高级数据结构

AVL 树

高度：空节点为 –1，单节点为 0

平衡因子：左子树高度 - 右子树高度

约束：

单旋转：LL（旋转左子树至根），RR

双旋转：LR（R旋转左儿子右子树至左儿子，L旋转左儿子至根），RL

复杂度：

$H = O(\log N)$

$T(N) = O(H) = O(\log N)$

伸展树（Splay Tree）

伸展：

被访问的节点为 X

1. X 父节点为树根，单旋转
2. X 有父节点与祖父节点
    1. 若 X 为 ZigZag 型，双旋转
    2. 若 X 为 ZigZig 型，一字型旋转
3. 递归向上进行至根

删除：

访问待删除元素，得到左右子树，访问左子树最大元素（一路向右）使其到达根，将右子树作为左子树右儿子。

二叉搜索树的遍历

升序列出：中序遍历

获得高度：后序遍历

获得深度：前序遍历

B+ 树

秩：M

约束：

1. 根为叶节点或含有 2 至 M 个子节点
2. 中间（Interior）节点有 ⌈M / 2⌉ 至 M 个子节点
3. 所有叶节点在同一深度上，包含数据数也在 ⌈M / 2⌉ 至 M 之间

性质：

1. 所有数据存储在叶节点上
2. 中间节点有 M 个指向叶节点的指针，及 M - 1 个叶节点中最小值在其中间。

插入：不满足时递归向上分裂

注意：非直接中间节点的索引值需要看右子树叶节点最小数据而非其右儿子最小索引

删除：节点过少时与兄弟合并，递归向上检查

复杂度：

```
D = O(log(⌈M / 2⌉, N))

T_Find(N) = O(logN)

T(M, N) = O(M*log(M, N)) = O(M/logM*logN)
```

倒排文件索引（Inverted File Index）

以关键字为键，（文档号，单词号）列表为值

Word Stemming：还原为词根

Stop Words：排除词（a，it）

实现：哈希、搜索树（B+，B-，Tire（字典树、前缀树））

左式堆（Leftist Heap）

零路径长（Null Path Length，NPL）：到无两儿子节点的最短路径长

计算：NPL(X) = min(NPL(Child of X)) + 1，空节点 NPL 为 –1

约束：左儿子零路径长 >= 右儿子零路径长

性质：最短路径为右路径（一路向右），右路径有 r 个节点则树至少有 2^r - 1 个节点

合并：将根元素较大的堆与根元素较小堆右儿子递归合并，合并时按需要交换左右儿子维持堆序

删除最小值：删除根节点，合并两儿子

复杂度：O(logN)

斜堆（Skew Heap）

复杂度：M 次操作 O(MlogN)，摊还时间 O(logN)

合并：除右路径上最后一节点无右儿子不必交换外，其他均无条件交换

迭代式合并：将右路径合并，左儿子不断作为右儿子

二项堆（Binomial Heap）

B_k 根节点有 k 个儿子，高度为 k 的 Bk 有 2^k 个节点，深度 d 处节点数为 C(d, k)

搜索最小元：遍历树根，O(logN)；记录最小元，O(1)

合并：二进制加法进位，O(logN)

插入：单节点合并，O(logN)，平均时间期望为常数

建堆：O(N)

删除最小元：删除最小元后树与其他树合并，O(logN)

实现：/* 书：链表，大小递减地保存树 */实现使用数组，`B_0 ~ B_n`

贪心法

哈夫曼码

使用字典树（Trie），左 0 右 1

频率：出现次数

代价：Sum(深度*频率)

哈夫曼树代码必须均在叶上以保证无歧义（前缀码）

哈夫曼算法：以树叶频率和为权，循环 N - 1 次，每次新建节点，从最小堆中取出两个树或节点作为左右儿子，更新权（相加）后放回

复杂度：T = O(NlogN)

调度算法

单处理器：

$$等待时间 = \sum{(N - k + 1)t_i_k} = (N + 1)\sum{t_i_k} - \sum{k*t_i_k}$$

使t_i_k非严格递增即可（Non-decreasing）

多处理器：

NP 难

Flow-Shop Scheduling：

任务分两部分分别被两处理器执行，后部分依赖前部分完成

（？）

若对任意一对任务J_i，J_j，有 $\min{J_i2, J_j1} \geq min{J_i1, J_j2}$，则可解。

将J_i_1，J_i_2非严格递增排序，每次取最小元素，若为J_i_1且J_i不在队列中则放在最左侧空位，若为J_i_2且J_i不在队列中则放在最右侧空位，循环

T = O(NlogN)

（比例）背包问题

最大效率项目优先，直到填满

近似装箱问题（Bin Packing）

将项目装入至最少书目的箱子

NP 难

联机算法（Online Algorithm）：每次给出项目后必须放置才能获得下一个项目

M：最优解的背包数目

有输入可以使任何算法均至少需要使用 4 / 3*M

Next Fit：如果当前箱能装则装，否则创建下一个箱子放入并关注之。不多于 2M，但有输入使其使用 2M - 2

First Fit：遍历所有箱子，放入第一个能装入的，否则创建新箱子放入。O(NlogN)。不多于`17/10*M`，但有输入使其使用`17/10*(M-1)`

Best Fit：放入能使其最紧凑的箱子。O(NlogN)。不多于 1.7M

脱机算法（Offline Algorithm）：最终给出答案

First Fit Decreasing：将项目按非严格递减排列，使用 First/Best Fit。不多于`11/9*M+4`，但有输入使其使用`11/9M`

动态编程（Dynamic Programming）

使用表记录结果，避免不必要的递归

斐波那契数

记录前两个结果。O(N)

矩阵乘法排序（*）

方法数量：$b_n = \sum*{i=1}^{n−1}{b_i{\cdot}b*{n-i}} = O(\frac{4^n}{\sqrt{n}})$

代价：`M_i`为$r*{i−1}{\times}r*{i}$矩阵，则代价$m_ij = \min*{i{\leq}l<j}{m*{il}+m*{{l+1}j}+r*{i−1}{r_l}r_j}, j > i$（$m_ii = 0$）

实现：初始化数组`M[i][i] = 0`；取 j - i 为 k，k 自 1 至 N - 1，i 自 1 至 N - k 迭代，j 相应等于 i + k，用 L 迭代 i 到 j - 1 使用以上公式（$m*{il}+m*{{l+1}j}+r_{i−1}{r_l}r_j$）取最小值赋给`M[i][j]`。可令用一 lastChange 二维数组记录 l 以得到实际结果

复杂度：T(N) = $O(N^3)$

最优二叉搜索树

简单贪心：保持二叉搜索树性质，取最大作为根

动态规划：

将树拆分为 Left, ⋯, i–1 与 i+1, ⋯, Right

\(C_{Left,Right} = \min_{Left{\leq}i{\leq}Right}{C_{Left,i-1}+C}_{i+1,Right}+\sum_{j=Left}^{Right}{p_j}\)

复杂度：$O(N^{3)或O(N}2)$

所有点对最短路径（*）

D_{k,i,j}为经由D_{1...k}自D_i至D_j最短路径，则D_{|V|,i,j}为最短路径 \(D_{k,i,j} = \min\{D_{k-1,i,j},D_{k-1,i,k}+D_{k-1,k,j}\}\)

实现：初始化数组D[i][j] = A[i][j]，k 自 0 至 N - 1 迭代，每次 i 自 0 至 N - 1 迭代，每次 j 自 0 至 N - 1 迭代，若D[i][k] + D[k][j] < D[i][j]则更新D[i][j]。可使用另一Path[i][j]记录 k

复杂度：O(N^3)

回溯（Backtracking）

深搜剪枝

八皇后问题

纵横斜均唯一

自距离重建点集（Turnpike Reconstruction）

1. N(N–1)/2 得到点数量
2. x_1 = 0，x_N = max_distance
3. 找下一个最大距离置于x_2或x_{N-1}并检查

分治（Divide and Conquer）

\(T(N) = aT(N/b)+\Theta(N^klog_pN)\)

\(T(N)=O(N^{log_ba}),a>b^k;O(N^klog_{p+1}N),a=b^k;O(N^klog_pN),a

若合并需O($N^{2)}$，则T=O(N_2)

最近点问题（Closet Points Problem）

分离：按 x 坐标排序，分为两集合

合并：取两侧最小距离最小值为中线左右正负偏差δ，形成一带，二重循环暴力寻找最小值。若带中点以 y 坐标排序，则若两点 y 距离大于δ则跳出内循环

NP 完全

哈密顿回路问题（单环路包含所有边）、单源头无权最短路径问题、单源头无权最长路径问题，三者均无多项式时间解

不可判定问题：停机问题：调用停机问题算法判断自身，反其道而行之（自

指，又与联机问题类似）

图灵机：改变状态，在当前位置擦除并写入符号，左、右、不移动

确定型图灵机：一般图灵机

非确定性图灵机：总是选择最好的步骤以得到解（可理解为图灵机树，总是取最优路径）

非确定性多项式时间可解（Non-deterministic Polynomial-time, NP）：若我们能在多项式时间内验证解的正确性，则为 NP

并非所有可判定问题均在 NP 中，如确定图中是否有哈密顿回路

NP 完全问题：任何 NP 问题均可在多项式时间内被规约为它。如哈密顿回路问题、背包问题（Knapsack problem）、旅行商问题、可满足性问题

若哈密顿回路问题 NP 完全，证明旅行商问题（Traveling Salesman Problem，TSP）（完全图，求是否有访问所有边且代价小于 K 的路径） NP 完全：因检查解正确性简单，故为 NP；将哈密顿回路中包含的边权值设为 1，不包含的边权值设为 2 并添加，则若 K 为 原边数 V 时旅行商问题有解等价于哈密顿回路问题有解（证明方法：NP 且可规约为 NPC）

第一个 NP 完全问题：满足性问题，能否给变量赋值使某表达式为真

摊还分析（Amortized Analysis）

摊还时间（Amortized Time）：M 次操作时间为 O(M*F(N))，则摊还时间为 O(F(N))

最坏时间 >= 摊还时间 >= 平均时间（？）

实际时间 + 势能 = 摊还时间