AVL

1. 结点个数：n(h)=n(h-1)+n(h-2)+1 ->保证树高logn
2. insert： 插入后，最多只有logn个节点不平衡，最多只需两次旋转恢复平衡
3. delete：刚删除完毕，最多只有一个节点不平衡，最多需要O(logn)次旋转恢复平衡
4. The balance factor BF(node) = hL - hR

红黑树

- 结点个数：算上NIL：奇数个
- 黑高：不包括自身，包括NIL。到叶子的path上的黑节点之和

  -> h(T)<=2bh(T)（每个node到leaf的path上红点数不超过黑点）
- 每个结点到叶子的path的长度之比不超过2
- A red-black tree with N internal nodes has height at most 2ln(N +1).
  - size(Tu) : = # internal nodes in Tu
  - size(Tu) >=2^bh(u)-1
  - bh(T) <=log2(n+1)
  - h(T)<=2log2(n+1)

插入：

1. 插入点标为红
2. 出现红红冲突后看父节点的兄弟节点
   1. 红：将其全部标记为黑，将父节点标记为红，向上推
   2. 黑或无：zigzig：将中间节点上拎；zigzag：将下面的节点上拎, 然后转化为zigzig

删除：

1. 只交换key不交换颜色
2. 看double black的兄弟节点：
   1. 黑：
      1. 孩子全为黑，左右减一个黑，double black 上移
      2. 右孩子为红 zigzig：把中间点拎起（全部标黑）
      3. 左孩子为红 zigzag：把最后点拎起（全部标黑）
   2. 红：

      把此节点拎起标黑，原先的父节点标红

delete 旋转最多3次，O(1)

均摊分析

- aggregation method：T(m)/m
- accounting method：借钱还钱
- potential function：势函数

  In general, a good potential function should always assume its minimum at the start of the sequence.

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) = \left(\sum_{i=1}^{n} c_i\right) + \underbrace{\Phi(D_n) - \Phi(D_0)}_{\geq 0}$$

Splay树

- w-v-u
- zig-zag：拎u两次
- zig-zig：先拎v，再拎u

B+树

- order B=3（2-3树） order = 4 (2-3-4 tree)
- internal node = [B/2] (上整) - B
- root = 2-B
- leaf = [B/2] (上整) - B
- # leafs <= N/ [B/2] (上整)
- space = 2N
- height <=O(logBN)

Leftist heap

- unbalanced
- The null path length, Npl(X), of any node X is the length of the shortest path from X to a node without two children.
  - Npl(NULL) = –1
  - Npl(X) = min { Npl(C) + 1 for all C as children of X }
- A leftist tree with r nodes on the right path must have at least 2^r – 1 nodes.

Skewed tree

- 均摊分析：
  - Di = the root of the resulting tree
  - Φ (Di) = number of heavy nodes
  - A node p is heavy if the number of descendants of p's right subtree is at least half of the number of descendants of p, and light otherwise. Note that the number of descendants of a node **includes the node itself.**
  - 在merge过程中，The only nodes whose heavy/light status can change are nodes that are **initially on the right path.**
  - 交换子树的过程中，原来重的都变轻了，而我们假设轻的都变重了以此获得Φ (Di) 的上界
  - the right path上的light nodes最多为lgn个。

Binomial Queue

- A binomial tree of height 0 is a one-node tree.
- Bk consists of a root with k children, which are B0,B1,B2...... . Bk has exactly 2^k nodes. The number of nodes at depth d is C(d,k) .
- A priority queue of any size can be uniquely represented by a collection of binomial trees.
- If the smallest nonexistent binomial tree is Bi , then Tp = Const · (i + 1).
- Performing N Inserts on an initially empty binomial queue will take **O(N) worst-case time.** Hence the average time is constant.
- A binomial queue of N elements can be **built by N successive insertions in O(N) time.**

operation	binary heap	leftist heap	skewed heap	binomial heap
make-heap	O(n)	O(n)	O(nlogn)	O(n)
find-min	O(1)	O(1)	O(1)	O(1)
insert	O(logn)	O(logn)	O(logn)	O(logn)
delete-min	O(logn)	O(logn)	O(logn)	O(logn)
merge	O(n)	O(logn)	O(logn)	O(logn)

Inverted File Index

- Index is a mechanism for locating a given term in a text.
- Inverted file contains a list of pointers (e.g. the number of a page) to all occurrences of that term in the text.
- Word Stemming：把派生词都变回原型
- Stop Words：such as “a” “the” “it”
- Distributed indexing：Each node contains index of a subset of collection
  - Term-partitioned index（A~C）
  - Document-partitioned index（1~10000）
- Dynamic indexing
- a search engine：How fast does it index/How fast does it search/Expressiveness of query language
- User happiness：
  - Data Retrieval Performance Evaluation > Response time > Index space
  - Information Retrieval Performance Evaluation > + How relevant is the answer set?

Backtracking

- N皇后问题——**NP-Hard**
  - For the problem with n queens, there are n! candidates in the solution space.
- The Turnpike Reconstruction Problem
- Tic-tac-toe
- α-β pruning: when both techniques are combined. In practice, it limits the searching to only O(根号n) nodes, where N is the size of the full game tree.
  - α pruning：max在上，min在下
  - β pruning：min在上，max在下

Greedy Algorithms

- Greedy algorithm works only if the local optimum is equal to the global optimum
- Beneath every greedy algorithm, there is almost always a more cumbersome dynamic-programming solution
- Activity Selection Problem
  - Given a set of activities S = { a1, a2, ..., an } that wish to use a resource (e.g. a classroom). Each ai takes place during a time interval [si, fi).
  - 按最早结束的排序——O(nlogn)
- Huffman Codes——O(nlogn)
  - optimal prefix tree：full tree

Divide and Conquer

- Cases solved by divide and conquer
  - The maximum subsequence sum – the O(N log N) solution
  - Tree traversals – O(N)
  - Mergesort and quicksort – O(N log N)
- Closest Points Problem
  - Given N points in a plane. Find the closest pair of points.
  - Sort according to x-coordinates and divide;
  - Conquer by forminga solution from left,right, and cross.
- 主定理

Dynamic Programming

- ww

NP-Completeness

- 停机问题——incomputable（undecidable）
- computable——complexity class/P、NP、ENP
- 图灵机
  - A Deterministic Turing Machine executes one instruction at each point in time. Then depending on the instruction, it goes to the next unique instruction.
  - A Nondeterministic Turing Machine is free to choose its next step from a finite set. And if one of these steps leads to a solution, it will always choose the correct one.
- **NP**—— **Nondeterministic polynomial-time**
  - The problem is NP if we can prove any solution is true in polynomial time.
- **NP-complete**——An NP-complete problem has the property that any problem in NP can be polynomially reduced to it.

  If we can solve any NP-complete problem in polynomial time, then we will be able to solve, in polynomial time, all the problems in NP
- 经典NPC问题
  - The Hamilton Cycle Problem
  - the Travelling Salesman Problem
  - the Clique Problem
  - the vertex cover problem
  - **The first problem that was proven to be NP-complete was the Satisfiability problem (Circuit-SAT)**

    proved it by solving this problem on a nondeterministic Turing machine in polynomial time.
- language

- P = { L ⊆ {0, 1}* : there exists an algorithm A that decides L in polynomial time }
- A language L belongs to NP iff there exist a two-input polynomial-time algorithm A and a constant c such that L = { x ∈ {0, 1}* : there exists a certificate y with |y| = O(|x|c) such that A(x, y) = 1 }. We say that algorithm A verifies language L in polynomial time.
- complexity class co-NP = the set of languages L such that $\overline{L} \in NP$

- 规约
  - **Hamiltonian cycle problem** vs **Traveling salesman problem**
    - Hamiltonian cycle problem: Given a graph G=(V, E), is there a simple cycle that visits all vertices? 【已知NPC】
    - Traveling salesman problem: Given a complete graph G=(V, E), with edge costs, and an integer K, is there a simple cycle that visits all vertices and has total cost <= K?
    - TSP is obviously in NP, as its answer can be verified polynomially.
    - **G has a Hamilton cycle iff G' has a traveling salesman tour of total weight |V|.**
    - HCP -> TSP
  - **Clique problem** vs **Vertex cover problem**
    - Clique problem: Given an undirected graph G = (V, E) and an integer K, does G contain a complete subgraph (clique) of (at least) K vertices? 最大团（最大子图）【已知NPC】
    - Vertex cover problem: Given an undirected graph G = (V, E) and an integer K, does G contain a subset V' ∈ V such that |V'| is (at most) K and every edge in G has a vertex in V' (vertex cover)? （每条边至少有一个端点被选中）
    - Cique -> vertex cover
    - **G has a clique of size K iff G的补图 has a vertex cover of size |V| - K.**

Approximation

- **【Definition】 An *approximation scheme* for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem, but also a value $\varepsilon > 0$ such that for any fixed ε, the scheme is a (1+ ε)-approximation algorithm.**
  We say that an approximation scheme is a *polynomial-time approximation scheme* (*PTAS*) if for any fixed $\varepsilon > 0$, the scheme runs in time polynomial in the size n of its input instance.
- PTAS 只要求**对输入实例的规模n为多项式时间复杂度**，而不考虑ε。因此当算法运行时间复杂度为O(n^1/ε)甚至O(n^exp(1/ε)) 时，仍是PTAS算法。
- **fully polynomial-timeapproximation scheme (FPTAS)** ——要求算法**对问题规模n和1/ε都是多项式时间复杂度的。**
- **Bin Packing —— NP-hard**
  - On-line Algorithms
    - Next Fit

            - Let M be the optimal number of bins required to pack a list I of items. Then next fit never uses more than 2M – 1 bins. There exist sequences such that next fit uses 2M – 1 bins.
        - First Fit
            - Let M be the optimal number of bins required to pack a list I of items. Then first fit never uses more than 17M / 10 bins. There exist sequences such that first fit uses 17(M – 1) / 10 bins.
        - Best Fit
            - Place a new item in the tightest spot among all bins.
            - T = O(N log N) and bin no. ≤ 1.7M
        - There are inputs that force any on-line bin-packing algorithm to use at least **5/3** the optimal number of bins.
    - Off-line Algorithms
        - FFD/BFD
            - Let M be the optimal number of bins required to pack a list I of items. Then first fit decreasing never uses more than **11M / 9 + 6/9** bins. There exist sequences such that first fit decreasing uses 11M / 9 + 6/9 bins.
            - 3/2
    - The Knapsack Problem — **NP-hard**
    - The K-center Problem — **NP-hard**
        - **Unless P = NP, there is no p -approximation for center-selection problem for any p < 2.**

Local Search

- Vertex cover problem: Given an undirected graph G = (V, E). Find a minimum subset S of V such that for each edge (u, v) in E, either u or v is in S.
- Hopfield Neural Networks：
    - Graph G = (V, E) with integer edge weights w (positive or negative).
    - 好边：weSuSv<0
        - If we < 0, where e = (u, v), then u and v want to have the same state;
        - if we > 0 then u and v want different states.
    - The absolute value |we| indicates the strength of this requirement.
- **Maximum Cut Problem — NP-hard**
    - Given an undirected graph G = (V, E) with positive integer edge weights we, find a node partition (A, B) such that the total weight of edges crossing the cut is maximized.
    - **Unless P = NP, no 17/16 approximation algorithm for MAX-CUT.**
- K-L heuristic

Randomized Algorithms

-

Parallel Algorithms

- Parallel Random Access Machine (PRAM)
- Work-Depth (WD)
- ☞ Exclusive-Read Exclusive-Write (EREW)

  ☞ Concurrent-Read Exclusive-Write (CREW)

  ☞ Concurrent-Read Concurrent-Write (CRCW)

  Measuring the performance

  ☞ **Work load – total number of operations: $W(n)$**

  ☞ **Worst-case running time: $T(n)$**

  - $W(n)$ operations and $T(n)$ time
  - $P(n) = W(n)/T(n)$ processors and $T(n)$ time (on a PRAM)
  - $W(n)/p$ time using any number of $p \le W(n)/T(n)$ processors (on a PRAM)
  - $W(n)/p + T(n)$ time using any number of p processors (on a PRAM)

  All asymptotically equivalent
- **The summation problem**
  - 串行：W=O(n) D=O(n)
  - 树：W=O(n) D=O(logn)
- **Prefix sum**：前1、2、3......i个数的和
  - serial：W=O(n) D=O(n)
  - native：W=O(n^2) 每个都算一次 D=(logn)
  - 树：W=O(n) D=O(logn)
- **Parallel merge sort**
  - 第i层有2^i个节点，每个节点有n/2^i个元素
  - Wv=O(n/2^i) Wi=O(n) W=O(nlogn)
  - Dv=O(n/2^i) Di=O(n/2^i) D=O(n)
  - **Merge 加快**

        - input：sorted array A and B
        - output：sorted C
        - serial： W = O(n)   D = O(n)
        - 假设已经知道 rank(i,B) 和 rank(i,A) —— W=O(n) D=O(1)
    - **Ranking**
        - serial ranking —— W = O(n) D = O(n)
        - binary search —— W = O(nlogn) D = O(logn)
        - parallel ranking
            - 把A和B每间隔k划分
            - 在组与组之间用binary search ranking —— W1 = O(2n/k * logn) D1 = O(logn)【最多2n/k组】
            - 在一组中用serial ranking —— W2 = O(n) D2 = O(k) 【一组最多2k个数】
            - W = O(n) D = O(logn)
    - **D = O(logn^2)**
- **Maximum finding**
    - serial : W=O(n) D=O(n)
    - use summation alg.【 每两个选其中大的那个】： W=O(n) D=O(logn)
    - 并行比较每两个数：W=O(n^2) D=O(1)
    - Divide - and -conquer：
        - 把n分成根号n个子问题
        - 用并行比较每两个数来解决根号n个数

$$W(n) = \sqrt{n}W(\sqrt{n}) + O(\sqrt{n}^2)$$
$$D(n) = D(\sqrt{n}) + O(1)$$

        W=O(nloglogn) D=O(loglogn)
    - 分为k组，用D&C找到k组中最大的数
        - W=O(n) D=O(loglogn)
    - random sampling
        - W=O(n) D=O(1) with high probability 1-1/n^c return maximum

External Sorting

- k-way merge
    - In general, for a k-way merge we need 2k input buffers and 2 output buffers for parallel operations.
- polyphase merge
    - k+1 tapes for k-way merge