§ Lec 1. AVL. Splay. Amortized Analysis.

1.1. AVL. Tree.

~~Balance~~

Target: Speed up Searching.

By balance. $h = O(n) \to O(\log n)$.

Def 1.1. Height balanced:

1. An empty binary tree is height balanced.
2. If T is a nonempty binary tree with T_L and T_R. then T is height balanced iff.
   1) T_L & T_R is balanced
   2) $|h_L - h_R| \le 1$.

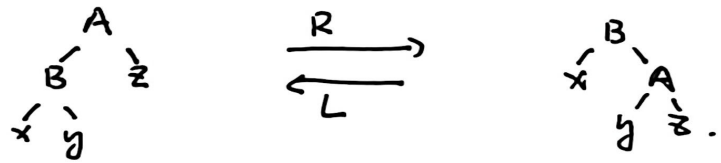
   The empty tree height $= -1$.

Def 1.2. Balance factor

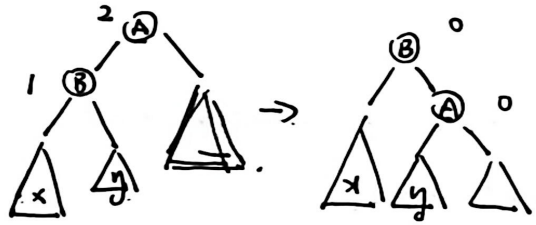
BF (node) $= h_L - h_R$.

BF $= -1, 0, 1$.

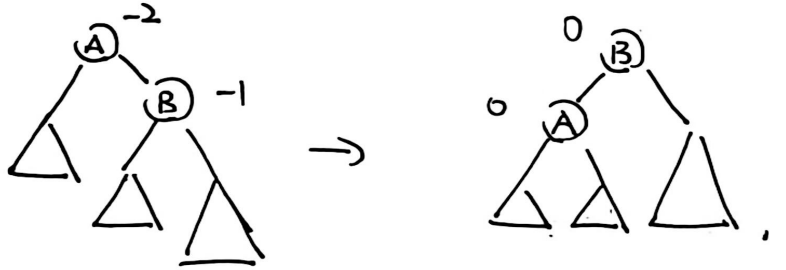
1.1. Tree Rotation



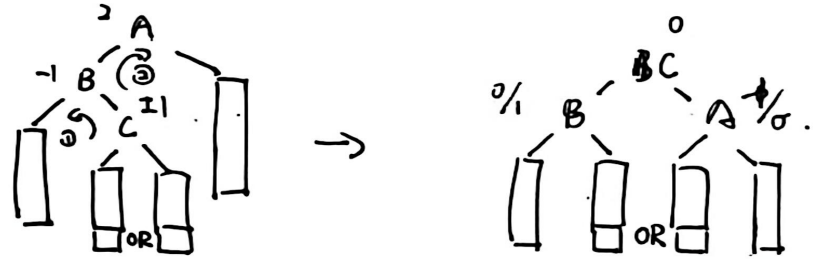
1) LL



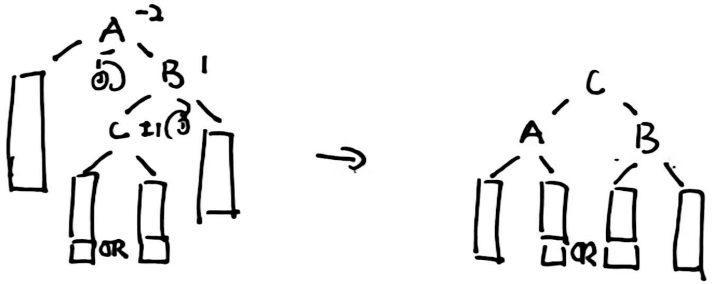
2) RR



3.) LR



4) RL.



1.2. Time Complexity of AVL.

$h = O(\log n)$.

Proof: To prove. n_h (the minimum #nodes in a height-balanced tree) $= F_{h+2} - 1 = O((\frac{\sqrt{5}+1}{2})^h)$.

$n_h = n_{h-1} + n_{h-2} + 1$.

$(n_{-1} = 0,\ n_0 = 1,\ n_1 = 2.)$

2. Splay Tree.

Target: Any M consecutive tree operations starting from an empty tree take at most $O(M \log n)$ time. (Amortized $O(\log n)$.).

2.1. Insertion.

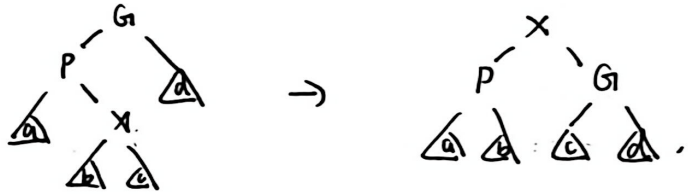
For any nonroot node X. denote its parent by P. grandpa by G.

1) P is the root:

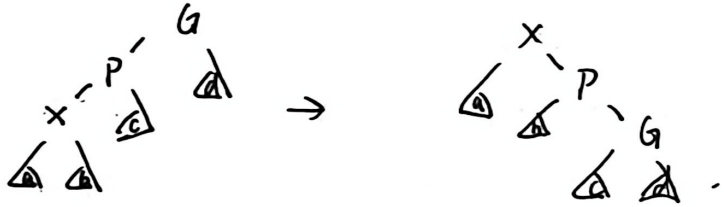
Rotate X & P.

2) P is not the root:

zig-zag

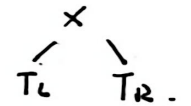


zig-zig



2.2. Deletions.

1) Find X.



2) Remove X.

3) Find Max(T_L).

4) Make T_R the right child.

3. Amortized Analysis. worst ≥ amortized ≥ average/expected.

3.1. Aggregate.

Show that for all n, a sequence of n operations takes worst-case time T(n) in total. In the worst case, the average cost, or amortized cost, per operation is therefore $\frac{T(n)}{n}$.

3.2. Accounting 记账法.

amortized $\hat{c}_i$. ~~error~~ actual cost c_i.

when an operation's $\hat{c}_i > c_i$, we assign the diff $\hat{c}_i - c_i$ as credit.

Credit can help pay for later ops that $\hat{c}_i < c_i$.

The most:

$$\forall n,\ \sum_{i=1}^{n} \hat{c}_i > \sum_{i=1}^{n} c_i$$

3.3. Potential 势能法.

The credit.

$$\hat{c}_i - c_i = \phi(D_i) - \phi(D_{i-1}).$$

$\phi(D_i)$:= potential function.

D := the data struct.

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \phi(D_i) - \phi(D_{i-1})).$$

$$= (\sum_{i=1}^{n} c_i) + \phi(D_n) - \phi(D_0).$$

$$\Rightarrow \quad \phi(D_n) - \phi(D_0) \geq 0.$$

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

势能函数在代价较高的操作后会有较大的下降.

Exercise:

1、multipop-stack. ~~Binary Counter~~.

	c_i	$\hat{c}_i$
push	1	2
pop	1	0
multipop	min(k, size(S_i))	0

② Aggregate:

$T(n) = T(\text{push}) + T(\text{pop} + \text{multipop})$

~~$n + n = 2n = O(n)$~~ ~~$2T(\text{push})$~~.

$= 2T(\text{push}) \leq 2(n-1) = O(n)$.

② Accounting:

	Δi
push	1
pop	0
multipop	0

③ Potential:

$\phi(S_i) = \text{size}(S_i)$.

2. Binary Counter.

0		
1	1	2
10	2	2
11	1	2
100	3	2
101	1	2
110	2	2
111	1	2
1000	4	2

.

① Aggregate:

$T(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots = 2n = O(n)$.

$\frac{T(n)}{n} = O(1)$.

② Accounting:

对二次置位设置 $\hat{c_i} = 2$. 为复位做准备.

Increment 至多置位一次.

③ Potential:

$\phi(D_i) =$ 共1s.

3. $c_i = \begin{cases} i & \text{if } i = 2^k \\ 1 & \text{otherwise.} \end{cases}$

$\hat{c_i}$
3
2.

① Aggregate:

$T(n) \le n + \sum_{i=0}^{\log_2 n} 2^i < n + 2n = 3n$.

$\hat{c_i} = \frac{T(n)}{n} < 3 = O(1)$.

② Accounting:

~~$\hat{c_i} = 2$~~. $\hat{c_i} = 3$.

③ Potential:

$\phi(D_i) = 2(i - 2^{\lfloor \log_2 i \rfloor})$.

4. Splay Tree.

Tamortized $= O(\log N)$.

$\phi(T) = \sum_{i \in T} \log S(i)$.

§ Lec 2. Red-Black Tree and B+ Tree.

1. Red-Black Trees

Target: Balanced binary search tree.

Def. II.1. A red-black tree is a binary search tree that satisfies the following red-black properties:

1) Every node is either red or black.

2) The root is black.

3) Every leaf (NIL) is black.

4) If a node is red, then both its children are black. 자손검사.

5). For each node, all simple paths from the node to descendant leaves contain the same number of black nodes. 블랙높이.

Def. II.2. The black-height of any node x. denoted by bh(x). is the number of black nodes on any simple path from x (x not included) down to a leaf.

Lemma 1.1. $h_N \le 2\ln(N+1)$.

Proof. To prove: 1) $size(x) \ge 2^{bh(x)} - 1$. (수학적귀납법)

2) $bh(Tree) \ge \frac{h(Tree)}{2}$, (명백).

So $\Rightarrow$ $size(root) \ge 2^{bh-1} \ge 2^{\frac{h}{2}} - 1$.

1.1. Insert.

Insert node as n in red. parent p. uncle u. grandpa g. Symmetric.

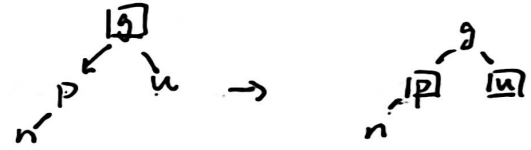
1) The tree is empty, insert the first node.

2) p is root and black. do nothing.

4) Case 4.

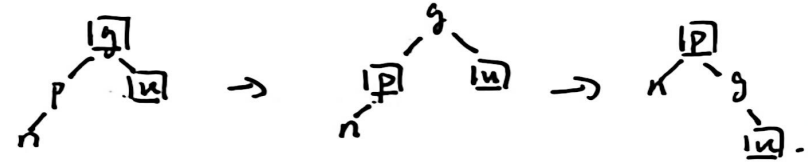
a. Dye p and u black- g red.

b. Recursively maintain g node.



5) Case 5. rotate p and then in Case 4

6) Case 6.



a. Dye p black and g red.

b. Rotate g.

1.2. Delete:

1) Case 0: n is root. delete it.

2) Case 1: n has degree 2, replace n by the predecessor, then delete the replacing node from the subtree.

After that need to be remaintained

3) Case 2: n has degree 1. replace n by s.

If n is black:

Dye s black

If s is already black: remaintain

4) Case 3: n has degree 0. delete it.

If n is ~~red~~ black, remaintain.

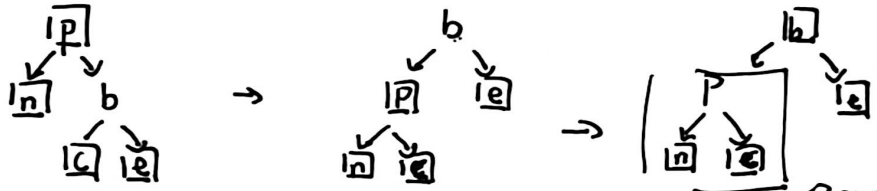
Re-maintained:

1) Case 1:

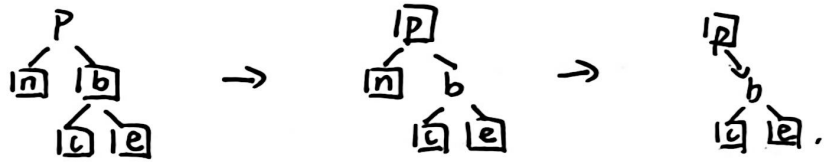
a. Rotate p.

b. Dye b blaik and p red.

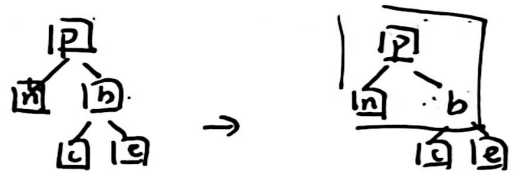
c. maintain subtree rooted at p.



2) Case 2: Dye b red and p black.



3) Case 3: Dye b black, recursively meintain p.

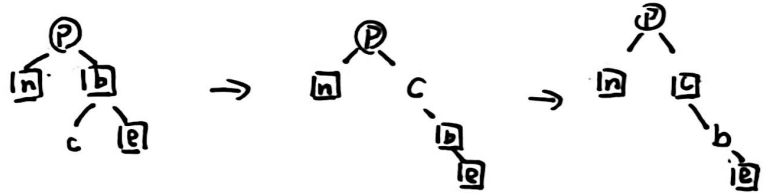


4) Case 4:

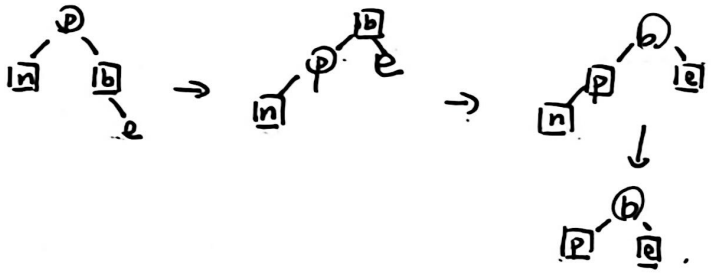
a. Rotate b.

b. Dye c red, b black.

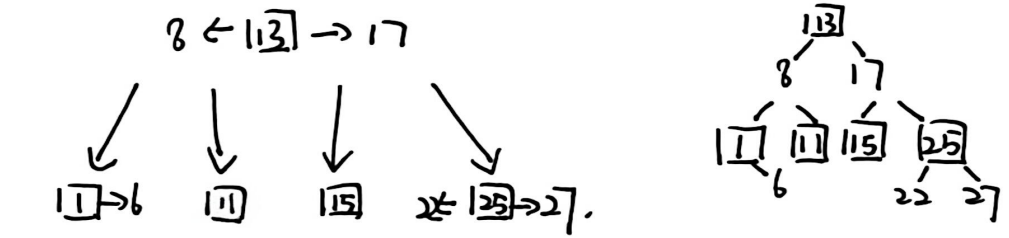
c. Then in Case 5.

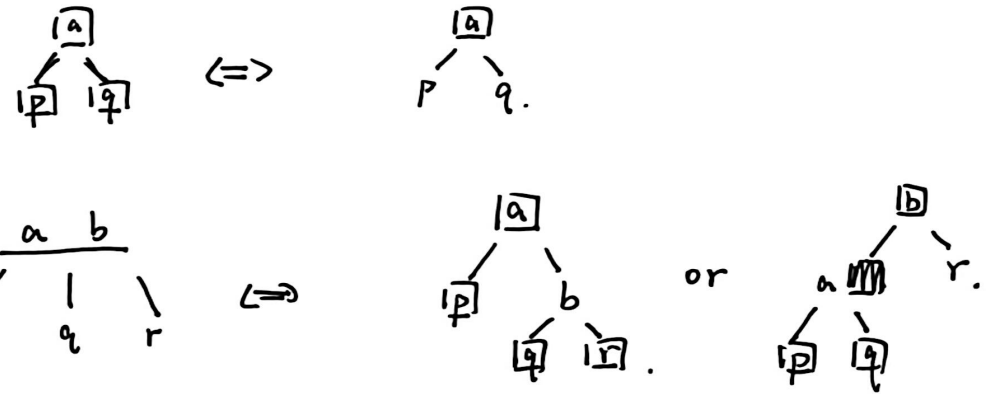


5) Case 5:



1.3 Isomorphism with 2-3-4 B tree.





2. B+ Trees.

Def II.3. A B+ tree of order M is a tree with the following structural properties:

1) The root is either a leaf or has between 2 and M children.

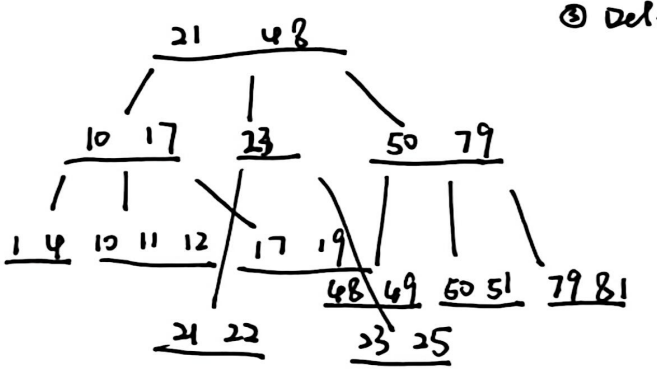
2) All nonleaf nodes (except the root) have between $\lceil \frac{M}{2} \rceil$ and M children.

3) All leaves are at same depth.

① find
② Insert
③ Delete.

$h = O(\log_{\lceil \frac{M}{2} \rceil} N)$.

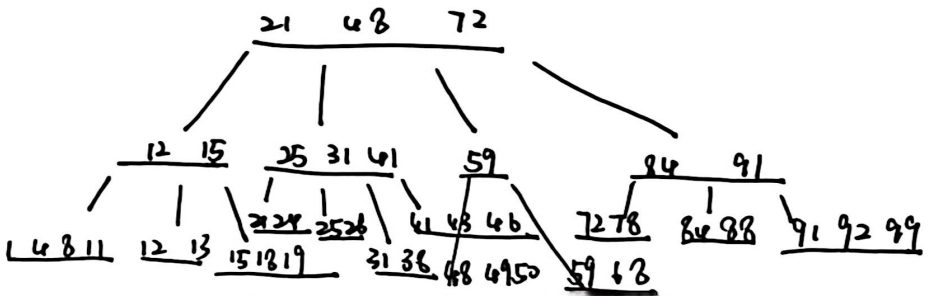
2-3 B+



1~2 elements.
2~3 children.

2~3 elements.

2-3-4 B+.



1. 2. 3 eles.
2. 3. 4 children

2. 3. 4 eles.

3. 2-3-4 (4阶 B 树)

1. 所有叶子节点深度相同.

2. 节点只能是 2-节点. 3-节点. 4-节点之一.

# ele	1	2	3
# children	2	3	4

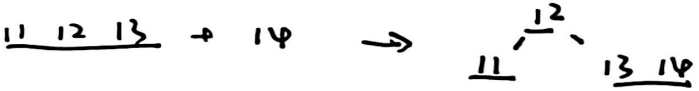
3. 查找树的性质.

插入.

1. 都是向最下面一层插入.

2. 升元: 2-节点 → 3节点 或 3-节点 → 4-节点.

3. 向 4-结点插入元素后, 需要将中间元素提到父结点升元. 原结点变为两个 2-节点. 再把元素插入 2-结点中. 如果父节点也是 4-结点. 则继续向上层升元. 至根节点使树高 +1.



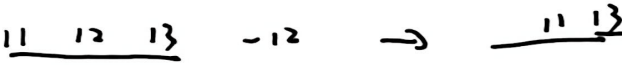
删除.

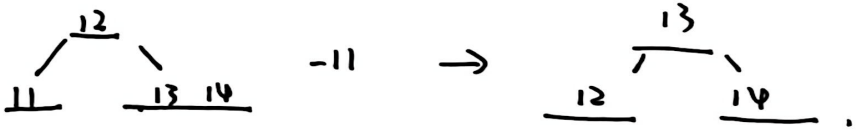
1. 查找最近的叶子节点中的元素替代被删除元素.
删除替代元素后, 从替代元素所处叶子节点开始处理.

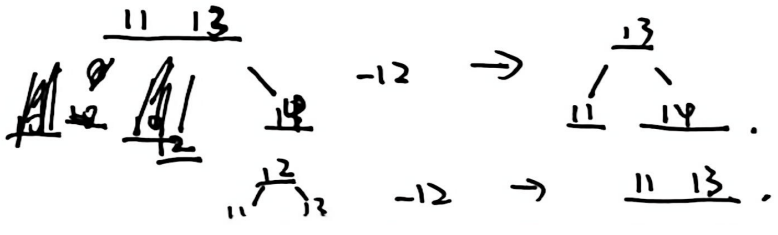
2. 降元.

3. 2节点中只有一个元素. 借兄弟节点中的元素补充删除后的空结点.

4. 兄弟也没有时, 将空结点降元. 实现的向上递归.







§ Lec 3. Inverted Files Index.

Solution:

1) scan string ✗

2) Term-Document Incidence Matrix

e.g Document sets. ✗.

1. Compact Version - Inverteed File Index.

Def 1.1. Index is a mechanism for locating a given term in a text.

Def 1.2. Inverted File contains a list of pointers (e.g. the No. of the page) to all occurrences of that term in the text.

No	Term	Times; Documents Words.
10	silver	<2; $1_2.3_3$, ...
11	truck	<3; $1_3.3_8.4_7$>.

1.1. Index Generator.

for all document D do
  for all ~~doc~~ term T in D do.
    if T doesn't exist in D then
      insert T in D.
    Insert a node to T's posting list.

1.2 while reading a term.

1) Word Stemming: Process a word so that only its stem or root form is left.

2) Stop words: a. the. to be or not to be.

So common that it is useless to index them.

1.3 While accessing a term.

Solution: { Search Trees $O(\log n)$, data✗.
Hashing. $O(1)$. NO "✗"

1.4. While not having enough memory.

Memory in blocks, and merge them.

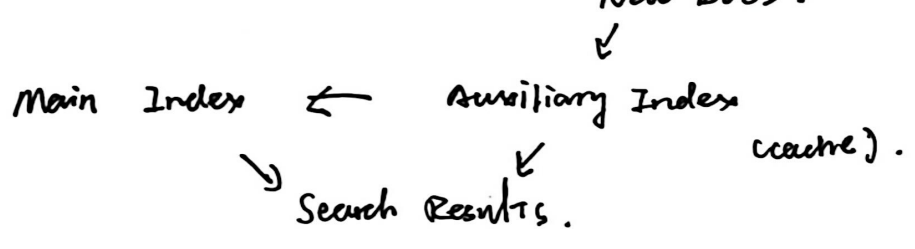
Distributed indexing — each nodes contains index of a subset of collection.

Solution: { Term-partitioned index.
Document-partitioned index. (Robust).

1.5. Dynamic indexing.

Docs come in over time. Docs get deleted

- postings updates.
- new terms added to dict..



1.6. Compression

美劳压缩.

Term.

a
arrive → arrive damage deliver fire gold
damage ↑ ↑ ↑ ↑ ↑.
deliver
fire
gold
of
in

index.

computer → 2, 15: 47.
computer → 2, 13. 32

1.7. Thresholding.

Document: only retrieve the top x documents where the documents are ranked by weight.

Problems:

- Not feasible for Boolean queries.
- Can miss some relevant due to truncation.

Query: Sort the query terms by their freq in ascending order;
Search according to only some percentage of the original query terms.

T1 T2	T3 T4	T5 T6 T7	...
20%	40%	80%	

2. Measures for a search engine.

- How fast does it index
- How fast does it search.
- Expressiveness of query language.

2-1. Relevance measurement.

1) A benchmark document collection.
2) A benchmark suite of queries.
3) A binary assessment of either Relevant or Irrelevant for each query-doc pair.

	Relevant	Irrelevant.
Retrieved	a	b
Not Retrieved	c	d.

$$\text{Precision} = \frac{a}{a+b}$$

$$\text{Recall} = \frac{a}{a+c}$$

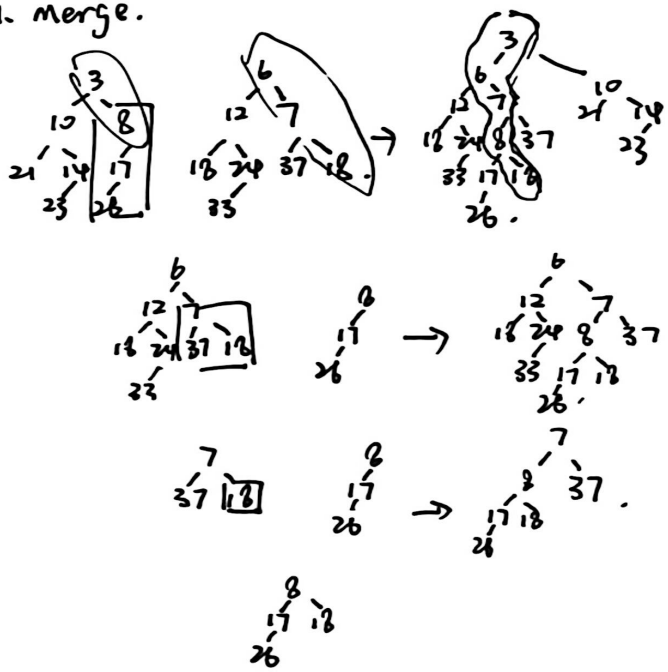
6 Lec4. Leftist Heaps and Skew Heaps

1. Leftist Heaps.

Heaps: Insert. Delete. GetMin.

+ Merge. 一般 O(N). Leftist Heaps O(log N).

1.1. Merge.



Recursion.

Merge:

保证 H1 根 H1 < H2.

O(log N).

1. Merge(H1→right, H2)
2. Attach.
3. If 左偏性质不满足了: Swap(H1→left, H1→right).
   (H1→left→npl < H2→left→npl).
4. H1→npl = H1→right→npl + 1.

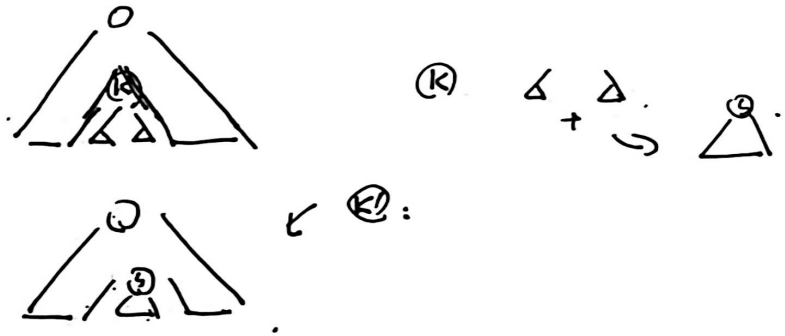
1.2. Insert.

只有一个节点的堆与另一个堆进行合并. O(log N).

1.3. DeleteMin

删除根节点，合并两个子堆. O(log N).

1.4. Decreasekey O(log N).



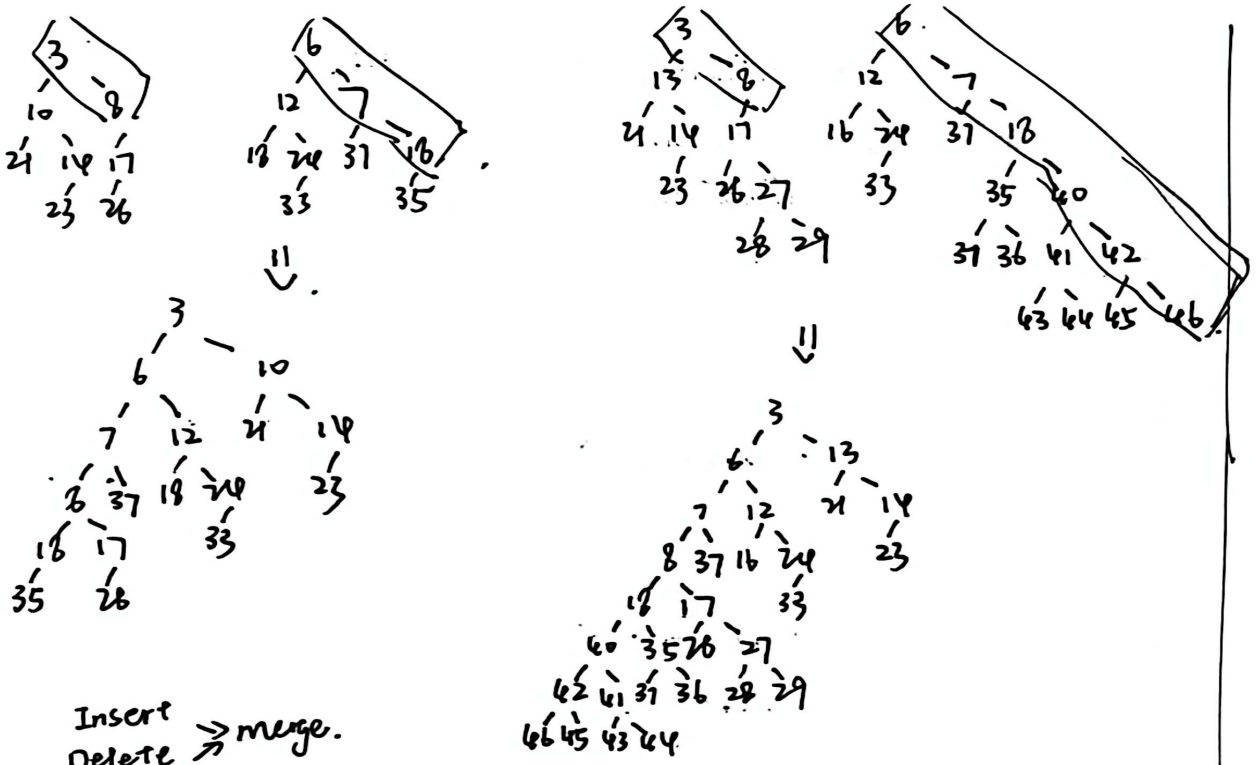
1.5. BuildHeap. O(N).

2. Skew Heaps.

Target: Any M consecutive operations take at most O(M log N) time.

Merge: Always swap

Except: the largest of all the nodes on the right paths doesn't have its children swapped.



Insert
Delete ⇒ merge.

Amortized Analysis. (merge).

A node p is heavy if ~~the number~~ 右子树的大小占整棵树的为 $\frac{1}{2}$
and light otherwise.

D_i = the root of the resulting tree.

$\phi(D_i)$ = number of heavy nodes.

只有在 right path 上的节点才会改变轻重性.

$H_i = l_i + h_i$ $i = 1, 2$. H_i 在第i路径的节点数.

$T_{worst} = h_1 + h_2 + l_1 + l_2$. $\phi_i = h_1 + h_2 + h$.
$\phi_{i+1} \le l_1 + l_2 + h$.

$T_{amortized} = T_{worst} + \phi_{i-1} - \phi_i \le 2(l_1 + l_2) = O(\log N)$.

② $L = O(\log N)$.

Proof. 即让 right path 上有 L 个 light 的时候为
有 $2^L - 1$ 个结点 (最差时).

§ Lec 5. Binomial Queue.

A binomial queue is a collection of heap-ordered trees. known as a forest.

Each heap-ordered tree is a binomial tree.

Binomial Tree:

B_0 B_1 B_2 B_3 ... B_K

B_{K-1} — B_K.

- B_K has 2^K nodes.
- # nodes at depth d is C_K^d.

A priority queue of any size can be uniquely represented by a collection of binomial trees.

(二进制).

1. operations.

1.1 FindMin

One of the roots. $O(\log N)$ or $O(1)$ by memo

1.2. Merge.

$O(\log N)$ =进制加法，合并是任意的.

Must keep the trees in the binomial queue sorted by height.

1.3. Insert.

Merge with ●.

If the smallest nonexistent binomial tree is B_i, then $T_p = \text{const}(i+1)$.

Worst case $O(\log N)$.

Amortized $O(1)$. (N continuous Insertion)

1.4. Delete Min.

1) FindMin in B_K $O(\log N)$

2) Remove B_K from H gets H' $O(1)$

3) Remove root from B_K gets H''. $O(\log N)$

4) Merge (H', H''). $O(\log N)$

$\Rightarrow O(\log N)$.

Lec 6. Backtracing.

1. Rationale of the Backtracing Algorithms.

The basic idea is that suppose we have a partial solution $(x_1 — x_i)$ where each $x_k \in S_k$ for $1 \le k \le i < n$.

First we add $x_{i+1} \in S_{i+1}$ and check if $(x_1 — x_i\ x_{i+1})$ satisfies the constraints. If the answer is "yes" we continue to add the next x. else we delete x_i and backtrack to the previous partial solution $(x_1 —— x_{i-1})$.

求决策点少的状态..

2. The Turnpike Reconstruction Problem.

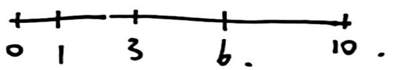
Given N points on the x-axis with words $x_1 < x_2 < — < x_N$. Assume that $x_1 = 0$.

There are $\frac{N(N-1)}{2}$ distances between pairs.

Prob: Given $\frac{N(N-1)}{2}$ distances.

Reconstruct a point set from the distances.

Eg. $\{1, 2, 3, 3, 4, 5, 6, 7, 9, 10\}$.



3. Tic-tac-toe: Minimax Strategy.

The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

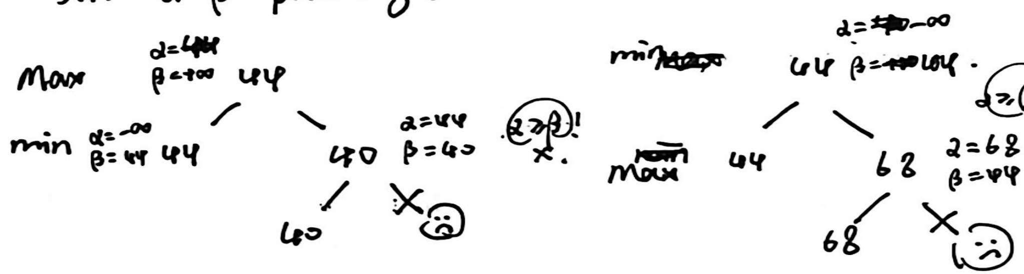
Use an evaluation function to quantify the "goodness" of a position. For example:

$$f(P) = W_{Computer} - W_{Human}$$

where W is the number of potential wins (在此种局面下让一方一直下可以得到的不同的胜利局面数) at position P.

The human is trying to minimize the value of the position P. The computer is trying to maximize.

3.1. α-β pruning.



§ Lec 7. Divide and Conquer.

Recursively:

1) Divide the problem into a sub-problems.
2) Conquer the sub-problems recursively.
3) Combine the solutions to the sub-problems into the solutions for the original problems.

General recurrence:

$$T(N) = aT(\frac{N}{b}) + f(N).$$

1. Closest points problem.

Naive: $O(N^2)$.

Divide and conquer:

$T(N) = 2T(\frac{N}{2}) + O(N)$.

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

2. Methods for solving recurrences

2.1 Substitution Method.

Guess, then prove by induction. $T(n) = \Theta(n)$ for small n.

Must prove the exact form.

2.2. Recursion-tree method.

$T(n) = \sqrt{n}\,T(\sqrt{n}) + O(n)$.

$T(n) = 2T(\sqrt{n}) + O(\log n)$.

$T(n) = n^{\frac{2}{3}} T(n^{\frac{1}{3}}) + O(n)$.

$T(n) = T(n^{\frac{1}{2}}) + T(n^{\frac{1}{3}}) + T(n^{\frac{1}{6}}) + \log n$.

$T(n) = \frac{1}{2}T(\frac{n}{2}) + \frac{1}{n}$.

2.3. Master method.

Theorem VII.1. (Master method I).

Let $a \geq 1$ and $b > 1$ be constants. $f(n)$ function.

$T(N) = aT(\frac{N}{b}) + f(N)$. Then.

1) If $\exists \varepsilon > 0.\ f(N) = O(N^{\log_b a - \varepsilon})$ (多项式比小于) 根 <0 时.

then $T(N) = \Theta(N^{\log_b a})$.

2) If $f(N) = \Theta(N^{\log_b a})$ (根=0 时

then $T(N) = \Theta(N^{\log_b a} \log N)$.

3) If $\exists \varepsilon > 0.\ f(N) = \Omega(N^{\log_b a + \varepsilon})$; (多项式比大于). 根>0 时.

and if $\exists n > 0, \forall N > n, \exists c < 1.\ af(\frac{N}{b}) < cf(N)$ (下层 当前层)

(regularity condition)

then $T(N) = \partial f(N)$.

Theorem VII.2. (Master Method II).

$$T(n) = aT(\frac{n}{b}) + f(n).$$

1) if $\exists k < 1.$ $\underbrace{af(\frac{n}{b})}_{\text{下层}} = \underbrace{kf(n)}_{\text{这一层}}$ 表格n倍

then $T(n) = \Theta(f(n))$.

2) if $f(n) = \Theta(n^{\log_b a})$.

then $T(n) = \Theta(n^{\log_b a} \log n)$.

3) if $\exists k > 1.$ $af(\frac{n}{b}) = kf(n)$ 等比n倍.

then $T(n) = \Theta(n^{\log_b a})$.

Theorem VII.3. (Master Method III).

$$T(n) = aT(\frac{n}{b}) + O(n^d \log^p n).$$

1) $a < b^d$:

$p \geq 0$: $T(n) = O(n^d \log^p n)$.

$p < 0$: $T(n) = O(n^d)$

2) $a = b^d$:

$p > -1$: $T(n) = O(n^d \log^{p+1} n)$.

$p = -1$: $T(n) = O(n^d \log\log n)$.

$p < -1$: $T(n) = O(n^d)$.

3) $a > b^d$:

$T(n) = O(n^{\log_b a})$.

Lec 8. Dynamic Programming.

An optimization problem suitable for Dynamic Programming (DP) should typically possess two key elements:

1. Optimal substructure 最优子结构.
2. Overlapping subproblems 子问题重叠.

设计 DP 四步:

1. 刻画一个最优解的结构特征.
2. 递归地定义最优解的值.
3. 计算最优解的值, 通常用底向上.
4. 利用计算出的信息构造一个最优解.

Solve sub-problems just once and save answers in a table. Use a table instead of recursion.

1. Fibonacci Numbers

$$F(n) = F(n-1) + F(n-2)$$

2. Ordering Matrix multiplications.

$M_1\ M_2\ M_3 \cdots M_n$.

$r_0\ r_1\ r_2\ r_3 \cdots r_{n-1}\ r_n$.

$$m_{ij} = \begin{cases} 0, & i = j \\ \min\limits_{i \le k < j} \{ m_{ik} + m_{(k+1)j} + r_{i-1} r_k r_j \}, & i < j \end{cases}$$

3. Optimal Binary Search Tree.

i	1	2	—	n
w_i	$w_1 < w_2 <$	—	$< w_n$	
p_i	p_1	p_2	—	p_n

$$c_{ij} = \sum_{k=i}^{j} p_k + \min_{i \le k \le j} \{ c_{i(k-1)} + c_{(k+1)j} \}, \quad i \le j.$$

$O(n^3)$. There exists a $O(n^2)$.

4. Floyd-Warshall Algorithm.

$D_{ij}^{(-1)} := e_{ij}$. $O(n^3)$.

$D_{ij}^{(k)} = \min \{ D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)} \}$, $k \ge 0 \le k < n$

5. WIS (one dimension).

打家劫舍.

$$dp_i = \max \{ dp_{i-1}, dp_{i-2} + w_i \}.$$

6. Knapsack.

0-1: $dp^k[c] = \min \{ dp^{k-1}[c], dp^{k-1}[c - w_k] + v_k \}$.

↖↑ 可优化至一维. 反向遍历 c.

Complete: $dp^k[c] = \min \{ dp^{k-1}[c], dp^k[c - w_k] + v_k \}$.

←↑ 可优化至一维. 正向遍历 c.

Bounded: 拆为 0-1 (补零 / 二进制).

7. DNA Similarity → 归约. $\delta = +\infty$.

序列对齐，最长公共子序列. 编辑距离. — a_i — b_j

~~dp[i][0]~~. $dp_{i0} = dp_{0j} = k\delta$.

$$dp_{ij} = \begin{cases} \min \begin{cases} dp_{i-1,j-1} + cost(a_i, b_j) \\ dp_{i,j-1} + \delta \\ dp_{i-1,j} + \delta \end{cases} \end{cases}$$

8. Weighted Activity Selection.

Recall: If no weight? Greedy.

$$dp_i = \max \begin{cases} dp_{i-1} \\ dp_{pre} + w_i \end{cases}$$ $O(N \log N)$.

dp_{pre} ~ $\log N$

Lec 9. Greedy Algorithm.

1. Intro

Greedy Algorithms.

Optimization problem.

constraints. optimization function.

满足 constraints 的为 feasible solutions.

其中优化函数值最优的为 optimal solution.

Greedy Method.

Heuristic. 局部最优 = 全局最优时正确.

Paradigm:

求解问题 → 做出一个选择后，剩下一个有待继续求解决的子问题.

需证明. { 贪心选择性质. / 最优子结构.
Optimal substructure.

2. Examples.

Activity Selection.

$S = \{a_1, a_2, \dots, a_n\}$. $a_i = [s_i, f_i)$.

求在不发生冲突的前提下能安排最多活动的方案.

DP:

$a_1 a_2 \cdots a_i \cdots a_{ik} \cdots a_j \cdots a_n$ (S_{ij}是开区间).

S_{ij}.

c_{ij} 为活动 a_i、a_j 中最多的活动数. 则.

$$c_{ij} = \begin{cases} 0. & \text{if } S_{ij} = \emptyset. \\ \max\limits_{a_k \in S_{ij}} \{c_{ik} + c_{kj} + 1\} & \text{if } S_{ij} \neq \emptyset. \end{cases}$$

或 $c_{ij} = \begin{cases} 0 \\ \max\{c_{i,j-1}, c_{i,k}+1\}. \end{cases}$ $O(N^3) \to O(N^2)$

$a_m(j)$ 为 i,j 间不冲最晚.

Greedy: 选最早结束的活动.

贪心选择性质: 考虑任意非空子问题 S_k. 令 a_m 为 S_k 中最早结束的活动. 则，a_m 包含在某个最优解中.

Proof: 设解集 A_k. a_{ef} 为 A_k 中最早结束的活动.

① 若 $a_m = a_{ef}$: 证毕.

② 若 $a_m \neq a_{ef}$: 替换 a_m 与 a_{ef}.

A_k 是另一最优解. (交换参考论证).

最优子结构: a_k + 子问题最优解 S_k 定为原问题最优解

$O(N) \sim O(N \log N)$

- 加权活动选择:

$a_i = [s_i, f_i)$. w_i.

$$c_i = \begin{cases} w_i. \\ \max\{w_i + c_{k(i)}, c_{i-1}\}. \end{cases}$$

DP 数组 $O(N \log N)$

区间相交问题：

判为

即活动可能少的数量举办所有活动。（最大重叠数）

将所有活动按照开始时间排序，从前往后遍历。

有空闲会场放。

Schedule Problems.

n个任务，每个任务 i 有一个完成时间 t_i 和权重 w_i。

T. i 的完成时间 $C_i(T)$ = (t_i + 比 i 之前的任务时长之和)。

$$\min \sum_{i=1}^{n} w_i C_i(T)$$

根据 $\frac{w_i}{t_i}$ 降序排序。

最小化最大延时：

按 d_i 顺序排序。

Huffman Codes.

Prefix code. 使用 Trie.（所有的字符在叶子节点）。

Huffman codes:

```
void Huffman( PriorityQueue. heap[], int C ){
    consider the C characters as C single node Binary_trees.
    and initialize them into a min heap.
    for i in range C.
        heap.push( heap.pop() + heap.pop() ).
```

T = O(ClogC)

§ Lec 10. NP-Completeness

1. Easy v.s. Hard

The easiest: $O(N)$ - since we read inputs

The hardest: undecidable.

2. The Class NP.

2.1 Turing Machine.

Task: To simulate any kind of computation, which a mathematician can do by some arithmetical method

Components: Infinite Memory and Scanner.

Operations.

1) Change the finite control state.
2) Erase the symbol in the unit currently pointed by head, and write a new symbol in.
3) Head moves one unit left(L)/right(R), or stays(S).

Deterministic Turing Machine: executes one instruction at each point in time. Then depending on the instruction, it goes to the next instruction.

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.

2.2 NP: Nondeterministic polynomial-time.

The problem is NP if we can prove any solution is true in polynomial time.

Note: Not all decidable problems are in NP.

$$P \subseteq NP.$$

2.3 NP-Complete Problems.

An NP-complete problem has the property that any problem in NP can be polynomially reduced to it.

Def.x.1. (Reduction).

$Y(>)$:

$O(n^k)$ standard computational steps.

$O(n^k)$ $\times$

Then we say $Y \leq_p X$.

Given $\forall$ instance α of problem β. if we can find a program $R(\alpha) \to \beta$ of Problem B with $T_R(N) = O(N^{k_1})$, and another program $D(\beta)$ to get an answer in time $O(N^{k_2})$.

2.3.1. Example.

We already know that the Hamiltonian cycle problem is NP-Complete.

Prove that the traveling salesman problem is NP-complete as well.

Proof 2.1.

1) TSP $\in$ NP.

2).

G G'

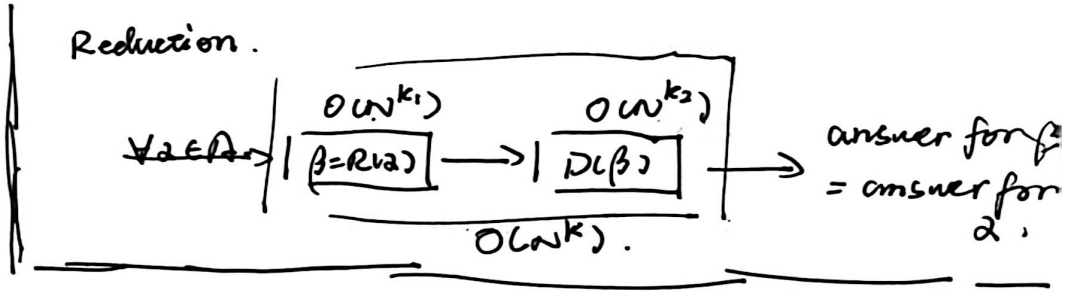
G has a Hamiltonian cycle iff.

G' has a traveling salesman tour of weight $|V|$.

3. A Formal-language framework.

3.1. Abstract Problem

An abstract problem Q is a binary relation on a set I of problem instances and a set S of problem solutions.



3.1.1. Example for SHORTEST-PATH problem

$I = \{ \langle G, u, v \rangle : G = (V, E)$ is an undirected graph; $u, v \in V \}$.

$S = \{ \langle u, w_1, w_2, \cdots, w_k, v \rangle : \langle u, w_1 \rangle, \langle w_1, \cdots \rangle, \langle w_k, v \rangle \in E \}$.

For every $i \in I$, SHORTEST-PATH$(i) = s \in S$.

For decision problem PATH:

$I = \{ \langle G, u, v, k \rangle : G = (V, E)$ is an undirected graph; $u, v \in V$; $k \geq 0$ is an integer $\}$.

$S = \{0, 1\}$.

3.2. Encodings.

Map I into a binary string $\{0,1\}^* \rightarrow Q$ is a concrete problem.

3.3. Formal-language Theory.

for decision problem Q.

1) An <u>alphabet</u> Σ is a finite set of symbols.

2) A <u>language</u> L over Σ is any set of strings made up of symbols from Σ ($L \subseteq \Sigma^*$).

3) Denote empty string by ε.

4) Denote empty language by $\emptyset$.

5) Language of all strings over Σ is denoted by Σ^*.

6) The complement of L is denoted by $\Sigma^* - L$.

7) The concatenation of two languages L_1 and L_2 is the language

$$L = \{x_1 x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}.$$

8) The closure or <u>Kleene star</u> of a language L is the language

$$L^* = \{\varepsilon\} \cup L \cup L_2 \cup L_3 \cup \cdots$$

where L_k is the language obtained by concatenating L to itself k times.

9) Algorithm A <u>accepts</u> a string $x \in \{0,1\}^*$ iff $A(x) = 1$

10) Algorithm A <u>rejects</u> a string x iff $A(x) = 0$.

11) A language L is <u>decided</u> by an algorithm A iff $\forall x \in L \; A(x) = 1$ and $\forall x \notin L \; A(x) = 0$.

12) To <u>accept</u> a language, an algorithm need only worry about strings in L, but to <u>decide</u> a language, it must correctly accept or reject every string in $\{0,1\}^*$.

$$P = \{L \subseteq \{0,1\}^* : D\}$$

where D means there exists an algorithm A that decides L in polynomial time.

13) A <u>verification algorithm</u> is a two-argument algorithm A, where one argument is an ordinary input string x and the other is a binary string y called a <u>certificate</u>.

14) A two-argument algorithm A <u>verifies</u> an input string x if there exists a certificate y such that $A(x,y) = 1$.

15) The <u>language</u> verified by a verification algorithm A is

$$L = \{x \in \{0,1\}^* : D\}$$

where D means there exists $y \in \{0,1\}^*$ such that $A(x,y) = 1$.

A language L belongs to NP iff there exists a two-input polynomial-time algorithm A and a constant c such that

$$L = \{x \in \{0,1\}^* : D\}$$

where D means 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

$$L \in NP.$$

(62) A language L_1 is polynomial-time reducible to a language L_2 ($L_1 \leq_p L_2$, L_1 no harder than L_2) if there exists a polynomial-time computable function

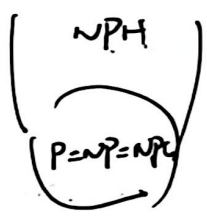
$$f: \{0,1\}^* \to \{0,1\}^*.$$

such that for all $x \in \{0,1\}^*$, $x \in L_1$ iff $f(x) \in L_2$. We call the function f the reduction function, and a polynomial-time algorithm F that computes f is called a reduction algorithm.

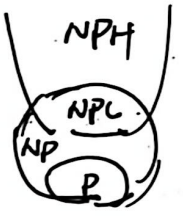
A language $L \subseteq \{0,1\}^*$ is NP-complete if

a. $L \in NP$, and

b. $\forall L' \in NP,\ L' \leq_p L$.



$P = NP$



$P \neq NP$

§. Lec 11. Approximation

Find near-optimal solutions in polynomial time.

1. Approximation Ratio

Def. XI.1. An algorithm has an approximation ratio of $\rho(n)$ if $\forall$ input of size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n)$$

If an algorithm achieves an approx ratio of $\rho(n)$, we call it a $\rho(n)$-approximation algorithm.

Def. XI.2. 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 $\forall$ fixed ε, the scheme is a $(1+\varepsilon)$-approx algo.

PTAS: $O(n^{f(\frac{1}{\varepsilon})})$

EPTAS: $O(n^k f(\frac{1}{\varepsilon}))$

FPTAS: $O(n^{k_1}(\frac{1}{\varepsilon})^{k_2})$.

TSP: $O(n^{\frac{1}{\varepsilon}})$

Independent Set: $O(4^n \cdot 2^{\frac{1}{\varepsilon}})$

Knapsack: $O(n^3 \frac{1}{\varepsilon})$

2. Approximate Bin Packing

Given N items of sizes $s_1, s_2, \dots, s_N$, such that $0 < s_i \le 1$ $\forall\ 1 \le i < N$. pack these items in the fewest number of bins, each of which has unit capacity. ($\rho \ge \frac{3}{2}$ unless P=NP.).

2.1 Next Fit.

Theorem XI.1. $NF \le 2M-1$. (Tight).

2.2. First Fit.

Theorem XI.2. $FF \le \frac{17(M-1)}{10}$ (Tight)

2.3. Best Fit. (place in the tightest).

Theorem XI.3.5. $BF \le \frac{17(M-1)}{10}$ (Tight).

But $O(N \log N)$

2.4. Online Algorithms

Place an item before processing the next one, and can't change decision.

Theorem XI.3. There are inputs that force any on-line bin-peecking algorithm to use at least $\frac{5}{3}$ the optimal number of bins.

Online $\le \frac{5}{3} M$

2.5. Offline Algorithms.

View the entire item list before producing an answer.

Theorem XI.4.

$$FFD \le \frac{11M+6}{9}.$$

NF: 2.

Any Fit (AF)

- First Fit (FF)
- Best Fit (BF)
- Worst Fit (WF) : 2
- Almost Worst Fit (AWF)

Almost Any Fit (AAF. 1.7).

3. The Knapsack Problem.

3.1 fractional version.

Greedy = OPT. by $\frac{p_i}{w_i}$.

3.2. 0-1 version.

Greedy on p_i

Greedy on $\frac{v_i}{w_i}$

} arbitrarily bad!

$$\max\begin{cases}\text{Greedy on } p_i\\ \text{Greedy on } \frac{p_i}{w_i}\end{cases} \Rightarrow \rho = 2.$$

~~3.3. FPTAS~~

Proof. $p_{max} \le P_{greedy} \le P_{OPT} \le P_{frac}$.

$$\Rightarrow \rho = \frac{P_{OPT}}{P_{greedy}} \le \frac{P_{frac}}{P_{greedy}} \le \frac{P_{greedy}+p_{max}}{P_{greedy}} = 1 + \frac{p_{max}}{P_{greedy}} \le 2.$$

3.3. FPTAS Algorithm.

prob: minimize $\sum w_i x_i$

under the constrains $\sum p_i x_i \ge P$.

$$dp_p^k = \min\begin{cases} +\infty \\ w_p^{k-1} \\ w_{p-p_k}^{k-1} + w_k. \end{cases}$$

$O(n^2 p_{max})$.

A FPTAS:

1. Given ε. $b = \frac{\varepsilon p_{max}}{n}$.
2. $p_i \to \lceil \frac{p_i}{b} \rceil$. $DP_1 \to p$.
3. $p_i' = \lceil \frac{p_i}{b} \rceil b$. bp is the approx solution

$(1+\varepsilon)$- approximation. $O(\frac{n^3}{\varepsilon})$.

K-center 与 Lec11. Approximation.

2.2. K-center Problems.

Clustering.

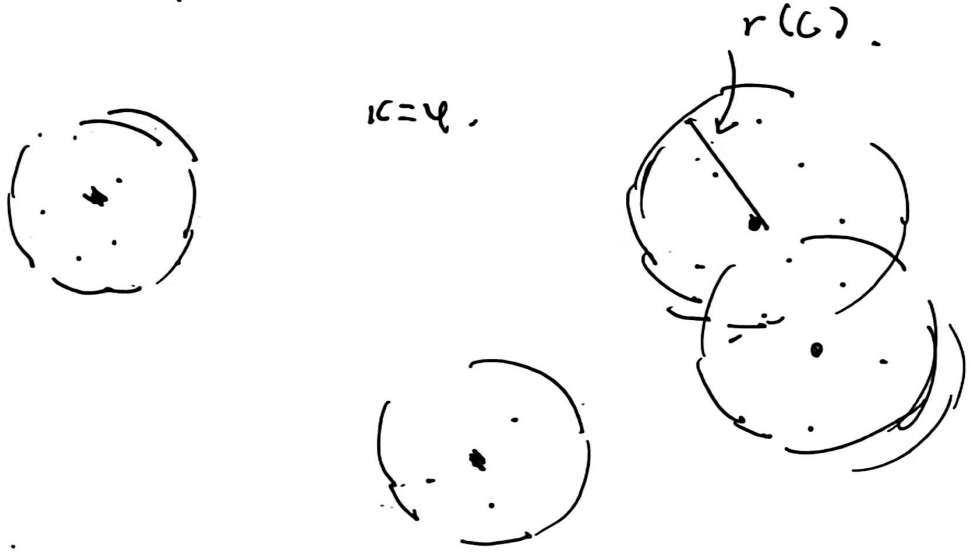
s_1 — s_n. 选择 K 个中心点 C. 使最小化:

任意把地到离它距离最近的中心点之间的距离的最大值.

令:

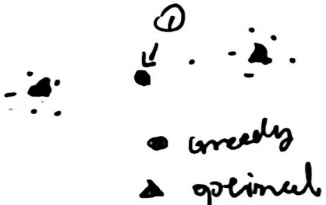
$$dist(s_i, C) = \min_{c \in C} \{dist(s_i, c)\}.$$

$$r(C) = \max_i \{dist(s_i, C)\}.$$ minimize r(C).



2.2.1. Naive Greedy.

:(: k=2:



让第一个中心点作为所有地址的中点. 随后加入的中心点每次成为 r(C) 最大的点.

2.2.2. 2r Greedy

```
Centers Greedy-2r(Sites S[], int n, int K, dou r){
    Sites S'[] = S[]; // S': the remaining sites.
    Centers C[] = NULL;
    while (S'[] != NULL){
        Select any s from S' and add it to C;
        Delete all s' from S' that are not
            dist(s', s) <= 2r;
    }
    if (|C| <= K) return C;
    else No solution
}.
```

Greedy-2r 算法在给定最优解 r 的情况下是一个 2-approx 算法.

(只需证明其可在 K 步内停止).

如果不停止. 则最优解大于 r. 二分查找.

2.2.3. Smarter Greedy

除非 P=NP. 不存在 ρ<2 的近似算法.

```
Centers Greedy-Kcenter(Sites S[],
                       int n, int K)
```

4.3. Smarter Greedy.

```
centers Greedy-Kcenter( Sites S[], int n, int K){
    Centers C[] = NULL;
    Select any s from S and add it to C;
    while (|C| ≤ K){
        Select s from S with maximum dist(s, C);
        Add s to C;
    }
    return C;
}.
```

证明：假设不正确，即存在一个 s，$dist(s, C) > 2r$。

则有 $dist(c, c') \geq dist(s, c') \geq dist(s, c) > 2r$。

即 Greedy-Kcenter 是另一种 Greedy-$2r$。

Greedy-$2r$ 不结束，说明最优解 $> r$，矛盾。

4.4. optimal ρ.

除非 P=NP，否则 K-center 问题不存在 ρ-近似算法 ($\rho < 2$)。

Proof:

Dominating Set problem:

$G = (V, E)$，K，判断是否有大小为 K 的集合 $S \subseteq V$，使每个点要么在 S 中，要么与 S 相邻。 (NPC)。

Dominating Set Problem $\leq_p$ K-center.

ρ-approx. ($\rho < 2$)

将 G 中相邻点间距离改为 1，不相邻为 2。

1、K-center 问题解只为 1 或 2。

2、如有半径为 1 的解，则 DSP 有解，反之无解。

设有 ρ-approx ($\rho < 2$)，则：

算法返回 $r < 2$ 的解时 $\Rightarrow$ K-center 有半径为 1。

反之 半径为 2。

归约成功。

5. Traveling Salesman Problem.

Theorem. 除非 P=NP，否则对于任意的 $k \geq 1$，

TSP 不存在 k-近似算法。

Proof. 归约到 Hamiltonian cycle.

$$w(e) = \begin{cases} 1, & e \in E. \\ 2+(k-1)n, & e \notin E. \end{cases}$$

但考虑三角不等式 $w(u, v) \leq w(u, x) + w(x, v)$。

(度量 TSP 问题)。则：

Theorem. 存在如下 2-近似算法：首先求出图 G 的最小生成树 T，在 T 上先序遍历得到一条哈密顿回路。

Lec 12. Local Search

Solve problems approximately → aims at a local optimum.

Framework of Local Search:

Local:

Define neighborhoods in the feasible set.

A local optimum is a best solution in a neighborhood.

Search:

Start with a feasible solution and search a better one within the neighborhood.

A local optimum is achieved if no improvement is possible.

Neighbor Relation:

$S \sim S'$: S' is a neighboring solution of S — S' can be obtained by a small modification of S.

$N(S)$: neighborhood of S — the set $\{S' : S \sim S'\}$.

1. The Vertex Cover Problem

$G=(V,E)$: Find a minimum subset S of V such that for each edge (u,v) in E, either u or v is in S.

Feasible solution set FS: all the vertex covers.

$cost(S) = |S|$

$S \sim S'$: Each vertex cover S has at most $|V|$ neighbors.

Search: Start from $S = V$; delete a node and check if S' is a vertex cover with a smaller cost.

(Arbitrarily bad!).

Greedy 2-approx:

(u,v). 将 u, v 同时加入到 C 中，然后把 u 和 v 所在的所有边全部移除，重复操作.

Proof: $|C^*| \geq |S|$

$|C| = 2|S| \leq 2|C^*|$.

2. Simulated Annealing.

The material is cooled very gradually from a high temperature, allowing it enough time to reach equilibrium at a succession of intermediate lower temp.

3. Hopfield Neural Networks

$G=(V,E)$ with integer edge weights $w(I)$.

- If $w_e<0$, where $e=(u,v)$, then

  u and v want to have the same state(I).

- If $w_e>0$. then u and v want different states.

The absolute value $|w_e|$ indicates the strength of this requirement.

Find a sufficiently good configuration S of the network:
— an assignment of the state s_u to each node u.

Def 3.1. In a config S, edge $e=(u,v)$ is good if

$$w_e s_u s_v < 0.$$

otherwise it is bad.

Def 3.2. In a config S, a node u is satisfied if the weight of incident good edges $\geq$ weight of incident bad edges.

$$\sum_{\substack{u,v: \\ e=(u,v)\in E}} w_e s_u s_v \leq 0.$$

Def 3.3: A config is stable if all nodes are satisfied.

3.1. State-flipping Algorithm.

```
ConfigType StateFlipping() {
    Start from an arbitrary config S;
    while(!IsStable(S)) {
        u = GetUnsatisfied(S);
        s_u = -s_u;
    }
    return S;
}
```

Claim 3.1. The state-flipping algorithm terminates at a stable configuration after at most

$$W=\sum_e |w_e| \quad \text{iterations.}$$

Proof: $\phi(S)=\sum_{e\in S} |w_e|$

Claim 3.2. Any local minimum in the state-flipping algorithm to minimize ϕ is a stable configuration

Proof: by contrapositive.

Related to Local search:

Problem: To minimize ϕ.

Feasible solution set Fs: configurations.

$S \sim S'$: S' obtained from S by flipping a single state.

4. The Maximum Cut Problem.

$G=(V,E)$ with $w_e > 0$,

find a node partition (A,B), maximize

$$w(A,B) := \sum_{u \in A, v \in B} w(u,v).$$

Related to Local Search:

Problem: To maximize $\phi(S) = \sum_{e \text{ is good}} |w_e|$.

Feasible solution set FS: any partition (A,B).

$S \sim S'$: S' can be obtained from S by moving one node from A to B, or one from B to A.

Claim 4.1. $\frac{w(A,B)}{w(A^*,B^*)} \geq \frac{1}{2}$.

Unless $P=NP$, No $\frac{17}{16}$-approx algo for MAX-CUT.

4.1. Big-improvement flip

Only choose a node which, when flipped, increases the cut value by at least,

$$\left\lfloor \frac{2\varepsilon}{|V|} w(A,B) \right\rfloor$$

Claim 4.2. $\frac{w(A,B)}{w(A^*,B^*)} \geq \frac{1}{2+\varepsilon}$.

$(1+\frac{1}{x})^x \geq 2$, so the objective function doubles at least every $\frac{n}{\varepsilon}$ flips.

Claim 4.3. It terminates after at most

$$O(\tfrac{n}{\varepsilon} \log W) \quad \text{flips.}$$

4.2. K-L heuristic

A better local, k-flip.

Step 1. make 1-flip as good as we can

$O(n) \Rightarrow (A_1, B_1)$ and v_1.

Step k. make 1-flip of an unmarked node as good as we can

$O(n-k+1) \Rightarrow (A_k, B_k)$ and $v_1 - v_k$.

Step n $(A_n, B_n) = (B, A)$.

Neighborhood of (A,B) =

$\{(A_1, B_1), \ldots, (A_{n-1}, B_{n-1})\}$.

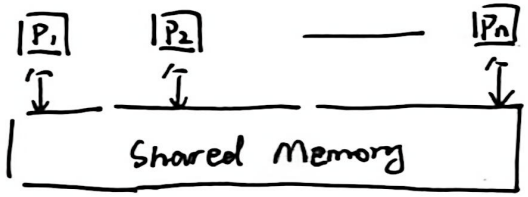
$O(n^2)$.

§14. Parallel Algorithm.

To describe a parallel algorithm.

- Parallel Random Access Machine (PRAM)
- Work-Depth (WD).

1. PRAM.



for P_i. $1 \le i \le n$ pardo.

B(i) := B(i)

1-1 To resolve access conflicts

Exclusive-Read	Exclusive-Write	EREW
Concurrent-Read	Concurrent-Write	CRCW
concurrent-Read	~~Concurrent~~ Exclusive-Write	CREW

Arbitrary rule

Priority rule (if with the smallest number).

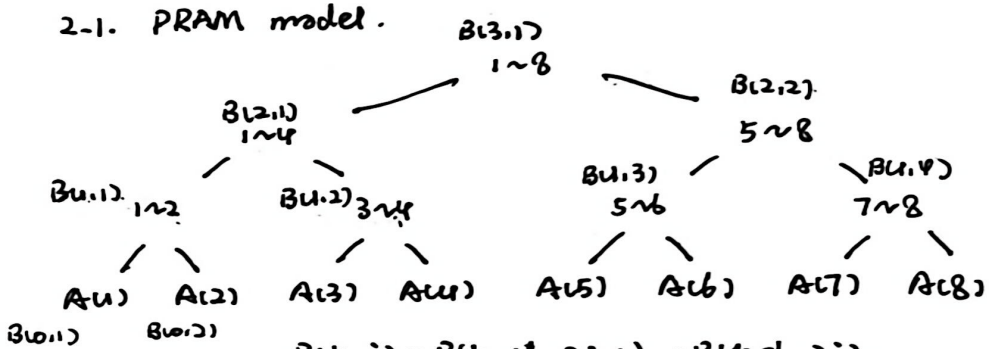
Common rule (if all the processors are trying to write the same value).

2. The summation problem

Input: A(1), A(2), ---, A(n)

Output: A(1) + A(2) + --- + A(n)

2-1. PRAM model.



B(h,i) = B(h-1, 2i-1) + B(h-1, 2i).

```
for(P_i, 1≤i≤n) pardo{
    B(0,i) = A(i)
    for(h=1 to log n){
      if(i ≤ n/2^h)
        B(h,i) := B(h-1,2i-1) + B(h-1,2i);
      else idle
    }
}
```

T(n) = log n + 2.

2-2 Work-Depth (WD) Presentation.

```
for(Pi, 1≤i≤n) pardo{
    B(0,i) := A(i);
for(h=1 to log n)
    for(Pi, 1≤i≤ n/2^h) pardo{
        B(h,i) := B(h-1,2i-1) + B(h-1,2i);
}
```

3. Measuring the performance.

- Work-load - total number of operations: $W(n)$
- Worst-case running time: $T(n)$

All asymptotically equivalent:

1) $W(n)$ operations and $T(n)$ time
2) $P(n) = \frac{W(n)}{T(n)}$ processors and $T(n)$ time (PRAM)
3) $\frac{W(n)}{p}$ time using any number of $p \le \frac{W(n)}{T(n)}$ processors (on a PRAM)
4) $\frac{W(n)}{p} + T(n)$ time using any number of p processors (on a PRAM).

4. Prefix-Sums

Input: $A(1), A(2), \ldots, A(n)$.

Output: $\sum_{i=1}^{1} A(i), \sum_{i=1}^{2} A(i), \ldots, \sum_{i=1}^{n} A(i)$.

Technique: Balanced Binary Trees.

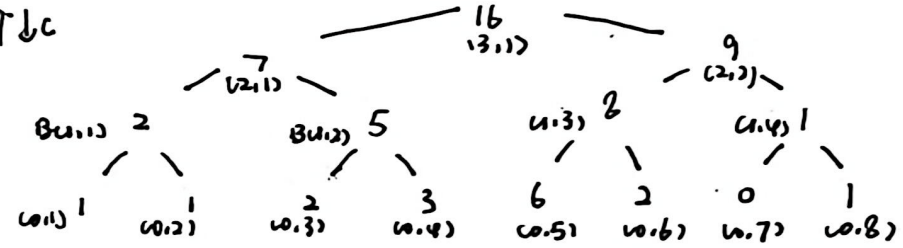
$$C(h,i) = \sum_{k=1}^{2^h} A(k).$$

where $(0,2)$ is the rightmost descendant leaf of node (h,i).

$$C(h,i) = \begin{cases} B(h,i) & i = 1 \\ C(h+1, \frac{i}{2}) & i \equiv 0 \bmod 2 \\ C(h+1, \frac{i-1}{2}) + B(h,i) & i \equiv 1 \bmod 2 \end{cases}$$

(left path)
(right child)
(left child)

B↑ C



5. Merging

Merge $A(1), A(2), \ldots, A(n)$ and $B(1), B(2), \ldots, B(n)$ (sorted) to $C(1), C(2), \ldots, C(2n)$ (sorted).

serial ~~contiguous~~: $O(n)$.

5.1. Parallel Ranking.

The ranking problem.

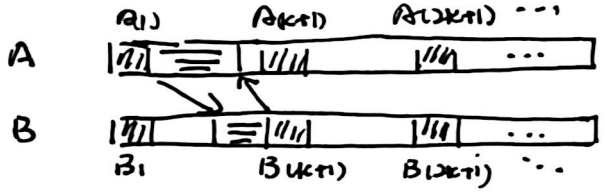
1) RANK(i, B) $1 \le i \le n$.
2) RANK(i, A) $1 \le i \le n$.

Binary Search:
$W = O(n \log n)$
$D = O(\log n)$

Claim 5.1. Given a solution to the ranking problem, the merging problem can be solved in $O(1)$ time and $O(n+m)$ work.

```
for (Pi, 1 ≤ i ≤ n) pardo
    C(i + RANK(i, B)) := A(i)
for (Pi, 1 ≤ i ≤ n) pardo
    C(i + RANK(i, A)) := B(i).
```

Partition Paradigm



$\frac{n}{k}$ partitions.

1. use binary search to determine the rank of the selected $\frac{2n}{k}$ elements.

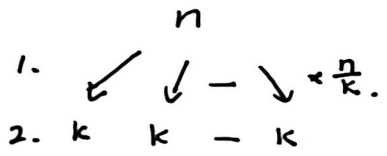
$W_1 = O(\log n) \cdot \frac{2n}{k} = O(\frac{n \log n}{k})$

$D_1 = O(\log n)$

2. determine the rank of elements with the same color using serial ranking.

$W_2 = O(k) \cdot O(\frac{n}{k}) = O(n)$

$D_2 = O(k)$.



	W	D
1	$O(\frac{n \log n}{k})$	$O(\log n)$
2	$O(n)$	$O(k)$

~~Tnext~~

取 $k = \log n$. $W(n) = O(n) \cdot D(n) = O(\log n)$.

6. Maximum Finding

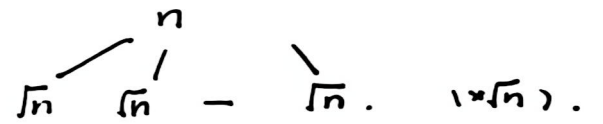
Input A_1 — A_n

Output $\max_i A_i$.

	T	W	
Serial	$O(n)$	$O(n)$	
Binary Tree	$O(\log n)$	$O(n)$	
Parallel	$O(1)$	$O(n^2)$	(CRCW common).
Doubly-log	$O(\log\log n)$	$O(n \log\log n)$.	

6.1. A Doubly-logarithmic Algorithm.

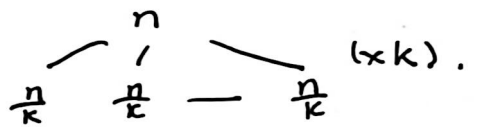
(Divide and Conquer).



1. partition into $\sqrt{n} \times \sqrt{n}$.
2. recursively find the maximum in each.
3. find the maximum among $\sqrt{n}$ elements parallelly

$$\begin{cases} T(n) = T(\sqrt{n}) + 1 \\ W(n) = \sqrt{n}\, W(\sqrt{n}) + n \end{cases} \Rightarrow \begin{cases} T(n) = O(\log\log n) \\ W(n) = O(n \log\log n). \end{cases}$$

6.2. Accelerating Cascades Paradigm.

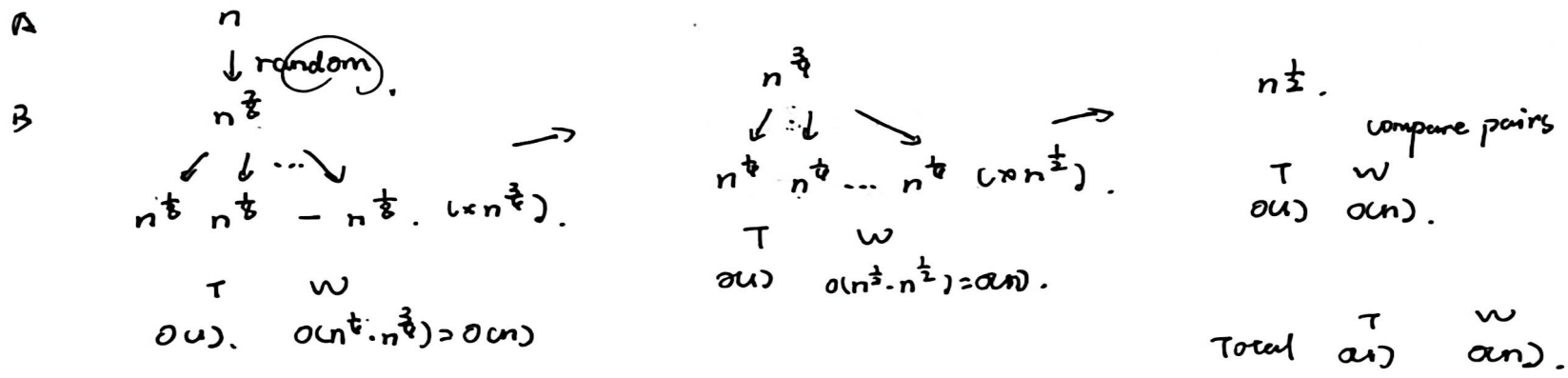


1. solve each group serially.
2. find the maximum of the k elements doubly-logarithmically.

	T	W
1	$O(\frac{n}{k})$	$O(n)$
2	$O(\log\log k)$	$O(k \log\log k)$.
$k = O(\frac{n}{\log\log n})$		
Total	$O(\log\log n)$	$O(n)$.

6.3 Random Sampling.

with very high probability. on an Arbitrary CREW PRAM. $T = O(1)$ $W = O(n)$ $p = 1 - \frac{1}{n^c}$.



§ Lec 15. External Sorting.

Target: Sort N records where $N \gg M$ (Memory Size)

Constraint: Disk I/O is the bottleneck. ($a[i] \neq o(1)$).

Access 要 is sequential and slow.

Cost: Number of Passes (reading and writing the data once)

Tool: Mergesort.

To simplify:

1) Store data on tapes. (can only be accessed sequentially)

2) Can use at least 3 tape drives.

1. Simple External Merge Sort

Suppose $M = 3$.

T_1: 81 94 11 96 12 35 17 99 28 58 41 75 15

IM: 12 35 96

① T_2: 11 81 94 17 28 99 15

T_3: 12 35 96 41 58 75

② T_1: 11 12 35 81 94 96 15

T_4: 17 28 41 58 75 99

③ T_2: 11 12 17 28 35 41 58 81 75 94 96 99

T_3: 15

#passes $= 1 + \lceil \log_2 \frac{N}{M} \rceil$.

2. Multi-way Merge.

Target: Reduce the number of passes.

Method: k-way merge. Merge k runs into 1.

2.1. Complexity

passes $= 1 + \lceil \log_k \frac{N}{M} \rceil$.

tapes $= 2k$.

Internal Work:

Finding the minimum among k elements takes $O(k)$ or $O(\log k)$ (with Heap / Tournament Tree)

Internal time: $O(N \log_2 k \cdot \log_k \frac{N}{M}) = O(N \log \frac{N}{M})$.

3. Polyphase Merge

Problem: 2k tapes.

Solution: Use k+1 tapes for k-way merge, by polymergeSort.

	#1	1'	2	3	4	5	6	7	8
T_1	34		13	5		3	1		
T_2		21	8		5	2		1	
T_3		13		8	3		2	1	

$k=2$: $F_N = F_{N-1} + F_{N-2}$. $F_0 = 0$. $F_1 = 1$.

$k=k$: $F_N^{(k)} = F_{N-1}^{(k)} + \cdots + F_{N-k}^{(k)}$. $F_{\text{其他}}^{(k)} = 0$ ($0 \le i \le k-2$). $F_{k-1}^{(k)} =$

4. Replacement selection.

Target: Generate initial runs longer than M.

Average Run Length: 2M.

4.1. Algorithm

Use a Min-Heap in memory.

1. Fill memory with elements. Build min-heap.
2. Min = DeleteMin(Heap).
3. Output Min.
4. Read Next from input.
5. If Next ≥ Min:
6. Insert Next into Heap.
7. If Next < Min:
8. Place Next at the dead space.
9. Decrease Heap size.
10. Repeat until Heap is empty.
11. Rebuild Heap with dead elements and start new Run.

k↑ ⇒, # input buffers ↑,
buffer size ↓, block size on disk ↓,
seek time ↑.

4.2. Analysis.

Best Case: N.

Worst Case: M.

Average Case: 2M.

Eg. 81 94 11 96 12 35 17 99 28
58 41 75 15.

T_1	11 81 94 96
T_2	12 17 28 35 41 58 75 99
T_3	15

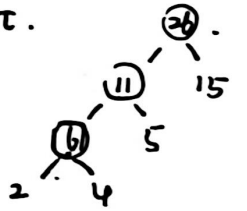
5. Minimizing Merge Time (Huffman).

Problem: In what order should we merge them to minimize total record movements?

Solution: Greedy: Huffman coding.

1) merge the two smallest first.

Total Cost: O(Weighted ~~Path Length~~ External Path Length)



6. ~~Pre~~ Buffering: Parallelism.

Double Buffering:

Use 2 ~~pp~~ input buffers and 2 output buffs

While CPU processes one, I/O the other.