ADS

Main Topic:

(1) 用查找树动态查找. 让代价降低. 保证树的平衡与树的高度

让让树基本平衡: AVL: ① 树 结点左右高度差1 ② 满足查找树特征 左-右 左-左 右-左 右右

倒排索引: 概念

用高级结构解决动态查找问题

Splay ① 先find, 后将结点转动到顶 ② 摊还分析

B^+树 ① 多叉查找树 ② 插入、删除、结点分裂合并

红黑树 ① 红结点不能相邻 ② 从根到任何叶结点黑结点数相同

③ 新插点为红结点. 看父亲再调整 它.

(2) 有限资源分配. 用高级数据结构解决优先队列.

Structure property: 完全二叉树

order property: 堆

含并结构特征 快速 merge 的高速 Leftist heap

*斜堆 摊还分析 证明其高度为log n Skew heap O(log n)

(2) 算法设计思想与分析

大问题解决小问题

大问题由小问题的解决而解决

I. divide and conquer 递归解决问题

dynamic programming 从小问题开始

greedy, backtracking 从局部解找最优

以上除greedy均保证最优

II. 近似解: $\rho(n)$-approximation Knapsack: 2

approximation: Nextfit(2n-1) Firstfit 17M/10

Max-Cut: unless P=NP, no 17/16 approximation

Vertex cover Hopfield Neural Networks

Search: 定义一个neighbor Best fit 1.7M unless P=NP, there is no P-approximation for center-selection problem for any $\rho<2$

randomized algorithms: 陷入局部解, 跳出

III. 并行算法

用斐林表示集合: binomial heap

任何一个 B[i] 只能出现一次

查找完备性? (能否表达全部集合)

T(A), W(A) 一般理并行算法的负载

IV. 外部排序

二进制加法运算解决merge问题 V. NP、NP-c、分析.

Greedy:

Typical problems:

Activity Selection Problem: earliest finish first O(log c)

Huffman codes 设为树问题, 任何结点要么是0, 要么是2) c=字母数

二. Divide and Conquer Divide

Main ideas: three steps conquer combine typical problem: quick/merge sort, closest points

Running time: substitution 代 method, Recursion-tree method,

Master method (看谁影响更大, T(各)? f(n)?)

I. $a f(N/b) = K f(N), K<1, T(N) = \Theta(f(n))$

II. $a f(N/b) = f(N), T(N) = \Theta(f(n)\log_b N)$

III. $a f(N/b) < c f(N), T(N) = \Theta(f(n))$

$T(n) = aT(n/b) + f(n)$

I. $f(n) = O(N^{\log_b a - \varepsilon})$, $T(N) = \Theta(N^{\log_b a})$

II. $f(n) = \Theta(N^{\log_b a})$, $T(N) = \Theta(N^{\log_b a} \log N)$

III. $f(n) = \Omega(N^{\log_b a + \varepsilon})$, $T(N) = \Theta(f(N))$

Closest points $f(n) = O(n)$ NumPointInStrip $= O(\sqrt{n})$

三. Dynamic Programming

typical problem:

I: ordering Martix Multiplications

$M_{ij} = M_i \cdots M_j \Rightarrow M_{in} = M_l \cdots M_n = M_{il} * M_{l+1,n}$

$\Rightarrow b_n = \sum_{i=1}^{n-1} b_i * b_{n-i}$ where $n>1$ and $b_1 = 1$

$b_n = O(\frac{4^n}{n\sqrt{n}})$

Let m_{ij} be the cost of the optimal way to compute $M_i * \cdots * M_j$

M_i is an $r_{i-1} \times r_i$ Martix

$$m_{ij} = \begin{cases} 0 & \text{if } i=j \\ \min_{i \le l < j} \{ m_{il} + m_{l+1,j} + r_{i-1} r_l r_j \} & \text{if } j>i \end{cases}$$

if $j-i=k$, then the only values M_{pq} requires to compute M_{ij} satisfy $q-p<k$

Code:

```
void OptMatrix(const long r[], int N, TwoDimArray M)
{ int i, j, k, L;
  long ThisM;
  for(i=1; i<=N; i++) M[i][i]=0;
  for(k=1; k<N; k++) // k is right - left
    for(i=1; i<=N-k; i++) { // For each position
      j=i+k; M[i][j] = Infinity;
      for(L=i; L<j; L++) {
        ThisM = M[i][L] + M[L+1][j]
                + r[i-1]*r[L]*r[j];
        if (ThisM < M[i][j]) // update min
          M[i][j] = ThisM;
      } // end for L
    } // end for-Left
}
```

$O(N^3)$

II. Optimal Binary Search Tree

minimize the expected total access time.

$T(N) = \sum_{i=1}^{N} P_i \cdot (1 + d_i)$ word w_i probability for searching each w_i is p_i

$O(N^3)$

$C_{ij} = P_{(k)} + C_{i,k-1} + C_{k+1,j}) + P_{i,k-1} + P_{k+1,j}$

$= P_{ij} + C_{i,k-1} + C_{k+1,j}$

III. All-pairs shortest Path

$D^k[i][j] = \min\{\text{length of path } i \to \{l \le k\} \to j\}$

and $D^{-1}[i][j] = \text{cost}[i][j]$

shortest path is $D^k[i][j]$

① $k \notin$ the shortest path $i \to \{l<k\} \to j \Rightarrow D^k = D^{k-1}$; or

② $k \in$ the shortest path $i \to \{l<k\} \to j$

$=$ {the S.P. from i to k} $\cup$ {the S.P. from k to j}

$\Rightarrow D^k[i][j] = D^{k-1}[i][k] + D^{k-1}[k][j]$

$\therefore D^k[i][j] = \min\{D^{k-1}[i][j], D^{k-1}[i][k] + D^{k-1}[k][j]\}, k \ge 0$

```
void AllPairs(TwoDimArray A, TwoDim Array D, int N)
{ int i, j, k;
  for(i=0; i<N; i++) // initialize D
    for(j=0; j<N; j++)
      D[i][j] = A[i][j];
  for(k=0; k<N; k++)
    for(i=0; i<N; i++)
      for(j=0; j<N; j++)
        if(D[i][k]+D[k][j] < D[i][j])
          D[i][j] = D[i][k] + D[k][j];
}
```

Works if there are negative edge costs, but no negative-cost cycles

四. Backtracking

Main idea: exhaustive search + elimination

Typical problem: 8-queen {1, 2, 3, 4, 5, 6, 7, 8, 10}

$N(N-1)/2$ distance

game tree: α pruning β pruning

$x_1=0$ $x_6=10$ $x_2=2$ $x_5=8$

IV. product assembly $O(N)$ time $+O(N)$ space

f[line][stage] = min(

f[line][stage-1] + t_process[line][stage-1],

f[1-line][stage-1] + t_transit[1-line][stage-1]

find the next largest distance and check

How to design a DP method?

Characterize an optimal solution

Recursively define the optimal values

compute the values in some order

Reconstruct the solving strategy