1  struct NODE {
2   struct NODE *left;
3   int value;
4   struct NODE *right;
5  }
6
7  preorder(struct NODE *curr) {
8     printf("%d", curr->value);
9     if (curr->left != NULL) preorder(curr->left);
10    if(curr->right != NULL) preorder(curr->right);
11 }
12 inorder(struct NODE *curr) {
13  if (curr->left != NULL) inorder(curr->left);
14  printf("%d", curr->value);
15  if (curr->right != NULL) inorder(curr->right);
16 }
17 postorder(struct NODE *curr) {
18    if (curr->left != NULL) postorder(curr->left);
19    if (curr->right != NULL) postorder(curr->right);
20    printf("%d", curr->value);
21 }
```

Example of Tree Traversal

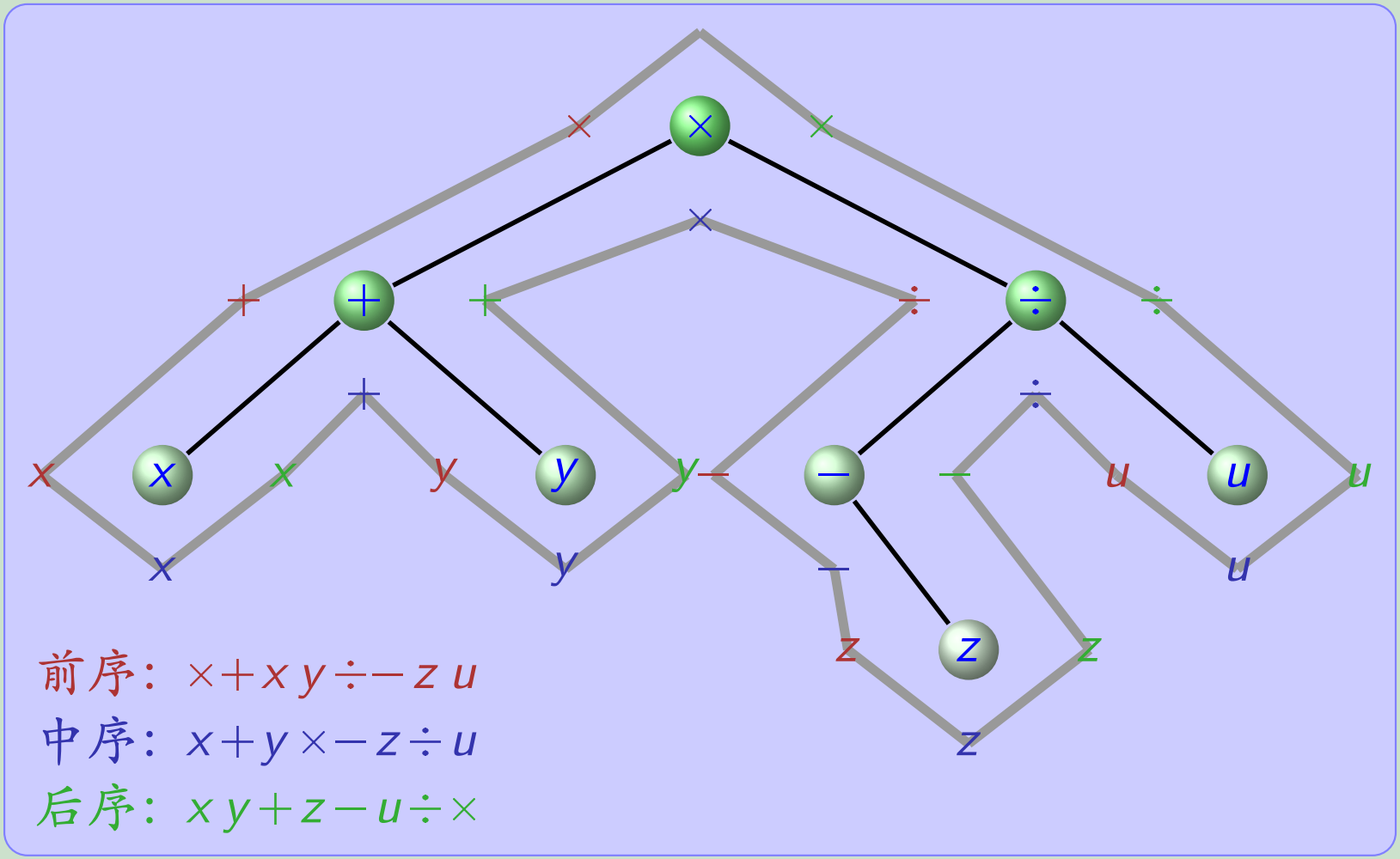
Example



1. Preorder: $\times + xy \div -zu$;
2. In-order: $x + y \times -z \div u$;
3. Postorder: $xy + z - u \div \times$;

DFS & Tree Traversal

DFS & Tree Traversal



Tree Traversal — Stack + While-loops

```
1  struct NODE {
2    struct NODE *left;
3    int value;
4    struct NODE *right;
5  }
6
7  preorder(struct NODE *curr) {
8     stack S;
9     push (curr);
10    while ( !empty(S) ) {
11    curr = pop(S);
12    printf("%d", curr->value);
13    if (curr->right != NULL) push(S,curr->right);
14    if (curr->left != NULL) push(S, curr->left);
15    }
16 }
```

Tree Traversal — Reference to parent + While-loops

```
1  struct NODE {
2    struct NODE *parent;/* an extra reference to its parent*/
3    struct NODE *left;  /* a while loop to travel the tree */
4    int value;          /* an example for in-order traversal*/
5    struct NODE *right;
6  }
7  visit(struct NODE *root) {
8    struct NODE *prev = NULL, *current = root,  next = NULL;
9    while (current != NULL) {
10     if (prev == current->parent) {
11       prev = current;
12       next = current->left; }
13     if (next == NULL || prev == current->left) {
14       print current->value;
15       prev = current;
16       next = current->right; }
17     if (next == NULL || prev == current->right) {
18       prev = current;
19       next = current->parent; }
20     current = next;
21     }
22 }
```

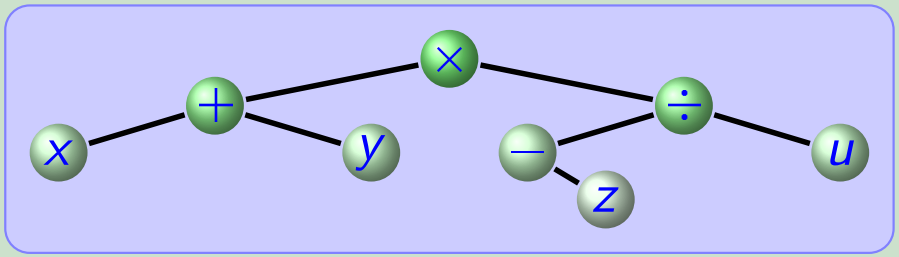
Tree Traversal & Traversal Order sequence

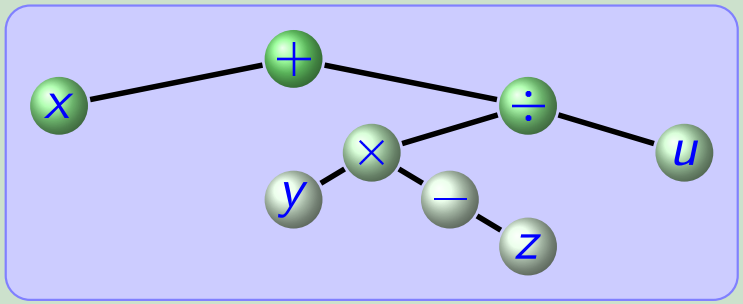
Proposition

- 设$v_1, v_2, \ldots, v_n$是树的前序(后序)遍历序列，并且每个结点的出度已知，则存在唯一的一个有向树使得其前序(后序)遍历序列与该序列一致；但是中序遍历与其遍历序列不是一一对应关系；
- 表达式对应前序(后序)遍历序列称为(逆)波兰表示((Reverse) Polish notation).

Example

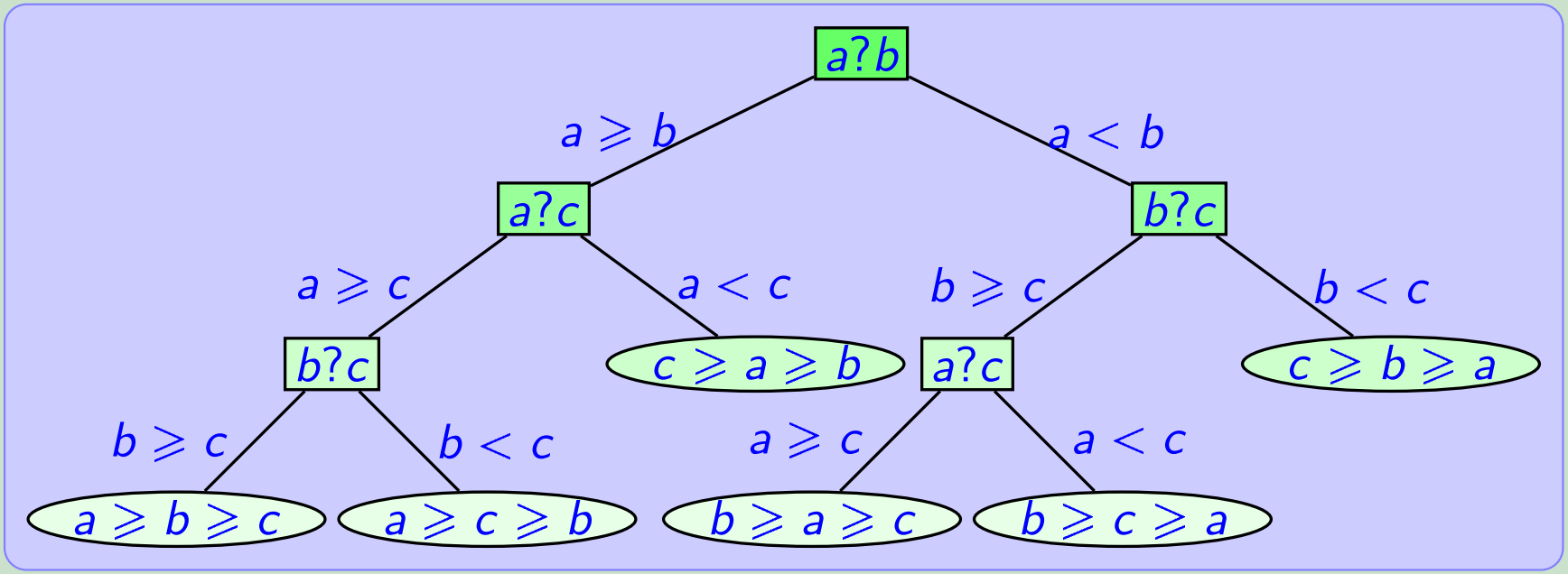
中缀表达式$x + y \times -z \div u$有多个表达式树与之对应:





决策树

Example of Decision Tree

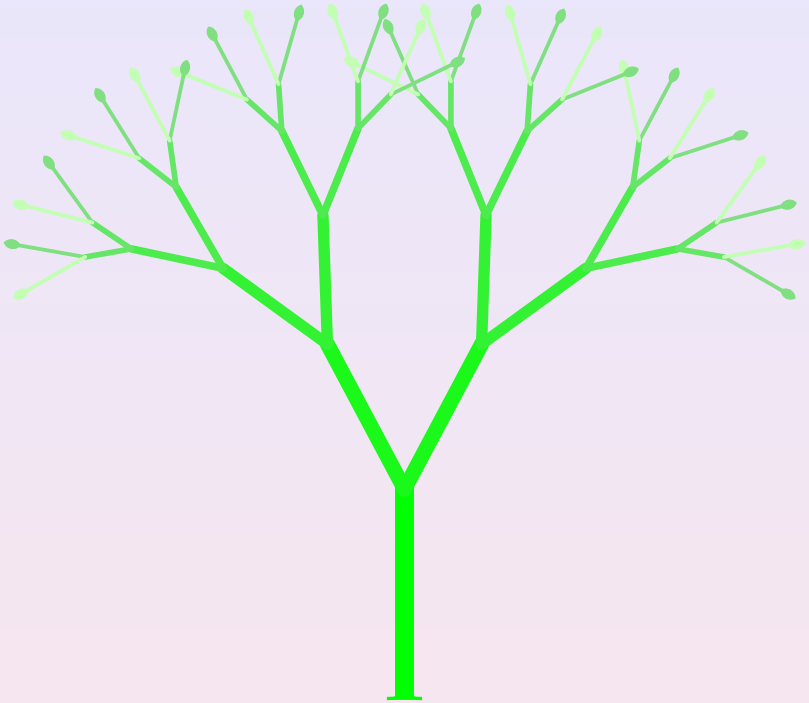


n个元素排序的决策树

n个元素排序的决策树

设T是n个元素的决策二叉树，h是深度，则：

- T有$n!$个叶结点；
- 深度为h完美二叉树有2^h个叶结点；2^h是深度为h的二叉树的叶结点的最大数；
- $\therefore n! \leqslant 2^h$，即$\log_2(n!) \leqslant h$；
- $\because n! \leqslant n^n$，So $\lg n! \leqslant n \lg n$，即：$\log_2(n!) = \mathrm{O}(n \lg n)$；
- 所以任何排序算法在最坏的情况下需要至少需要$\mathrm{O}(n \lg n)$次计算；
- 即最好的排序算法的时间复杂度为：$\mathrm{O}(n \lg n)$.



树是计算机的生命

TCS logo drawn by tantau

本章小节