数据结构

Chap 2 Algorithm analysis

分析两种 Time complexities: $T_{avg}(N)$ $T_{worst}(N)$ N为输入

Space complexities

§2 Asymptotic /ˌæsɪmˈtɒtɪk/ (渐近) Notation (O, Ω, Θ, o)

① $T(N) = O(f(N))$ $T(N) \le cf(N)$

② $T(N) = \Omega(g(N))$ $T(N) \ge Cg(N)$

③ $T(N) = \Theta(h(N))$ $T(N) = O(h(N))$ $T(N) = \Omega(h(N))$

④ $T(N) = o(p(N))$ $T(N) = O(p(N))$ $T(N) \ne \Theta(p(N))$

$O(f(N))$ 为 $f(N)$ 下确界 $o(f(N))$ 为 $f(N)$ 真下确界

$\Omega(f(N))$ 为 $f(N)$ 上确界

$\Theta(f(N))$ 为 $f(N)$ 同阶

几个结论:

① 若 $T_1(N) = O(f(N))$, $T_2(N) = O(g(N))$

(a) $T_1(N) + T_2(N) = max(O(f(N)), O(g(N)))$

(b) $T_1(N) * T_2(N) = O(f(N) * g(N))$

② 若 T(N)是k次多项式, $T(N) = \Theta(N^k)$

③ $\log^k N = O(N)$, 对于任意k

计算法:

① loops: 循环次数

② nested for loops: 每层循环次数积

③ consecutive (连续) statements: 加和

④ if / else 最大为 判断条件+ if/else中较大者

⑤ recursions:

```
例: int Fib(int N) {          T(N)
      if (N <= 1)             O(1)
         return 1;            O(1)
      else
         return Fib(N-1)+Fib(N-2);
                              O(1) T(N-1) T(N-2)
    }
```

$\therefore T(N) = T(N-1) + T(N-2) + 2 \ge Fib(N)$

又 $\because (\frac{3}{2})^N \le Fib(N) \le (\frac{5}{3})^N$

$\therefore$ T(N) 为指数增长

问题1: 找出给定数列中子序列最大和

方案1: Divide and conquer

计算: 左边最大 MaxLeftSum = MaxSubSum(Array, Left, Center);

右边最大 MaxRightSum = MaxSubSum(Array, Center+1, Right);

以及从中间开始向左逐个加最大和 MaxLeftBorderSum, 向右逐个加最大和 MaxRightBorderSum } = MaxBorderSum

返回三者最大值

方案2: Online Algorithm

```
for(int i=1; i<N; ++i) {
   ThisSum += A[i];
   if (ThisSum > MaxSum)
      MaxSum = ThisSum;
   else if (ThisSum < 0)
      ThisSum = 0;
}
```

检查分析是否正确的方法:

法1: 若 $T(N) = O(N)$, 检查 $T(2N)/T(N)$ 是否为2 (约为)

若 $T(N) = O(N^2)$, 检查 $T(2N)/T(N)$ 是否约为 4

⋮

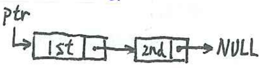
法2: 若 $T(N) = O(f(N))$ 检查 $\lim_{N\to\infty} \frac{T(N)}{f(N)} \approx Constant$ 是否成立

Chap 3 Lists, Stacks, and Queues

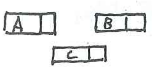
§1 Abstract Data Type

Data Type = {object} ∪ {operation}

Linked List



insertion: 先连后断



I. c→next = a→next

II. a→next = c

§2 List

object: (item₀, item₁, …, itemₙ₋₁)

① length ② print ③ empty ④ find k^{th}

⑤ insert ⑥ delete ⑦ next ⑧ previous

delete:

I. a→next = c→next

II. delete c

Doubly Linked Circular List

The Polynomial ADT

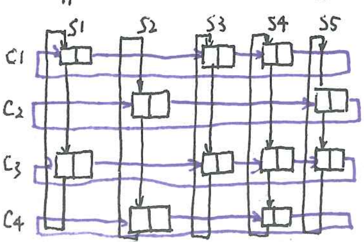
Object: $P(x) = a_1 x^{e_1} + \cdots + a_n x^{e_n}$ $\langle e_i, a_i \rangle$ a_i系数 e_i指数

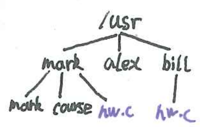
① find degree ② addition ③ subtraction ④ Multiplication

⑤ differentiation

Multilists

Suppose we have 40000 students and 2500 courses





二叉树性质:

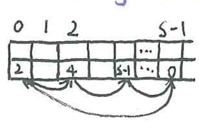
对任意二叉树,

令 n_0为叶子结点数,

n_2为degree为2的结点数,

有 $n_0 = n_2 + 1$

Cursor Implementation of Linked List



§3 The stack ADT

Last-in-First-out (LIFO)

Operations:

① isEmpty() ② CreateStack()

③ DisposeStack() ④ MakeEmpty()

⑤ Push() ⑥ Pop() ⑦ Top()

例 a*(b+c)/d

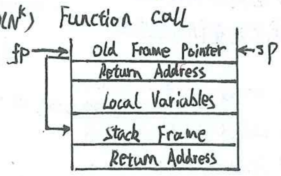
⇓

abc+*d/

a-b-c ⇒ ab-c-

$2\wedge2\wedge3\ (2^{2^3}) \Rightarrow 223\wedge\wedge$

Function call



Recursion can always be completely removed.

实现法:

① 链表

缺陷: malloc free 代价大.

解决方法: 用另一个大stack 存放闲置element.

free等效于从闲置stack push

malloc等效于从闲置stack pop

生成post fix

(1) 遇到数字直接输出 (两个数字-相邻也输出)

(2) 遇到符号push (前面只要有一个符号)

I. 如果push时栈内已有元素, 比较运算优先级

要push的大于top则入栈, 否则弹出top

(4) 遇到')' 直接pop到第一个pop'(' 止

(已在栈内的'(' 优先级降为最低)

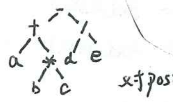
用处: I. 检查括号 (), [], {} 的左右平衡

II. postfix evaluation

infix: a+b*c-d/e

prefix: -+a*bc/de

postfix: abc*+de/-



对postfix运算

操作: I. 遇到数字push

II. 遇到符号pop两次并运算 将结果push

III. pop两次失败则报错

§4 The Queue ADT

First-In-First-Out (FIFO)

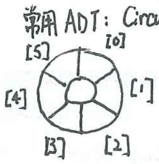
Operation:

① isEmpty() ② createQueue()

③ disposeQueue() ④ makeEmpty()

⑤ enqueue() ⑥ front() ⑦ dequeue()

常用ADT: Circular Queue:



当该Queue 仍有一空位时 该Queue需判断为Queue已满 不然其无法分辨当前是全空还是全满

Chap 4 Trees

§1 Preliminaries

① 一棵有N个点的树, 有N-1条边.

② degree of a node: 点上子树的数量

③ degree of a tree: $\max_{node \in tree} \{degree(node)\}$

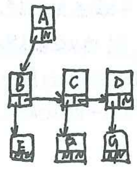
④ depth of n_i: length of the unique path from the root to n_i. Depth(root) = 0

⑤ height of n_i: length of the longest path from n_i to a leaf. Height(leaf) = 0

§2 Implementation

① List 版: 缺陷: 每个点大小将随分支数量而变化

② First Child - NextSibling 版:



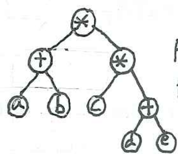
因为兄弟结点可以为任意顺序所以表示不惟一

```
void preorder(tree_ptr tree) {
   if (tree) {
      visit(tree);
      for (each child C of tree)
         preorder(C);
   }
}

void postorder(tree_ptr tree) {
   if (tree) {
      for (each child C of tree)
         postorder(C);
      visit(tree);
   }
}
```

Binary Trees

例: (a+b)*(c*(d+e)), 构建postfix表达式



inorder: a+b*c*d+e

postorder: ab+cde+**

preorder: *+ab*c+de

level order *+*abc+de

应用: 分层文件系统的目录列表

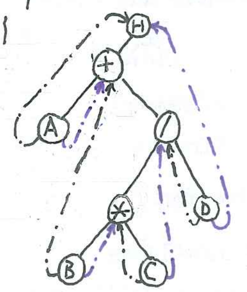
改进二叉树: Threaded Binary Tree

原始二叉树弱点: 有NULL指针 浪费空间

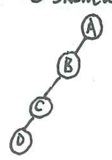
改进: 如果 Tree→left为null, 将其换成指向inorder前序结点的指针

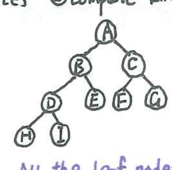
如果 Tree→Right为null, 将其换成指向inorder后序结点的指针

二叉树种类



① Skewed Binary Trees ② Complete Binary Tree





All the leaf nodes are on two adjacent levels