

哈尔滨工业大学（深圳）2021 年春《数据结构》

第二次作业 树型结构

学号	姓名	成绩
200110619	梁鑫嵘	

表1

（一）概念题

1. 在二叉树的顺序存储结构中，实际上隐含着双亲的信息，因此可和三叉链表（含有父链指针）对应。假设每个指针域占4个字节，每个信息域占 k 个字节。试问：对于一棵有 n 个结点的二叉树，在顺序存储结构中最后一个节点的下标为 m ，在什么条件下顺序存储结构比三叉链表更节省空间？

Node
[12]Node *left, *right, *father; // 12个字节
[k]Type data; // k个字节()

在顺序储存结构中最后一个节点下标为 m ，则占用空间 mk 个字节；

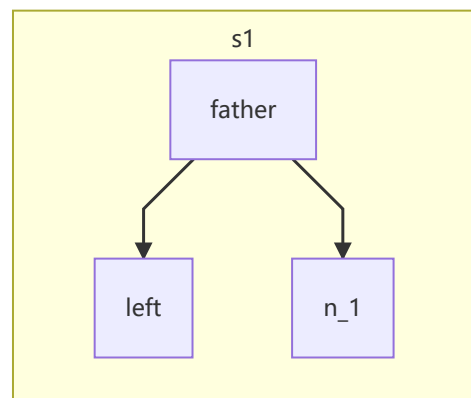
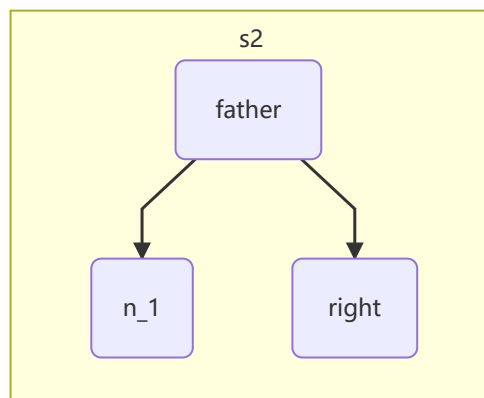
在三叉二叉树中有 n 个节点，每个节点占空间 $(12 + k)$ 个字节，总占用 $n(12 + k)$ 字节。

所以当 $n(12 + k) > mk$ 的时候，顺序储存比三叉链表更加省空间。

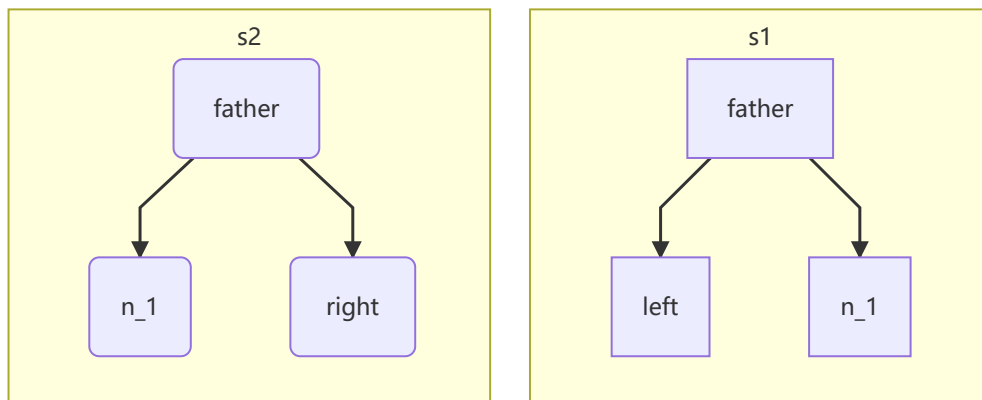
2. 对于二叉树 T 的两个结点 n_1 和 n_2 ，我们应该选择二叉树 T 结点的前序、中序和后序中哪两个序列来判断结点 n_1 必定是结点 n_2 的祖先？试给出判断的方法。（不需证明判断方法的正确性）

选择前序和后序遍历。方法如下：

1. 检查前序遍历序列，如 n_2 在 n_1 之前出现，如果是情况 s_2 则 n_2 是 n_1 祖先，如果是情况 s_1 说明 n_2 是 n_1 的祖先或者在 n_1 的左兄弟子树中；如果 n_2 在 n_1 之后出现，如果是情况 s_2 则 n_1 必然是 n_2 的祖先，如果是情况 s_1 则 n_2 在 n_1 的左兄弟子树。



2. 检查后序遍历序列，如 n_2 在 n_1 之前出现，如果是情况 s_1 则 n_1 是 n_2 祖先，如果是情况 s_2 则在 n_1 的左兄弟子树中；如果 n_2 在 n_1 之后出现，如果是情况 s_2 则 n_1 是 n_2 的祖先或者 n_2 是 n_1 的祖先，如果是情况 s_1 则 n_2 在 n_1 的左兄弟子树。



3. 一棵深度为 H 的满 k 叉树有如下性质：第 H 层上的结点都是叶子结点，其余各层上每个结点都有 k 棵非空子树。如果按层次顺序从1开始对全部结点编号，问：

- (1) 各层的结点数目是多少？
- (2) 编号为 p 的结点的父结点（若存在）的编号是多少？
- (3) 编号为 p 的结点的第 i 个儿子结点（若存在）的编号是多少？
- (4) 编号为 p 的结点有右兄弟的条件是什么？其右兄弟的编号是多少？

1. 第 h 层节点数目为 k^{h-1}
2. $\lfloor \frac{p}{k} \rfloor$
3. $kp + i$
4. $p \bmod k \neq k - 1$ ，编号为： $p + 1$ if $p \bmod k \neq k - 1$

4. 已一棵度为 k 的树中有 n_1 个度为1的结点， n_2 个度为2的结点， \dots ， n_k 个度为 k 的结点，问该树中有多少个叶子结点(n_0)？

节点数 = 分支数目 + 1, $\sum_{i=0}^k d_i = \sum_{i=0}^k i + x - 1$, 所以

$$\text{左式} = \sum_{i=0}^{\frac{k(k+1)}{2}} d_i = \sum_{i=0}^k i^2 = \frac{k(k+1)(2k+1)}{6} = \text{右式} = \frac{k(k+1)}{2} + x - 1, \text{求得} x = \frac{2k^3 - 2k + 6}{6}$$

(一) 算法设计

针对本部分的每一道题，要求：

- (1) 采用C或C++语言设计数据结构；
- (2) 给出算法的基本设计思想；
- (3) 根据设计思想，采用C或C++语言描述算法，关键之处给出注释；
- (4) 说明你所设计算法的时间复杂度和空间复杂度。

1. 已知一棵二叉树按顺序方式存储在数组 `int A[1..n]` 中。设计算法，求出下标分别为 i 和 j ($i \leq n, j \leq n$) 的两个结点的最近的公共祖先结点的位置和值。

基本设计思想：

1. 把 i 和 j 提升到同一层
2. 再每次向上一层寻找，直到找到相同的祖先

```

1 int findFather(int i, int j) {
2     if (i <= 1 || j <= 1) return 0;
3     if (i == j) return i / 2;
4     int t = 1;
5     // 使得 i 小于等于 j
6     if (i > j) {
7         int tmp = i;
8         i = j;
9         j = tmp;
10    }
11    if (2 <= i && i <= 3) {
12        // 特殊判断边界
13        t = 2;
14    } else {
15        // 得到i的高度
16        while (t * 2 < i) {
17            t *= 2;
18        }
19    }
20    // 降低到同一层
21    while (j > t * 2) j /= 2;
22    if (i == j) return i / 2;
23    while (i != j) {
24        i /= 2;
25        j /= 2;
26    }
27    return i;
28 }
29 // a[findFather(i, j)]即为所求的值

```

2. 假设二叉树引采用二叉链表存储，在二叉树 T 中查找值为 x 的结点，试编写算法打印值为 x 的结点的所有祖先，假设值为 x 的结点不多于一个。试分析该算法的时间复杂度。

```

1 #include <stdio.h>
2
3 using T = int;
4 const size_t TREE_MAX_NODE_NUM = 10;
5 class Tree {
6 public:
7     T data{};
8     Tree *left = nullptr, *right = nullptr;
9     Tree(T d) : data(d) {}
10 };
11
12 // 对树做前序遍历，每个节点都执行一遍某动作
13 template <typename F>
14 bool treeAply(Tree *t, Tree **stack, Tree **&top, F const &f) {
15     if (!t) return true;
16     if (stack != nullptr) *(top++) = t;
17     if (!f(t)) return false;
18     if (!(treeAply(t->left, stack, top, f) &&
19         treeAply(t->right, stack, top, f)))
20         return false;
21     if (stack != nullptr) top--;
22     return true;
23 }
24
25 void findAllFathers(Tree *t, T val) {
26     // 寻找到的目标节点
27     Tree *node = nullptr;
28     // 访问栈
29     Tree **stack = new Tree [TREE_MAX_NODE_NUM];
30     Tree **top = stack;
31     treeAply(t, stack, top, [&val, &node](Tree *tr) -> bool {
32         // 访问到目标节点就停止，栈保留到从根节点到目标节点的路径

```

```

33     // printf("visit: %d\n", tr->data);
34     if (tr->data == val) {
35         node = tr;
36         return false;
37     }
38     return true;
39 });
40 // 找不到
41 if (node == nullptr) return;
42 Tree **p = stack;
43 // 一路向下打印父亲节点
44 while (p != top - 1) {
45     printf("%d%s", (*p)->data, (p == top - 2 ? "\n" : " -> "));
46     p++;
47 }
48 }
49
50 int main() {
51     // 建立一棵树
52     Tree *root = new Tree(0);
53     root->left = new Tree(1);
54     root->right = new Tree(2);
55     Tree *left = root->left, *right = root->right;
56     left->left = new Tree(4);
57     left->right = new Tree(6);
58     left = left->left;
59     left->left = new Tree(7);
60     left->right = new Tree(9);
61     findAllFathers(root, 9);
62     return 0;
63 }

```

1. 使用前序遍历，复杂度 $O(N)$
 2. 每次访问节点都会修改一次路径栈，所以修改的时间复杂度是 $O(N)$
 3. 所以总体的时间复杂度是 $O(N)$
3. 一棵二叉树 T 的繁茂度定义为各层结点个数的最大值（也称二叉树的宽度）和二叉树的高度的乘积。试设计算法，求给定二叉树 T 的繁茂度。

```

1  #include <stdio.h>
2
3  using T = int;
4  const size_t TREE_MAX_NODE_NUM = 40;
5  class Tree {
6  public:
7      T data{};
8      Tree *left = nullptr, *right = nullptr;
9      Tree(T d) : data(d) {}
10 };
11
12 class Queue {
13 public:
14     Tree *data = nullptr;
15     Queue *next = nullptr;
16     int depth = -1;
17     Queue(Tree *p) : data(p) {}
18     Queue(int d) : depth(d) {}
19     Queue() {}
20     void link(Queue *q) { this->next = q; }
21 };
22
23 int BFS(Tree *t) {
24     int depthMax = 0;
25     int widthMax = 0;

```

```

26 Queue *front = new Queue(t), *temp = nullptr;
27 front->depth = 0;
28 Queue *top = new Queue(1);
29 front->link(top);
30 Tree *f = nullptr;
31 while (front != top) {
32     // pop
33     f = front->data;
34     int depthNow = front->depth;
35     if (depthNow > depthMax) depthMax = depthNow;
36     // printf("visit: %d | %d\n", depthNow, f->data);
37     temp = front;
38     front = front->next;
39     delete temp;
40     if (f->left != nullptr) {
41         top->data = f->left;
42         top->depth = depthNow + 1;
43         top->next = new Queue(nullptr);
44         top = top->next;
45     }
46     if (f->right != nullptr) {
47         top->data = f->right;
48         top->depth = depthNow + 1;
49         top->next = new Queue(nullptr);
50         top = top->next;
51     }
52     // 即将轮到下一层了
53     if (front->next != nullptr) {
54         if (front->next->depth > depthNow) {
55             Queue *t = front;
56             // 实际上是下一层的宽度
57             int widthNow = 0;
58             while (t != top) {
59                 t = t->next;
60                 widthNow++;
61             }
62             // printf("widthNow = %d\n", widthNow);
63             if (widthNow > widthMax) widthMax = widthNow;
64         }
65     }
66 }
67 return widthMax * depthMax;
68 }
69
70 int main() {
71     // 建立一棵树
72     Tree *root = new Tree(0);
73     root->left = new Tree(1);
74     root->right = new Tree(2);
75     Tree *left = root->left, *right = root->right;
76     left->left = new Tree(4);
77     left->right = new Tree(6);
78     right->left = new Tree(10);
79     left = left->left;
80     left->left = new Tree(7);
81     left->right = new Tree(9);
82     printf("BFS(root) = %d\n", BFS(root));
83     return 0;
84 }

```

4. 设计算法，对于二叉树 T 中每一个元素值为 x 的结点，删去以它为根的子树，并释放相应的空间。

```

1 #include <stdio.h>
2

```

```

3 using T = int;
4 const size_t TREE_MAX_NODE_NUM = 40;
5 class Tree {
6 public:
7     T data{};
8     Tree *left = nullptr, *right = nullptr;
9     Tree(T d) : data(d) {}
10    // 利用析构函数自动释放内存
11    ~Tree() {
12        printf("\t~deleting %d\n", this->data);
13        if (this->left) delete this->left;
14        if (this->right) delete this->right;
15    }
16 };
17
18 class Queue {
19 public:
20     Tree *data = nullptr;
21     Queue *next = nullptr;
22     Queue(Tree *p) : data(p) {}
23     Queue() {}
24     void link(Queue *q) { this->next = q; }
25 };
26
27 void BFS(Tree *t, T val) {
28     if (t->data == val) {
29         delete t;
30         return;
31     }
32     Queue *front = new Queue(t), *temp = nullptr;
33     Queue *top = new Queue(nullptr);
34     front->link(top);
35     Tree *f = nullptr;
36     while (front != top) {
37         // pop
38         f = front->data;
39         printf("visit: %d\n", f->data);
40         temp = front;
41         front = front->next;
42         delete temp;
43         if (f->left != nullptr) {
44             if (f->left->data == val) {
45                 delete f->left;
46             } else {
47                 top->data = f->left;
48                 top->next = new Queue(nullptr);
49                 top = top->next;
50             }
51         }
52         if (f->right != nullptr) {
53             if (f->right->data == val) {
54                 delete f->right;
55             } else {
56                 top->data = f->right;
57                 top->next = new Queue(nullptr);
58                 top = top->next;
59             }
60         }
61     }
62 }
63
64 int main() {
65     // 建立一棵树
66     Tree *root = new Tree(0);
67     root->left = new Tree(1);

```

```
68 | root->right = new Tree(2);
69 | Tree *left = root->left, *right = root->right;
70 | left->left = new Tree(4);
71 | left->right = new Tree(6);
72 | right->left = new Tree(4);
73 | left = left->left;
74 | left->left = new Tree(7);
75 | left->right = new Tree(9);
76 | BFS(root, 4);
77 | return 0;
78 | }
```