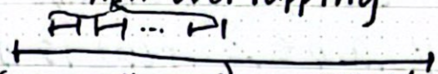
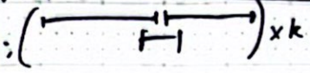
Greedy Algorithms

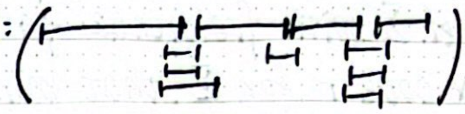
Activity selection (Interval Schedualing)

Input: a set of activities $(s_1, f_1) \dots (s_n, f_n)$

Output: a maximum set of mutually $\underbrace{\text{compatible}}_{\text{non-overlapping}}$ activities

rule1. smallest start time 反例:  OPT=k Greedy=1

rule2. shortest interval 反例:  $\times k$ OPT=2k Greedy=k

rule3. fewest conflict 反例:  $\times k$ OPT=4k Greedy=3k

rule4: earliest finish time ✓

1. let R be the set of activities } O(1)
2. Let $A = \emptyset$
3. Sort R by finish time $O(n\log n)$
4. for ~~for~~ activity $i \in R$ $\times n$
5. if i is compatible with A 只需A最后结束时间和s_i比. O(1).
6. add i to A
7. return A. $\Rightarrow T(N) = O(N\log N)$

→ Theorem: A is optimal

Proof: (反证法) suppose A is not optimal $|OPT| > |A|$

A: $i_1, i_2, \dots, i_k$

OPT: $j_1, j_2, \dots j_k, j_{k+1} \dots j_m$. exchange argument ↴ 把最优解转换成贪心解形式找矛盾

⇓

$i_1 \dots i_{t-1} \quad i_t \quad i_{t+1} \dots$

$\| \quad\quad \| \quad\quad \ne$

$j_1 \dots j_{t-1} \quad j_t \quad j_{t+1} \dots$

↳ 若替换成 i_t，是否会冲突？

由 greedy $\Rightarrow f_{i_t} \le f_{j_t}$ 不会冲突

↙替换后

OPT': $i_1, i_2, \dots, \underbrace{i_k, j_{k+1}}, \dots j_m$

↳ Greedy 时未放入 j_{k+1}, conflict.

Data Compression

* prefix: for a string $s = a_1 \cdots a_n$, for $i = 0 \ldots n$, $a_1 a_2 \ldots a_i$ is a prefix of s.

⇒ 把不存在前缀的编码称为 prefix code.

A prefix (free) code for an alphabet Σ is a function $r: \Sigma \to \{0,1\}^*$ such that for any $a, b \in \Sigma$, $r(a)$ is not a prefix of $r(b)$

Input: An alphabet Σ with frequence f_a for each $a \in \Sigma$. Assume $|\Sigma| \geq 2$.

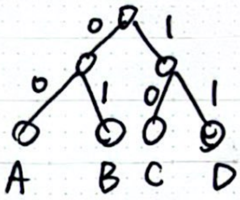
Output: a prefix code r for Σ that minimizes $\sum_{a\in\Sigma} |r(a)| \cdot f(a)$

code ⇔ binary tree

例:

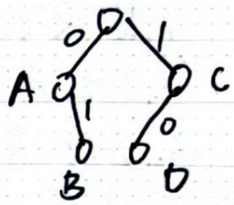
A 00
B 01
C 10
D 11

⇔



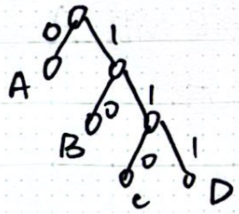
A 0
B 01
C 10
D 1

⇔



A 0
B 10
C 110
D 111

⇔



同时可以当解码器.
(读到0往左, 1往右, 直到读到叶节点)

prefix code r ⇔ a binary tree where only leaves are labelled by distinc symbo

Σ-tree

$\sum_{a\in\Sigma} |r(a)| f(a)$ ⇔ $\sum_{a\in\Sigma} \text{depth}(a) \cdot f(a)$

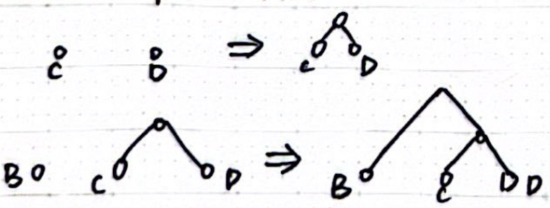
⇓ 找编码问题转变成找树

Input: Σ and $\{f_a\}_{a\in\Sigma}$

Output: a Σ-tree that minimizes $\sum_{a\in\Sigma} \text{depth}(a) \cdot f(a) = C(T)$

↓

bottom-up construction

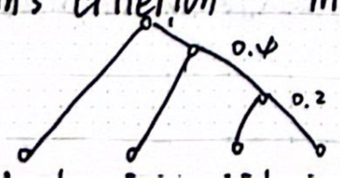


$\Delta = f_C + f_D$

$\Delta = f_B + f_C + f_D$

Huffman's criterion – minimum increasement in the average leaf depth

e.g.



Huffman's Algorithm

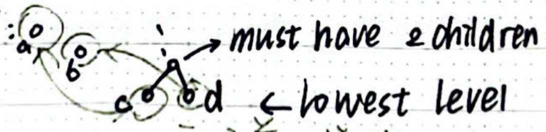
```
for each a ∈ Σ
    create a tree Ta of a single node  O a
    f(Ta) = fa
let F be the set of all trees
while |F| ≥ 2
    let T1 and T2 be the two trees with minimum freq.
    F := F - {T1, T2}
    T3 := merge (T1, T2)
    f(T3) := f(T1) + f(T2)
    add T3 to F
return the tree remaining in F.
```

下证这个贪心算法是最优的:

Lemma: Let a, b be the symbols with min. freq. There is an optimal tree in ~~tree~~ which a and b are siblings.

Proof ~~(by contradiction)~~

assume OPT:



反证法易证

和之前一样的 exchange 的贪心算的思想

$f_a \le f_c$, $f_b \le f_d$. $\Rightarrow$ 换完后还是最优解.

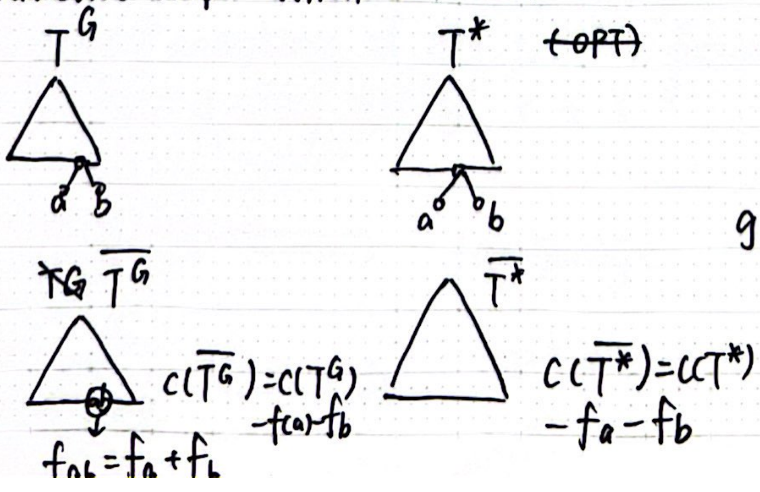
Theorem: Huffman's algorithm gives an optimal Σ-tree.

Proof: by induction on $|\Sigma|$.

base case: $|\Sigma| = 2$. optimal (就是 0 和 1)

inductive hypothesis: assume when $|\Sigma| = k$, optimal.

inductive step: when $|\Sigma| = k+1$



goal: $C(T^G) \le C(T^*)$

$\downarrow$

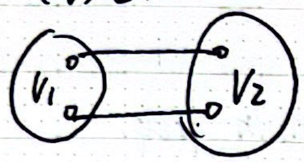
$C(\overline{T^G}) \le C(\overline{T^*})$

T^G is Σ-tree for $\Sigma' = \Sigma - \{a,b\} + \{ab\} \curvearrowright$ metasymbol

$|\Sigma'| = |\Sigma| - 1 = k \Rightarrow$ 归结到 Inductive hypothesis. 结合LEMMA可证

Prim's Algorithm 最小生成证明.

$G = (V, E)$

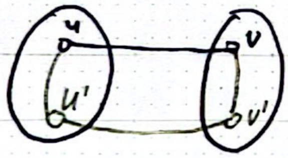


$(V_1, V_2) \rightarrow$ cut

$\delta(V_1, V_2) \rightarrow$ cut set　割边集

(assume distinct edge weight) For any cut (V_1, V_2) of G, the min edge in $\delta(V_1, V_2)$ must be chosen by the MST.

反证法：假设OPT 为T^*：(不包括(u,v) 但 (v,u) 是这个割边集的最短边)



也是在作交换把最优解变成贪心解.

$T^* \cup \{(u,v)\} - \{(u',v')\}$ 更小 $\Rightarrow$ 与OPT矛盾.

$\rightarrow$ Prim's Algorithm 每次选的边都是一种割里最小的边.

若边权有一样作轻微扰动即可

NP - Completeness

hardness complexity

sum —— $O(1)$

hardest: incomputable (undecidable)

→ Halting Problem: Given a program P and an input X, does P halt on X?

Assume ∃ program Halt: X → 不存在 ⇒ 不可解

$$\text{Halt}(P, x) = \begin{cases} \text{yes}, & \text{if } P \text{ halts on } x \\ \text{no}, & \text{otherwise} \end{cases}$$

(对角化证明)

Diagonal(P): 1. if Halt(P, P)

2. go to step 1

Diagonal halts on P if and only if P loops on P

let P = Diagonal ⇒ Contradiction

problem
- incomputable
- computable
  - complexity class
  - P, NP, EXP
  - PSPACE, co-NP, RP ...

1. Given a weighted graph G, and vertices s and t, what is the shortest path from s to t?
2. .. what is the length of ... ?
3. Given G, s, t and integer k, is there a path from s to t whose length $\leq k$?

└ decision problem yes or no

难度: $1 \geq 2 \geq 3$ (能解决1则能解决2和3).

-使用问题3 + 二分 ($\log_2 \Sigma$ ~~dl~~ length) 可解决2 ⇒ $3 \geq 2$

-使用对G删边看问题2 ~~能否解~~ 是否会变化. 可解1 ⇒ $2 \geq 1$

} $1 = 2 = 3$

$\langle G, s, t, k\rangle \xrightarrow{\text{encode}}$ binary string

定义 X = {encodings of $\langle G, s, t, k\rangle$ for ~~wh~~ which the answer is yes}

↔ Given a string s, is $s \in X$?

↓ language

decision problem ⟷ language

instance $\langle G, s, t, k\rangle$ ⟷ string s

把判定问题抽象为集合.

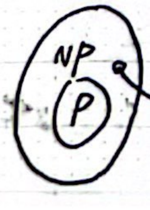
An algorithm A is a program when given a string s, return yes or no → A(s)
An algorithm A solves a problem X, if for any string s, A(s) = yes if and only if $s \in X$.
An algorithm A has a polynomial running time if there is a polynomial function p( ) so that for every string s, A terminates on s within p(|s|) steps.
P is the set of all problems for which there exists a polynomial time algorithm.

satisfiability problem (SAT) 通过给布尔函数赋值使结果为真
$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \Rightarrow$ hint $(x_1=1, x_2=1, x_3=0, x_4=1)$
We say B is an efficient verifier for problem x if:
(1) B is a polynomial-time algorithm that takes two argument s and t
(2) there exists a poly function P( ) so that for every string s, $s \in X$ if and only if $\boxed{\exists \text{ a string } t}$ such that B(s,t) = yes, $|t| \le P(|s|)$
①存在性 (不管找到的过程) ②只要求答案是yes的时候存在
B(s,t):
1. evaluate s under t.
2. return yes if s is satisfied by t
no otherwise.

Hamilton cycle problem
Given a G=(V,E), is there a simple cycle that visits all vertices exactly once?
⇒ hint: a simple cycle B: 按给的圈走一次

NP is the set of all problems for which there exists an efficient verifier.
所有多项式时间可验证的问题

LEMMA : $P \subseteq NP$
$x \in P \Rightarrow \exists A$ solves x 则 B(s,t): 1. run A on s (多项式时间) 直接求解
2. return the result



P = NP ? - unknown 未被证明or证伪.
看hardest ⇒ 如何比较难度? 归约 (reduction)

reduction 归约

$X \qquad\qquad\qquad\qquad Y$

$I_x \xrightarrow{f\text{-polynomial-time}} I_y = f(I_x)$ Y更难. 因为 poly Y 可推 poly X.

$I_x \in X$ iff $I_y = f(I_x) \in Y$ 记作 $\boxed{X \le_p Y}$

∃ polynomial-time algorithm A solves Y, $I_x \in X$? 判断转换后是否 $\in Y$

$I_x \to \boxed{f} \to f(I_x) \to \boxed{A} \to A(f(I_x))$

poly($|I_x|$) poly($f|I_x|$)

输出规模不会超过 running time

总时间: poly($|I_x|$) + poly(ploy($|I_x|$)) = poly(I_x)

→ If $X \le_p Y$, and $Y \in P$, then $X \in P$.

考虑两个问题:

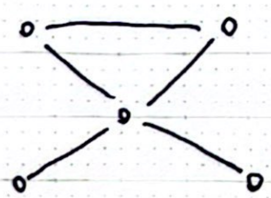
① HCP - Hamilton Cycle Problem (见前面问题描述)

② TSP - Travelling Salesman Problem

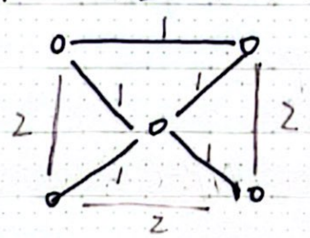
Given a weighted complete graph $G=(V,E)$, and an interger k, is there a simple cycle that visits all vertices exactly once and with total cost $\le k$.

如何证 $HCP \le_p TSP$? 本质是找 f. $G \xrightarrow{f} G'$ & k.

$G=(V,E)$



$G'=(V',E')$ and $k'=|V|$



if G has a Hamiltonian Cycle, G' has a solution with cost at most $|V|$

if G has no Hamiltonian Cycle, every solution of G' has cost at least $|V|+1$

①Clique Problem:

Given a graph $G=(V,E)$ and an integer k, does G has a complete subgrap with at least k nodes?

clique

② Vertex Cover Problem:

Given $G=(V,E)$ and k, does G has a vertex cover with at most k vertices

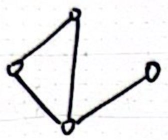
$V' \subseteq V$ s.t. every $e \in G$ has at least one endpoint in

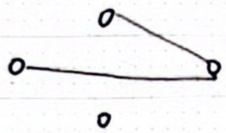
Clique $\le_p$ Vertex Cover

$G=(V,E)$ $\quad$ $G'=(V',E')$ 点集一样，边集互补

k $\quad$ k'

$E' = V \times V - E$





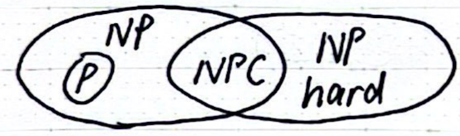
C is a clique in G if and only if $V-C$ is a vertex cover in G'

$\Rightarrow$) 反证法（假设 C 为 clique 时，$V-C$ 不是 vertex cover）

$\exists e=(u,v)\in E'$ and $u,v \notin V-C \Rightarrow (u,v)\notin E$ and $u,v\in C$ 和 clique 矛盾

$\Leftarrow$) $\exists u,v\in C$ and $(u,v)\notin E \Rightarrow u,v\notin V-C$, $(u,v)\in E'$ 和 vertex cover 矛盾

（假设 C 不为 clique）



LEMMA: ① If $X \le_p Y$ and $Y\in P$, then $X\in P$

② if $X\le_p Y$ and $Y \le_p Z$, then $X \le_p Z$

We say a problem X is NP-Complete, if: 1. $X\in NP$

2. any problem in NP $\le_p X$

$\rightarrow$ LEMMA: if a NP-C problem $\in P$, then $P=NP$

first NPC problem: circuit-SAT problem

$X\in NP$

NP-C $\le_p X$

$\Rightarrow$ X is NP-C

NP-hard: at least as hard as NP-C

co-NP $= \{\bar{x} \mid x\in NP\}$

答案是否定的，可以被hint验证 $\quad$ 答案是肯定的，可以被hint验证

$P \subseteq$ co-NP $\quad$ proof: $x\in P$, $\bar{x}\in P \Rightarrow \bar{x}\in NP \Rightarrow x\in$ co-NP

Approximation

算法要求:

1、all instance
2、polynomial time
3. optimal solution $\xrightarrow{\text{松弛}}$ 近似解

Binpack Problem (NP-hard)

Input: n items with size $s_1, s_2, \dots, s_n$ $(0 < s_i \le 1)$

Output: packing the items using fewest bins with unit capacity

法1、nextfit: 顺序存放.放不下放下一个箱子

记 $B_1, B_2, \dots, B_k$ 的容item的size为 $s(B_i)$

$$\left.\begin{aligned} &s(B_1)+s(B_2)>1 \\ &s(B_2)+s(B_3)>1 \\ &\quad\vdots \\ &s(B_{k-1})+s(B_k)>1 \end{aligned}\right. \Rightarrow s(B_1)+2s(B_2)+\dots+2s(B_{k-1})+s(B_k)>k-1$$

$$2\sum_{i=1}^{k} s(B_i) > k-1 \rightarrow \sum_{i=1}^{k} s(B_i) = \frac{k-1}{2}$$

$$\Rightarrow OPT > \frac{k-1}{2}, \quad NF=k: \begin{cases} k=2m, & OPT \ge m \\ k=2m+1. & OPT \ge m+1 \end{cases}$$

$$\Rightarrow \frac{NF}{OPT} \le 2$$

考虑 $\frac{1}{2}, \varepsilon, \frac{1}{2}, \varepsilon, \dots$ 组合
可证这个bound是tight的.

→ 2-approximation algorithm

it has an approximation ratio of (at most) 2.

△ Given an algorithm A. if for any instance I of a problem, $\boxed{\frac{A(I)}{OPT(I)}} \le \rho(n)$ we say A is a $\rho(n)$-approximation algorithm.

absolute approx ratio

同时考虑极小极大化问题: $\max\left\{\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right\}$

法2. Anyfit. (不封箱)

- firstfit: 找到第一个可装
- bestfit: 找最满的装
- worstfit: 找最空的装

for i = 1 to n:
  if any opened bin has enough space
    put item i into one of such bins.
  else
    open a new bin 如何选取?
    put item i into it.

Theorem:

$$\left.\begin{matrix} BF(I) \\ FF(I) \end{matrix}\right\} \le 1.7\, OPT(I) \text{ for Any } I.$$ and the bound is tight.

$\exists I,\ BF(I)\ (FF(I)) \ge 1.7\,(OPT(I)-1)$

法3. $\begin{cases} \text{first fit decreasing} = \text{sort} + \text{first fit} \\ \text{best fit decreasing} = \text{sort} + \text{best fit.} \end{cases}$

asynpototic approx ratio

Theorem: For any instance I, $\begin{matrix} FFD(I) \\ BFD(I) \end{matrix} \le \frac{11}{9} OPT(I) + \frac{6}{9}$

$\Rightarrow$ 绝对近似比: $\frac{FFD(I)}{OPT(I)} \le \frac{\lfloor \frac{11}{9} OPT(I) + \frac{6}{9} \rfloor}{OPT(I)} \le \frac{3}{2}$ ← tight.

NF	FF	BF	FFD	BFD
2	1.7	1.7	1.5	1.5
	online		offline.	

Theorem: For any binpacking theorem, no poly-time algorithm can achieve an approximation ratio better than $\frac{3}{2}$ unless $P = NP$.

no online algorithm is better $\frac{5}{3}$.

Knapsack Problem

Input: n items $(v_1, w_1) \cdots (v_n, w_n)$.

capacity C.

Output: fit the knapsack so as to maximize the total value.

DP - 伪多项式 ⇒ 如何在多项式时间内作近似?

Fractional version (允许选小数个 item) ⇒ greedy on $\frac{v_i}{w_i}$

Integral version (NP-hard) A1: greedy on $\frac{v_i}{w_i}$ 不可行.

反例: $C=10$.

item	value	weight
1	2	1
2	9	10

$\frac{OPT(I)}{A_1(I)} \ge \frac{C-1}{2}$

A2. greedy on v_i 反例: $C=10$

item	v	w
1~10	9 $C-1$	1
11	10_C	10_C

$OPT(I) = C(C-1)$

$A_2(I) = C$

← A2 $\frac{OPT(I)}{A_2(I)} \ge C-1$

$A^*(I)$: 1. run A_1 and A_2 on I

2. return the better of $A_1(I)$ and $A_2(I)$.

Theorem: A^* has an approx. ratio of 2.

Proof: I.

$$\underbrace{A_1(I) \ge OPT_{FRAC}(I) - V_{max} \qquad A_2(I) \ge V_{max}}_{A^*(I) \ge}$$

$$2A^*(I) \ge OPT_{FRAC}(I) \ge OPT_{INT}(I)$$

$$\Rightarrow A^*(I) \ge \frac{OPT_{INT}(I)}{2}$$

若用DP: $O(nV)$, $V=\sum_i v_i \le nV_{max} \Rightarrow O(n^2 V_{max})$ ← 降V_{max}

$v_1 \cdots v_n$ $\quad d=\gcd(v_1,\dots,v_n)$ $\quad \frac{v_1}{d} \cdots \frac{v_n}{d}$ ← 降了v.

$w_1 \cdots w_n$ $\quad\quad w_1 \cdots w_n$

最优解的集合是同一个集合

取 $d=\frac{\delta V_{max}}{n}$, $\hat{v}_i = \left\lceil \frac{v_i}{d} \right\rceil$ -scaling $\Rightarrow \hat{V}_{max} = \left\lceil \frac{V_{max}}{d} \right\rceil = \left\lceil \frac{n}{\delta} \right\rceil = O\left(\frac{n}{\delta}\right)$

则 $O(n^2 \hat{V}_{max}) = O\left(\frac{n^3}{\delta}\right)$ 但v_i的相对关系发生变化, 解会有偏差.

对任意item子集S. 记$v(S)=\sum_{i\in S} v_i$, $\hat{v}(S)=\sum_{i\in S}\hat{v}_i = \sum_{i\in S}\left\lceil\frac{v_i}{d}\right\rceil \ge \sum_{i\in S}\frac{v_i}{d} = \frac{\sum v_i}{d} = \frac{v(S)}{d}$

$\hat{v}(S) \le \sum_{i\in S}\left(\frac{v_i}{d}+1\right) = \frac{v(S)}{d}+|S| \le \frac{v(S)}{d}+n$

代入

$\Rightarrow nd + v(S) \ge d\cdot\hat{v}(S) \ge v(S)$ —②

$nd = \delta V_{max}$: $\delta V_{max} + v(S)$ —①

记 OPT under $v_i \Rightarrow S^*, v(S)^*$, OPT under $\hat{v}_i \Rightarrow \hat{S}$, $v(\hat{S})$

由①: $\delta V_{max} + v(\hat{S}) \ge d\cdot\hat{v}(\hat{S})$ → 最优解大于可行解

由②: $d\cdot\hat{v}(\hat{S}) \ge v(\hat{S})$, $d\cdot\hat{v}(S^*) \ge v(S^*)$

$\Rightarrow v(\hat{S}) + \delta V_{max} \ge d\,\hat{v}(\hat{S}) \ge d\cdot\hat{v}(S^*) \ge v(S^*)$

$$\begin{cases} v(\hat{S}) + \delta V_{max} \ge v(S^*) \\ v(S^*) \ge V_{max} \end{cases} \Rightarrow v(\hat{S}) \ge (1-\delta)\,v(S^*)$$

$$\frac{v(S^*)}{v(\hat{S})} \le \frac{1}{1-\delta} \le 1+\varepsilon \ (\varepsilon = 2\delta)$$

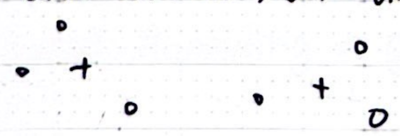
近似比可调

A polynomial-time approximation scheme (PTAS) is a family of algorithm $\{A_\varepsilon\}_{\varepsilon>0}$ such that for any $\varepsilon>0$, A_ε is an $(1+\varepsilon)$-approximation algorithm that runs in polynomial in n (given ε is a constant)

- $O(n^{\frac{1}{\varepsilon}})$ PTAS
- $O(f(\frac{1}{\varepsilon})\cdot \mathrm{poly}(n))$ efficient PTAS
- $O(\mathrm{poly}(\frac{1}{\varepsilon})\cdot \mathrm{poly}(n))$ full PTAS - FPTAS.

K-center Problem. (NP-hard)

Input: n site $s_1,\dots,s_n$ and an integer k.



⇒每个site离center距离max最小

Output: a set of k centers so as to minimize the maximum distance from a site to its nearest center.

$dist(x,y)$ = distance ~~betee~~ between x and y.

$dist(x,C) = \min_{y \in C} dist(x,y)$, $r(C) = \max_{x} dist(x,C)$.

find a set C of k centers to minimizes $r(C)$.

法1.

if $k=1$:

select one site as the center, $r^* \geq \frac{r}{2} \Rightarrow$ 二近似

<u>Assume we know OPT r^*</u> $\in (0, d_{max})$ (*) ; while there exists some site, pick an arbitary one as a center, remove all the sites within $2r^*$ from the center.

$r(C) \leq 2r^*$ (if 循环次数 $\leq k$)

proof: 反证法. assume $|C| \geq k+1$.

$\forall c_i, c_j \in C$, $dist(c_i, c_j) > 2r^*$

由于 $k+1$ 个点. OPT 只有 k 个 center. 必然有一个圆覆盖了两个 c_i. (**)

$\Rightarrow r^* \geq \frac{dist(c_i, c_j)}{2} > r^*$. 矛盾

(*) 猜 r^* 用二分在 $(0, d_{max})$ 内试. $\boxed{O(\log_2 d_{max})}$

法2. Greedy $(s_1, s_2, \ldots, s_n, k)$ 二近似算法.

1. $C_1 = \{s_1\}$
2. for $i=2$ to k
3. select the site s_j with maximun $dist(s_j, C_{i-1})$
4. $C_i = C_{i-1} \cup \{s_j\}$
5. return C_k.

$r(C_k) \leq 2r^*$.

Observation. ~~$C_k = \{c_1, c_2, \ldots, c_k\}$~~

① $r(C_1) \geq r(C_2) \geq \cdots \geq r(C_k)$.

② $C_k = \{a_1, a_2, \ldots, a_k\}$

$dist(a_i, C_{i-1}) = r(C_{i-1}) \geq r(C_k)$

$i<j$, $dist(a_i, a_j) \geq dist(a_j, C_{j-1}) \geq r(C_k)$.

Assume $r(C_k) > 2r^*$. $k+1$ 个 site 的 dist $\geq r(C_k) > 2r^*$.

由 (**) 同理得矛盾.

Theorem: 2 is tight bound unless $P = NP$.

Local Search

optimization problem (minization)

$\mathcal{C} = \{S \mid S \text{ is a feasible}\}$ ←解空间

$c: \mathcal{C} \to \mathbb{Z}$

find $\underset{S \in \mathcal{C}}{\arg\min}\ c(S)$

⇒ Local Search $(\mathcal{C}, c)$

1. pick a solution S from $\mathcal{C}$
2. while S has a better neighbor S' $(c(S') < c(S))$
3. $S := S'$

关键问题：找 neighbor

eg1. Vertex Cover Problem

Given a $G = (V, E)$ find a mininum vertex cover S

$S \subseteq V$, s.t. every $e \in E$ has at least one endpt

$\mathcal{C} = \{S \mid S \text{ is a vertex cover}\}$

$c(S) = |S|$

neighborhood of S $N(S) = \{S' \mid S' \text{ is a vertex cover and } S' \text{ can be obtained from } S$ by adding or deleting a single node$\}$

Metropolis Algorithm

1. let k, T be two constant
2. pick a solution S from $\mathcal{C}$
3. while true:
4. randomly pick a solution S' from $N(S)$
5. if $c(S') < c(S)$:
6. $S := S'$
7. else: // $c(S') \geq c(S)$
8. set $S := S'$ with probability ~~e~~ $e^{-\frac{\Delta C}{kT}}$ → $\Delta C = c(S') - c(S)$
9. break when certain condition holds.

Simulated Annealing ⇒ gradually decreasing T "退火"

eg2. Hopfield Network Problem

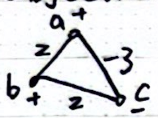
Input: $G = (V, E)$ with edge weight $w: W \to \mathbb{Z}$.

configuration
$s: V \to \{-1,+1\}$. Given a configuration s, $e=(u,v)$ is a good edge if
(1) $W_e > 0$, $s(u) \neq s(v)$ } $W_e S_u S_v < 0$
(2) $W_e < 0$, $s(u) = s(v)$

bad edge otherwise.

Objective 1. $\max \sum_{\substack{e\text{ is}\\ \text{good edge}}} |W_e|$ 好边权重大

Objective 2. u maximize $\sum_{\substack{\text{good edge}\\ e\text{ incident to }u}} |W_e|$ u的相邻边好边权重大.



good $\{bc\}$ $c \to +$ good $\{ac\}$
bad $\{ac, ab\}$ bad $\{ab, bc\}$
$U(c) = 2$ $U(c) = 3$ ← c不稳定

Given a configuration s, a node u is satisfied if:
$$\sum_{\substack{\text{good edge}\\ e\text{ incident to }u}} |W_e| \geq \sum_{\substack{\text{bad edge}\\ e\text{ incident to }u}} |W_e|$$
A configuration s is stable if every node u is satisfied.

State-Flipping:
1. pick an arbitrary configuration s
2. while some node u is not satisfied ← 会在有限步内终止吗?
3.   flip the state of u
4. return s.

Objective 1 → 可以看作是 local search to maximize $\underbrace{\text{objective 1}}_{\Phi(s)}$. proof

$\Phi(s) = \sum_{e \text{ is good}} |W_e|$ $s \xrightarrow{\text{flip } u} s'$ $\Phi(s') = \Phi(s) - \sum_{\substack{\text{good edge}\\ \text{incident to }u}} |W_e| + \sum_{\substack{\text{bad edge}\\ \text{incident to }u}} |W_e|$

$\Phi(s)$ $\Phi(s')$

$\Rightarrow \Phi(s') \geq \Phi(s) + 1$ $\Rightarrow \Phi(s) \leq \sum_e |W_e| = W$ 会

(e is a cut edge $\Leftrightarrow$ e is a good edge) for all e.

e.g.3 Maximum Cut (NP-hard) → A special case of Hopfield with $W_e > 0$
Given an undirected graph $G = (V, E)$ with edge weight $w: E \to \mathbb{Z}^+$.
A cut (A, B) is a partition of V into two non-empty subsets A and B.
$\delta(A,B) = \{(u,v) \in E \mid u \in A, v \in B\}$, $w(A,B) = \sum_{e \in \delta(A,B)} W_e$

Input: a edge weighted graph $G = (V, E)$. Output: a maximum cut.

State-flip-Max-Cut:

1. pick an arbitary cut (A,B)
2. while some node u is not satisfied.
   $\left(\sum_{\substack{\text{cut edge} \\ \text{incident to } u}} W_e < \sum_{\substack{\text{non-cut edge} \\ \text{incident to } u}} W_e \right)$
3. flip the memship of u

$\Rightarrow$ local optimum in $O(W)$ iterations
input $\log W$ → pseudo-poly.
local optimum → 2-approximation.

proof: (A,B) is stable.

for $u \in A$, $\sum_{\substack{e=(u,v)\in E \\ v\in A}} W_e$ 坏边 $\le \sum_{\substack{e=(u,v)\in E \\ v\in B}} W_e$ 好边

$$2\sum_{\substack{(u,v)\in E \\ u\in A, v\in A}} W_e = \sum_{u\in A}\sum_{\substack{e=(u,v)\in E \\ v\in A}} W_e \le \sum_{u\in A}\sum_{\substack{e=(u,v)\in E \\ v\in B}} W_e = w(A,B)$$

$$2\sum_{\substack{(u,v)\in E \\ u\in B, v\in B}} W_e \le w(A,B)$$ 同理

$$\sum_{e\in E} W_e = \sum_{\substack{e=(u,v) \\ u\in A, v\in A}} W_e + \sum_{\substack{e=(u,v) \\ u\in B, v\in B}} W_e + \sum_{e\in\delta(A,B)} W_e$$
$$\le \frac{1}{2}w(A,B) + \frac{1}{2}w(A,B) + w(A,B) \Rightarrow w(A,B) \ge \frac{\sum_{e\in E} W_e}{2} \ge \frac{OPT}{2}$$

如何把伪多项式复杂度变成多项式复杂度?

Idea: update only when there is a big improvement. ← 没有办法保证2-approximation (*)

$\Rightarrow$ flip a node u only when it increases $w(A,B)$ by a fraction of at least $\frac{\varepsilon}{|V|}$.
$\Rightarrow$ 每次flip: $w(A',B') \ge (1+\frac{\varepsilon}{n})\, w(A,B)$
由 $(1+\frac{\varepsilon}{n})^{\frac{n}{\varepsilon}} \ge 2 \Rightarrow$ 循环 $\frac{n}{\varepsilon}$ 次保证割的权重翻一番
$O(\frac{n}{\varepsilon} \cdot \log W)$ iterations.

(*) 用类似上述2近似证法可证 $(2+\frac{\varepsilon}{2})$-approximation.

↓ 邻域更丰富.

Kernighan and Lin (1970)

$(A,B) \to \{(A_1,B_1), (A_2,B_2), \dots (A_{n-1},B_{n-1})\}$.

neighbor 枚举改变后cut△最大的点
$O(n^2)$

Randomized Algorithms

deterministic algorithms 该确定 哪确定

randomized algorithms

Hiring Problem (Secretary problem)

n candidates

1、hire the best candidates

2、minimize # candidates that are hired

alg. d:

for $i=1$ to n

  if candidate i is the best so far $\Rightarrow$ any deterministic alg.

    hire i hires n candidates in worst case

随机：$\log n$ candidates in expectations.

randomly permute all the candidates.

run alg d

proof for $\lg n$: A_i = candidate i is the best among the first i candidates

$$\Pr(A_i) = \frac{1}{i}$$

$$X_i = \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{otherwise} \end{cases}$$

$$X = \sum_{i=1}^{n} X_i \quad (\text{总雇佣人数})$$

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{i} \le \ln n + 1 \qquad (*)$$

(*)可用定积分证明.

hire one candidate, maximize the probability that the best candidate

1. randomly permute all the candidates
2. interview the fairst k candidates
3. for $i = k+1$ to n
4. if candidate i is better than the best of the first k candidates
5. hire i
6. break.

Pr [the best candidate is hired]

$= \sum_{i=k+1}^{n} Pr[\underbrace{\text{the best candidate is at position } i}_{A_i} \text{ and } \underbrace{\text{candidate } i \text{ is hired}}_{B_i}]$

$= \sum_{i=k+1}^{n} Pr[A_i \wedge B_i] = \sum_{i=k+1}^{n} Pr[A_i] \cdot Pr[B_i \mid A_i]$ (**)

$Pr[A_i] = \frac{1}{n}$.

$Pr[B_i \mid A_i] = Pr$[candidate $k+1, \ldots, i-1$ is worse than the best of the first k] ($k+1$ 至 $i-1$ 均未被雇用)

$= Pr$[the best of first $i-1$ is among candidates $1 \ldots k$]

$= \frac{k}{i-1}$

$\Rightarrow$ (**) $= \sum_{i=k+1}^{n} \frac{1}{n} \cdot \frac{k}{i-1} = \frac{k}{n} \sum_{i=k+1}^{n} \frac{1}{i-1} = \frac{k}{n} \sum_{i=k}^{n-1} \frac{1}{i} \geq \frac{k}{n} \ln \frac{n}{k}$ (定积分可证)

$k = \frac{n}{e}$ 时 P 取 max $= \frac{1}{e}$.

random perm $(a_1, \dots, a_n)$

Idea1: $a_1, \dots, a_i, \cdots a_n$

$\downarrow$

$k_i = \text{random}(1, n^3)$ 重复则重新生成

$P[k_{a_i} = k_{a_j}] = \frac{1}{n^3} \Rightarrow P_{重复key} \le \sum_{i \cdot j} \frac{1}{n^3} \le \frac{1}{n^3} \cdot n^2 \le \frac{1}{n}$

Idea 2: random shuffle

for i = n to 1

j = random (1, i)

exchange a[i] with a[j]

3 SAT

$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\cdot\cdot \vee \cdot\cdot \vee \cdot\cdot) \wedge (\cdot\cdot \vee \cdot\cdot \vee \cdot\cdot)$

n: # variables

k: # clause

$x_i = \begin{cases} T & \text{with prob. } \frac{1}{2} \\ F & \text{with prob. } \frac{1}{2} \end{cases}$

Y = # clauses being satisfied

$E[Y] = \frac{7}{8}k \Rightarrow Pr(Y \ge \frac{7}{8}k) > 0$ (omitted. proof. 平均值一定有数$\ge a$)

Monte Carlo

$Pr(Y \ge \frac{7}{8}k) = ?$

$\frac{7}{8}k = E[Y] = \sum_{i=1}^{k} i \cdot Pr(Y=i)$ let k' be largest int $< \frac{7}{8}k \Rightarrow$ 至少差$\frac{1}{8}$

$= \sum_{i=1}^{k'} i \cdot Pr(Y=i) + \sum_{i=k'+1}^{nk} i \cdot Pr(Y=i) \le k' \sum_{i=0}^{k'} Pr(Y=i) + k \cdot \sum_{i=k'+1}^{nk} Pr(Y=i)$

$= k' Pr(Y < \frac{7}{8}k) + k\, Pr(Y \ge \frac{7}{8}k)$

$\le k' + k \cdot Pr(Y \ge \frac{7}{8}k)$

$\Rightarrow k \cdot Pr(Y \ge \frac{7}{8}k) \ge \frac{7}{8}k - k' \ge \frac{1}{8} \Rightarrow Pr(Y \ge \frac{7}{8}k) \ge \frac{1}{8k}$

$\boxed{\text{8k times in expectation satisfies} \ge \frac{7}{8}k}$ → Las Vegas

保证期望 ⇒ 更强的保证?

跑 $8k\ln k$ 次成功的概率? Prob of fail $\le (1-\frac{1}{8k})^{8k\ln k}$ (由 $(1-\frac{1}{x})^x \le \frac{1}{e}$)

$\le (\frac{1}{e})^{\ln k} \le \frac{1}{k}$.

$\Rightarrow 1-\frac{1}{k}$ probability find an assignment satisfying $\ge \frac{7}{8}k$ clauses

Quicksort(A)

if $|A| \le 5$: trivial

else

  chose a pivot p from A $O(1)$

  for each element $a \in A$ $O(\#\text{comparison})$

    put a in A^- if $a < p$

    put a in A^+ if $a > p$

  Quicksort (A^-)

  Quicksort (A^+)

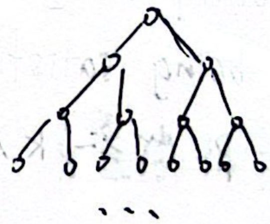
  Output A^-, p, A^+

A pivot is good if $|A^-| \ge \frac{1}{4}|A|$ and $|A^+| \ge \frac{1}{4}|A|$.

$pr(p \text{ is good pivot}) = \frac{1}{2}$

Idea 1.

1. random pick a pivot p from A
2. if p is good $\to O(n)$ in expectation
3.   use it
4. else
5.   go to 1



最多 $\log_{\frac{4}{3}} n$ 层

$O(n\log n)$ in expectation

Idea 2.

random pick a pivot
use it anyway $\Rightarrow O(n\log n)$ in expectation

total running time = O(total # comparisons)

$A = \{a_1, a_2, \dots, a_n\}$ in increasing order

for $a_i, a_j \in A$

$$X_{ij} = \begin{cases} 1, & \text{if } a_i, a_j \text{ are compared} \\ 0, & \text{otherwise} \end{cases}$$

$$X = \sum_i \sum_{j>i} X_{ij}$$

$$E[X] = \sum_i \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i}^{n} \frac{2}{j-i+1}$$

$\|$ 令 $t=j-i$

$$\sum_{i=1}^{n} \sum_{t=1}^{n-i} \frac{2}{t+1} \le \sum_{i=1}^{n} \sum_{t=1}^{n} \frac{2}{t}$$

$\|$

$O(n\log n)$

a_i a_j

① a_i or a_j was picked as a pivot

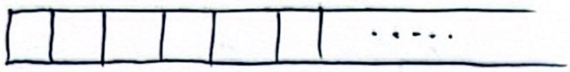
② a_i and a_j was in the same group at that time a+b　$O(\log a+\log b)$ or $O(1)$?

$O(\log a+\log b)$ ↓ 含编码 turning machine

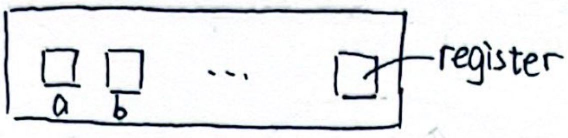
$O(1)$ ↓ RAM (Random Access Machine)

RAM

Memory: an infinite sequence of cells → store integer



CPU



four atomic operation

1. init register

$a=1$, $a=b$

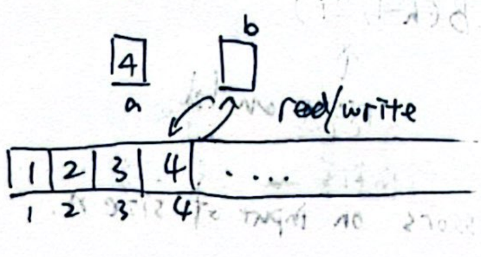
2. arithmetic

$c=a+b$, $-$ $*$ $/$ 均可 → int div，输入均为整数

3. comparison

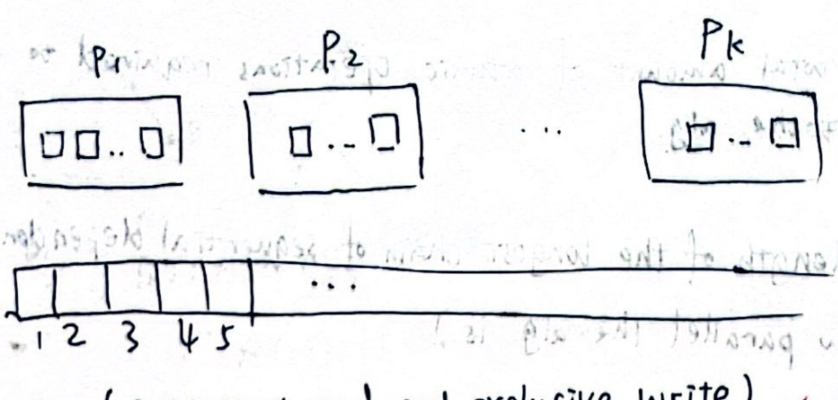
$a<b$?　$a>b$?　$a==b$?

4. memory access.



↓

PRAM



(k个 processor)

(共享一个内存)

1. CREW (concurrent read and exclusive write) △ ← 一般用这个(同时读不能同时写)

2.EREW 都不能同时

3.CRCW $\left\{ \begin{array}{l} \because \\ \because \\ \because \end{array} \right.$ (处理并行的规则)

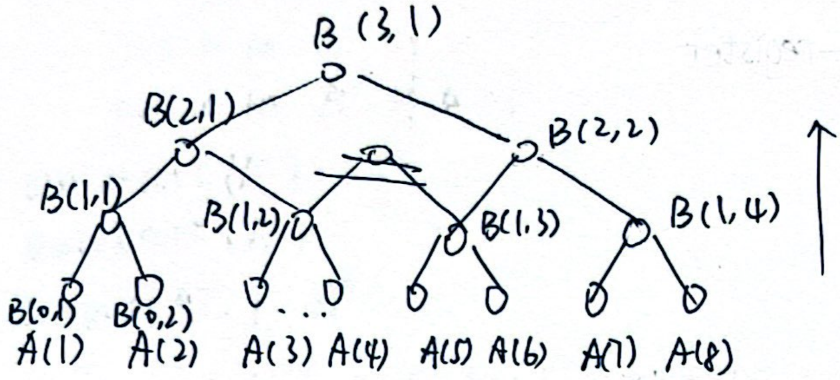
eg1. Summation

Input: A(1) ... A(n)

Output: $\sum_i A(i)$

$\Rightarrow$ $W = O(n)$, $D = O(\log n)$

并行计算方式.



for i, $1 \le i \le n$, pardo

$B(0,i) = A(i)$

for h=1 to $\log_2 n$

for i, $1 \le i \le \frac{n}{2^i}$ pardo

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

return $B(\log_2 n, 1)$ left child right child

$T_p(n)$: running time with p processors on input of size n.

$T_1(n) = O(n)$

↳ work W - total amount of atomic operations required to complete ~~to~~ the alg.

$T_\infty(n) = O(\log n)$

↳ Depth D - length of the longest chain of sequential dependencies (how parallel the alg. is)

$T_p(n)$ for arbitrary p : $\max(D, \frac{W}{p}) \le T_p(n)$

Brent's theorm: $T_p(n) \le \frac{W}{p} + D$

↓

proof: 可分成D个每组内可充分并行的组

g_1 . . . . g_D

$\sum_i g_i = W$

$\lceil \frac{g_1}{p} \rceil$ .. $\lceil \frac{g_i}{p} \rceil$

$$T_p(n) = \sum_{i=1}^{D} \lceil \frac{g_i}{p} \rceil \le \sum_{i=1}^{D} \left(\frac{g_i}{p} + 1 \right) = \frac{\sum_{i=1}^{D} g_i}{p} + D = \frac{W}{p} + D$$

A_1 W_1 D_1

A_2 W_2 D_2

1. run A_1
2. run A_2

串行 $W = W_1 + W_2$, $D = D_1 + D_2$

for $i, 1 \le i \le 2$ pardo
A_i

并行 $W = W_1 + W_2$, $D = \max(D_1, D_2)$

eg 2 Prefix Sum

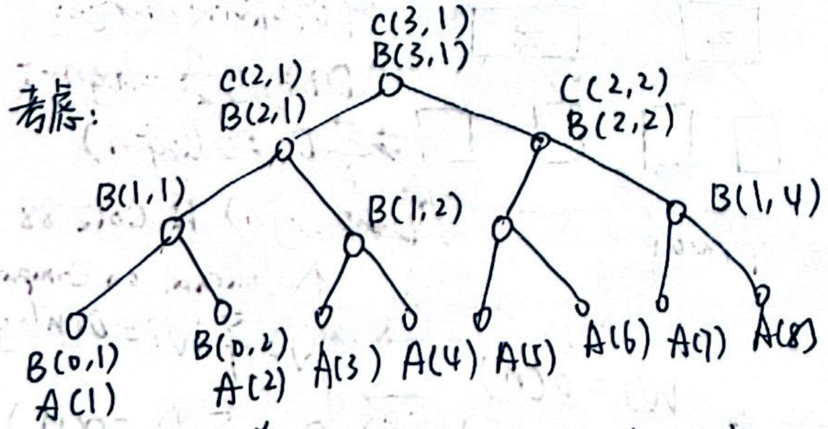
Input: $A(1), \dots, A(n)$

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

Serial: $W = O(n)$
$D = O(n)$. ← 无法并行.

Naive: $W = \sum_{j=1}^{n} O(j) = O(n^2)$
$D = O(\log n)$.
(每一项都用 e.g 1 的并行加和)

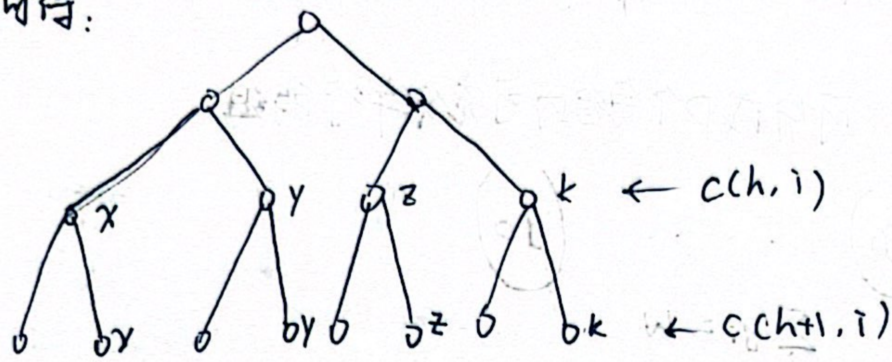
考虑:



$C(h,i) = \sum_{j=1}^{\alpha} A(j)$. $A(\alpha)$ is the rightmost leaf of the subtree rooted at $C(h,i)$

Goal: $C(0,1), C(0,2), \ldots C(0,n)$

由观察可得:



if $c(h+1,i)$ is a left child,

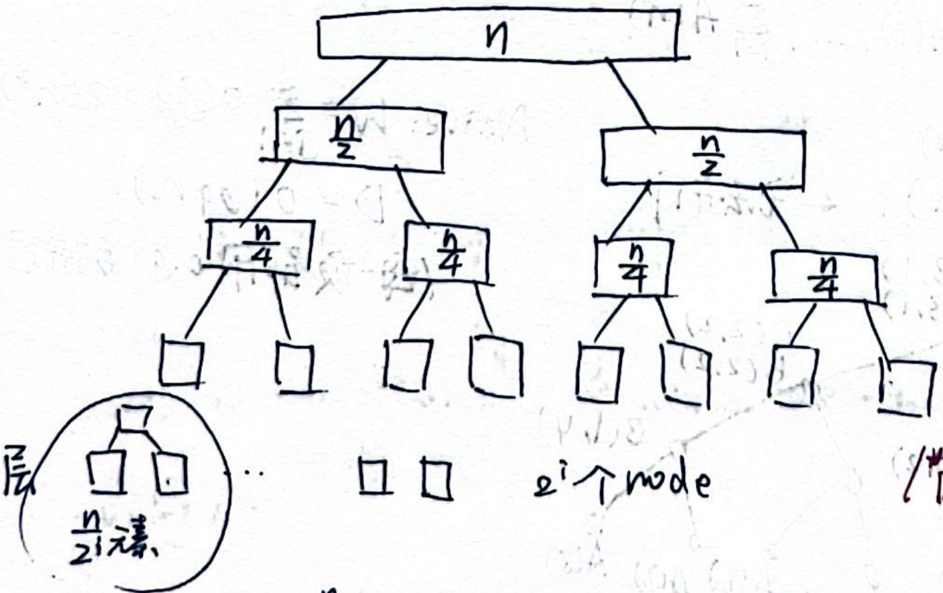
$$\underbrace{c(h+1,i) = c(h, \tfrac{i-1}{2})}_{\text{the node left to its parent}} + B(h+1,i)$$

remark: if $i==1$, $c(h+1,i) = B(h+1,i)$

if $c(h+1,i)$ is a right child, $c(h+1,i) = c(h, \underbrace{\tfrac{i}{2}}_{\text{parent}})$

由此可以一层层往下算 —— $W_B = O(n)$, $D_B = O(\log n)$, $W_C = O(n)$, $D_C = O(\log n)$ $\xrightarrow{\text{BC串行}}$ $W = O(n)$, $D = O(\log n)$

eg3. Parallel Merge Sort.



第i层 … 2^i 个 node

$\frac{n}{2^i}$ 元素

一组 $W_{ij} = O(\frac{n}{2^i})$, $D_{ij} = O(\frac{n}{2^i})$ $\xrightarrow{\text{每组可并行}}$ $W_i = O(n)$, $D_i = O(\frac{n}{2^i})$ $\xrightarrow{\text{每层}\Sigma}$ $W = \sum_i W_i = O(n\log n)$, $D = \sum_i D_i = O(n)$

$\Rightarrow$ 并行瓶颈: Merge

优化后的 Merge:

$D_i = \log\frac{n}{2^i} = \log n - i$

$\Rightarrow \sum_i D_i = O(\log^2 n)$

$\Rightarrow D = O(\log n)$ R. Cole 88

Journal on Computing

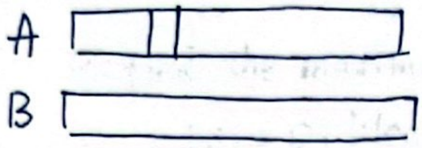
Merge

Input: Sorted Array A and B

Output: an Sorted Array C

serial: $W = O(n)$

$D = O(n)$



(假设无重复整数 in A.B)

rank(i, B): rank of A[i] in B

rank(i, A): rank of B[i] in A.

for i, $1 \le i \le n$ pardo

C[i+rank (i, B)] = A[i]

C[i+rank (i, B)] = B[i]

using parallel rank

$W = O(n)$

$D = O(1)$

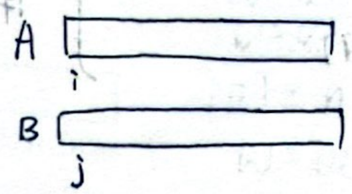
$W = O(n)$

$D = O(\log n)$

Ranking

Output: Rank (i, B) and Rank (i, A) for all i.

1. serial ranking (类似归并)



if A[i] < B[j]

rank (i, B) = j, i++

if A[i] > B[j]

rank (j, A) = i, j++.

$W = O(n)$

$D = O(n)$

2. binary search

for i, $1 \le i \le n$ pardo

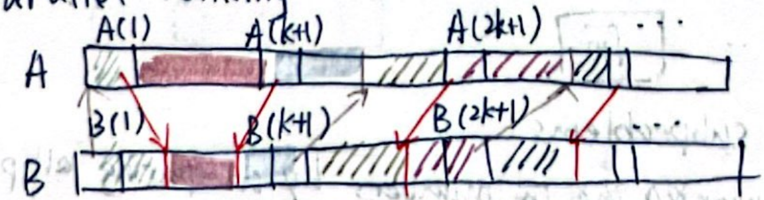
rank(i, B) = BS (A[i], B)

rank(i, A) = BS (B[i], A)

$W = O(n \log n)$

$D = O(\log n)$

3. parallel ranking



两线不会交叉 (反证可证)

↓

把两线之间 entries 分成若干组

① using binary search ranking on selected entries.

$W_1 = O(\frac{2n}{k} \cdot \log n)$, $D_1 = O(\log n)$

间隔k排+不交叉

↓

每组最多 ~~$\frac{2n}{k}$~~ 2K 个

② serial ranking for each group. (parallelly)

$W_2 = O(n)$

$D_2 = O(k)$

total: $W = W_1 + W_2 = O(\frac{n}{k} \cdot \log n + n) = O(n)$

$D = D_1 + D_2 = O(\log n + k) \underset{\uparrow}{=} O(\log n)$

Let $k = \log n$

e.g.4 Maximum finding

Input: $A(1) \ldots A(n)$

Output: $\max A(i)$

0. serial $\underline{W = O(n)}$ $D = O(n)$

1. use the summation alg. ($+ \to \max$) $\underline{W = O(n)}$. $D = O(\log n)$

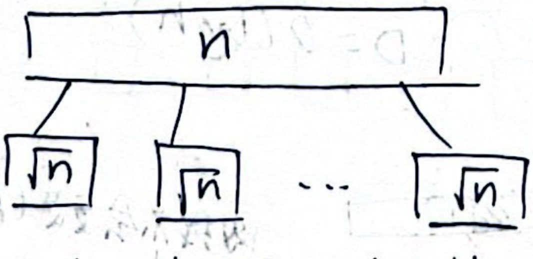
2. compare all pairs

```
for i. 1≤i≤n pardo
    B(i) = 0
for every pair (i,j) with i<j pardo
    if A[i] < A[j]
        B[i] = 1
    else  // A[j] < A[i]
        B[j] = 1
for i, 1≤i≤n pardo
    if B[i] == 0:
        A[i] is the maximum.
```

可能出现 B[i] 同时写的情况
用 Common CRCW

$W = O(n^2)$. $D = O(1)$.

3. Divide-and-Conquer.



① recursively solve $\sqrt{n}$ subproblems

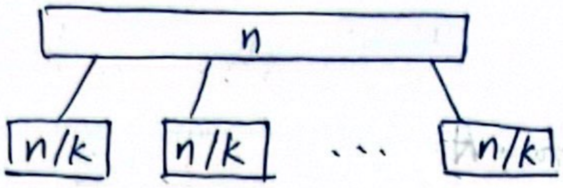
② find the maximum among the $\sqrt{n}$ numbers by comparing all pairs

$W(n) = \sqrt{n}\, W(\sqrt{n}) + O(n) \rightarrow W(n) = O(n \log\log n)$ 和 $W = O(n)$

$D(n) = D(\sqrt{n}) + O(1) \qquad \underline{D(n) = O(\log\log n)}$ 结合?

4、

3.1结合



① solve subproblems using serial ranking

$W_1 = O(n)$

$D_1 = O(\frac{n}{k})$

② find the maximum among the k numbers using D & C

$W_2 = O(k\log\log k)$

$D_2 = O(\log\log k)$ let $k = \frac{n}{\log\log n}$

total: $W = O(n + k\log\log k) = O(n)$

$D = O(\frac{n}{k} + \log\log k) = O(\log\log n)$

5. Random Sampling $W = O(n)$, $D = O(1)$ with high probability $1 - \frac{1}{n^c}$

return maximum.

A $|A| = n$

↓ random sample $W = O(n)$ $D = O(1)$

B $|B| = n^{\frac{7}{8}}$

↓ partition into $n^{\frac{3}{4}}$ groups $W = O(n^{\frac{7}{8}})$ $D = O(1)$

□ □ ... □

$n^{\frac{3}{4}}$ groups, each of size $n^{\frac{1}{8}}$

↓ find the maximum of each group by comparing all pairs $W = O(n^{\frac{3}{4}} \cdot n^{\frac{1}{4}}) = O(n)$ $D = O(1)$

C $|C| = n^{\frac{3}{4}}$

↓ partition $n^{\frac{1}{2}}$ groups each of size $n^{\frac{1}{4}}$ $W = O(n^{\frac{3}{4}})$ $D = O(1)$

□ □ □ □ .. □

↓ $W = O(n^{\frac{1}{2}} \cdot n^{\frac{1}{2}}) = O(n)$ $D = O(1)$.

D $|D| = n^{\frac{1}{2}}$

↓ finding the max (两两比较) $W = O(n)$ $D = O(1)$

max

random sample.

for i: $1 \leq i \leq n^{\frac{7}{8}}$ pardo

B[i] = random select from A.

---

round 2.

for i, $1 \leq i \leq n^{\frac{7}{8}}$ pardo

B[i]

for i, $1 \leq i \leq n$ pardo

if A[i] > m,

throw A[i] into a random place of B

find a maximum of B.

$$W = O(n). \quad D = O(1).$$

充分条件： $\underbrace{\text{rank}(m) \leq n^{\frac{1}{4}}}_{E_1}$ and $\underbrace{\text{all } A[i] > m \text{ was thrown in different places of B}}_{E_2}$

$\Downarrow$

SUCCESS

$$Pr(\text{success}) \geq Pr(E_1 \cap E_2) \geq Pr(E_1) \cdot Pr(E_2 | E_1)$$

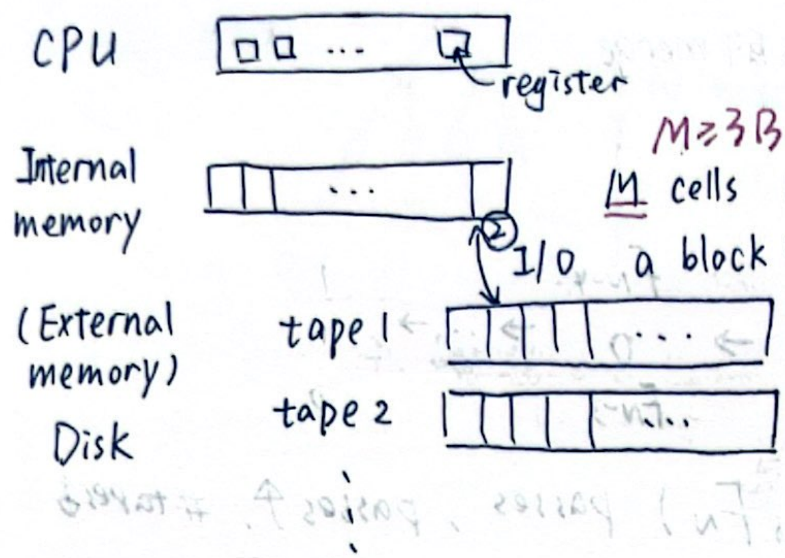
$$\geq \left(1 - \frac{1}{n^{\frac{3}{4}}}\right)^{n^{\frac{7}{8}}} \geq 1 - \left(\left(1 - \frac{1}{n^{\frac{3}{4}}}\right)^{n^{\frac{3}{4}}}\right)^{n^{\frac{1}{8}}}$$

$\leq e^{-\cancel{n^{\frac{3}{4}}}}$

$$\geq 1 - e^{-n^{\frac{1}{8}}}$$

(严谨证明过程不要求)

External Memory Model



M≥3B 通常设

M cells (items)

a block 为单位: B cells.

... infinite length

① 只支持 sequential access.

③ 关心 I/O cost:

\# blocks moved.

N: input size (# items in the input) N >> M

scan: $a_1, \dots, a_N$ I/O cost: $\frac{N}{B}$ → linear cost ~~time~~ ~~∝~~ $O(N)$

从头到尾扫一遍 - one pass

Sorting

2-way merge

\# passes : $1 + \lceil \log_2 \frac{N}{M} \rceil$ → #runs 每一次pass runs 都除以 k-way

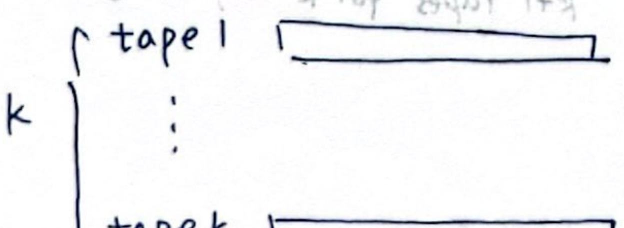
I/O cost : $O\left(\frac{N}{B} \cdot \log_2 \frac{N}{M}\right)$

k-way merge

\# passes : $1 + \lceil \log_k \frac{N}{M} \rceil$

I/O cost : $O\left(\frac{N}{B} \cdot \log_k \frac{N}{M}\right)$

k 的上限? — 磁带中读是以 block 为单位的.



k-way merge

→ 2k input buffers

2 output buffers

将 k blocks 读入内存比较, 2 blocks 作为存放输出, 则 $(k+2)B \le M \Rightarrow \boxed{k \le \frac{M}{B} - 2}$

$$\Rightarrow \begin{cases} \#\text{pass}: 1 + \log_{\frac{M}{B}-2} \frac{N}{M} \\ \text{I/O cost}: O\left(\frac{N}{B} \cdot \log_{\frac{M}{B}} \frac{N}{M}\right) \end{cases}$$

- longer run 工程实现

- 用 Huffman coding 实现低 cost 的 merge.

- Fibonacci 数列拆分.

3条磁带:

T_1 F_N → 0, F_{N-1}, F_{N-2} → F_{N-2}, F_{N-3}, 0 → F_{N-4}, 0, F_{N-3} → … → 1, 1, 0

$O(\log_{\frac{\sqrt{5}+1}{2}} F_N) = O(\log_{1.6} F_N)$ passes , passes↑, #tapes↓

4条磁带(三合一)

$$\begin{array}{cccccccccc} 0 & 0 & 1 & 2 & 3 & 6 & 11 & 20 & 37 & \cdots \\ F^{(2)}(0) & F^{(2)}(1) & F^{(2)}(2) \end{array}$$

$$F^{(2)}(N) = F^{(2)}(N-1) + F^{(2)}(N-2) + F^{(2)}(N-3)$$

T_1 $F^{(2)}_{N-1} + F^{(2)}_{N-2} + F^{(2)}_{N-3}$

T_2 $F^{(2)}_{N-1} + F^{(2)}_{N-2}$

T_3 $F^{(2)}_{N-1}$

T_4 0

→

T_1 $F^{(2)}_{N-2} + F^{(2)}_{N-1}$

T_2 $F^{(2)}_{N-2}$

T_3 0

T_4 $F^{(2)}_{N-1} = F^{(2)}_{N-2} + F^{(2)}_{N-3} + F^{(2)}_{N-4}$

⇓

---

k条磁带

$$\begin{array}{ccc} 0 \cdots & 0 & 1 \cdots \\ F^{(k)}(0) \cdots & F^{(k)}_{(k-2)} & F^{(k)}_{k-1} \end{array}$$

$$F^{(k)}_N = F^{(k)}_{N-1} + \cdots + F^{(k)}_{N-k}$$

T_1 $F^{(k)}_{N-1} + \cdots + F^{(k)}_{N-k}$

⋮ $F^{(k)}_{N-1} + \cdots + F^{(k)}_{N-k+1}$

⋮

T_k $F^{(k)}_{N-1}$

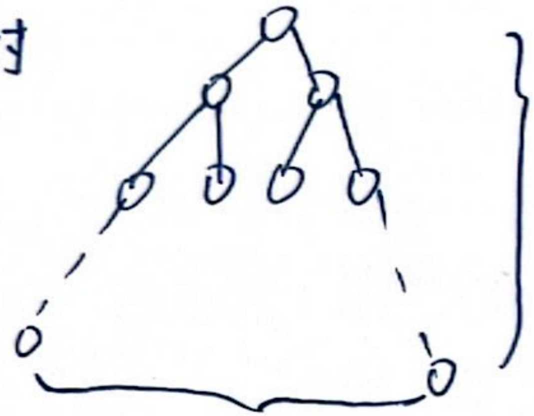
T_{k+1} 0

poly phase merge

k+1 tapes for k-way merge.

Searching

计：平衡二叉树



N nodes

~~N leaves~~

$\log_2 N$ levels

↓

I/O cost: $O(\log_2 N)$

↓改进

用B+树.

每个节点是一个block, I/O cost: $O(\log_B N)$