随机算法

deterministic ALG (past)

输入给定. 输出?

Randomized alg

(secretary) Hiring problem

n candidates 面试 Y/N?

1. hire the best candidates (all)
2. minimize # candidates that are hired.

1 2 3 ...... ↗

for $i=1$ to n
  if i is the best so far
    hire(i)

→ any deterministic hires n candidates in worst cases

$1+\ln n$ candidates in expections.

randomly permute all candidates,

再执行刚刚:

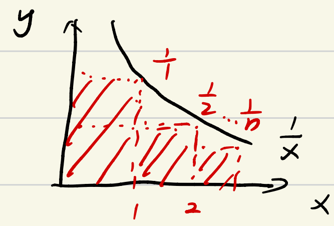
for $i=1$ to n
  if i is the best so far
    hire(i)

A_i = candidate i is the best among the first i candidates.

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

$$x_i=\begin{cases}1 & i \text{ hired}\\ 0 & \text{otherwise}\end{cases}$$

$$X=\sum_{i=1}^{n}x_i \qquad E[X]=\sum_{i=1}^{n}EX_i=\sum_{i=1}^{n}\frac{1}{i}\le \ln n+1$$



面积: $1+\int_1^n \frac{1}{x}=1+\ln n$

---

hire one candidate,

maximize the probability that the best candidate is hired.

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

Pr(the best candidate is hired)

$$=\sum_{i=k+1}^{n}Pr(\underbrace{\text{...... ...... at pos } i}_{A_i}\ \text{and}\ \underbrace{i \text{ is hired}}_{B_i})$$

$$=\sum_{i=k+1}^{n}Pr(A_i\wedge B_i)$$

$$=\sum_{i=k+1}^{n}\underbrace{Pr(A_i)}_{\frac{1}{n}}\cdot P(B_i\mid A_i)$$

1 ... k $k+1$ ... (no hire) ... i

$P(B_i \mid A_i) = Pr[$ candidate $k+1 \cdots i-1$ is worse than the best of the first k.

$= Pr($ the best of first $i-1$ is among the candidates $1 \cdots k)$ $= \frac{k}{i-1}$

$$原式 = \sum_{i=k+1}^{n} \frac{1}{n} \cdot \frac{k}{i-1} = \frac{k}{n} \sum_{i=k}^{n-1} \frac{1}{i} \geq \frac{k}{n} \cdot \ln \frac{n}{k}$$

$> \int_k^n \frac{1}{x} dx = \ln \frac{n}{k}$

当 $k = \frac{n}{e}$ 时 $Pr \geq \frac{1}{e}$

---

random permute $(a_1 \cdots a_n)$

idea 1 $a_1 \cdots a_i \cdots a_n$

$k_i = random(1, n^3)$, 按 k_i 重排

撞key: $a_i \cdot a_j = \frac{1}{n^3}$

tot: $\leq \sum_{i,j} a_{ij} \leq \frac{1}{n}$

idea2 random shuffle

for $i = n$ to 1

$j = random(1, i)$

exchange a_i with a_j

3-SAT $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\quad) \wedge (\quad)$

n: variables

k: # clauses

找到一个 assignment. 满足尽可能多的 clauses

$x_i = \begin{cases} T & \text{with Prob } \frac{1}{2} \\ F & \cdots \end{cases}$ $O(n)$ 不保证结果质量

Y = # clauses being satisfied.

Monte Carlo

蒙特卡洛 时间 √ 质量 ×

? —— $E[Y] = \frac{7}{8}k$

$\exists$ alg. 至少满足 $\frac{7}{8}k$ # ? √

$Pr(Y \geq \frac{7}{8}k) > 0$ 存在!

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

$E[Y] = \sum_{i=0}^{k} i \cdot Pr(Y=i) = \frac{7}{8}k$

Let k' be the largest int $< \frac{7}{8}k$.

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

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

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

$\leq k' + k \cdot Pr(Y \geq \frac{7}{8}k)$

整数

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

$Pr(Y \geq \frac{7}{8}k) \geq \frac{1}{8k}$

Las Wegas

质量 √

时间 ×

跑 $8k$ 次 $8k$ times in expections

不仅是期望 $8k\ln k$ 次 $\leq (1 - \frac{1}{8k})^{8k\ln k} \leq (e^{-1})^{\ln k} \leq \frac{1}{k}$

satisfy $\geq$

$(1 - \frac{1}{x})^x \leq e^{-1}$

Prob of fail

with probability $1 - \frac{1}{k}$

find an assignment satisfy $\geq \frac{7}{8}k$ clauses

Quicksort (A)

if $|A| \leq s$.. trivial

else choose a pivot P from A;

for each element $a \in A$

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

. . . . A^+ $>$

Quicksort (A^-)

. . . . . . A^+

Output A^{-1} p A^+

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

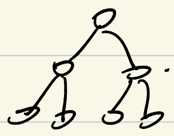
idea 1

random pick a pivot p from A.

1. if p is good.
2. use it
3. else.
4. go to 1.

$a_1 \cdots \overbrace{\tfrac{n}{4} \cdots \tfrac{3}{4}n}^{} \cdots a_n$

$Pr(p \text{ is good}) = 1/2$



. . . $O(n)$ in expection

$\log n$层 $O(n\log n)$ in expection

idea 2. random pick a pivot

use it anyway. $O(n\log n)$ in expection

pick: $O(1)$ + for $O(\#\text{comparison})$

total running time $= O(\text{total } \#\text{times})$

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

for $a_i, a_j \in A$

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

$X = \sum_{i<j} x_{ij}$ $E[X] = \sum_i \sum_{j>i} E[x_{ij}]$

比较的概率 $\underset{\circ}{a_i}$ $\underset{\circ}{a_j}$

① a_i/a_j was picked as a pivot

有个为pivot

② a_i & a_j was in the same group at that time

$Pr[a_i \text{ or } a_j \text{ was the first pivot among } a_i \cdots a_j]$

$= \frac{2}{j-i+1}$

$E[X] = \sum_i \sum_{j>i} \frac{2}{j-i+1} = \sum_{i=1}^{n} \sum_{t=1}^{n-1} \frac{2}{t+1} \leq \sum_{i=1}^{n} \sum_{t=1}^{n} \frac{2}{t} = O(n\log n)$

令$t=j-i$

并行算法

$a+b$ 需要多久 $O(\log a+\log b)$ or $O(1)$

Turing Machine (要编码) RAM model

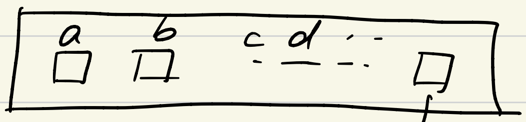
Random Access Machine

RAM: Memory : an infinite sequence of cells (1 2 3 4 ··· → addr)

store 1 integers

\+

CPU



常数个cell

register

4 atomic operations

认为单位时间

① init reg

e.g. $a=1$ $a=b$

② arithmetic → int div

$c = a + - * / b$

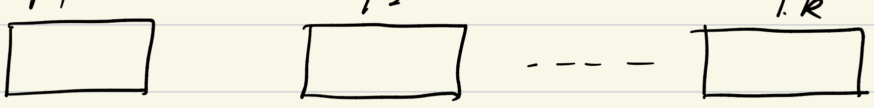
③ comparison

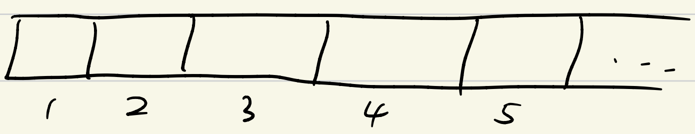
$a<b$?

④ memory access

a ← R/W (1 2 3 4 ···)

PRAM 多核





共享内存 (可能有冲突)

一般 △ ① CREW Concurrent read & Exclusive write 写排队

② EREW 都排队

③ CRCW { ···· 处理冲突

Summation

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

Output: $\sum A(i)$

for i $1 \le i \le n$ pardo → 可并行执行

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

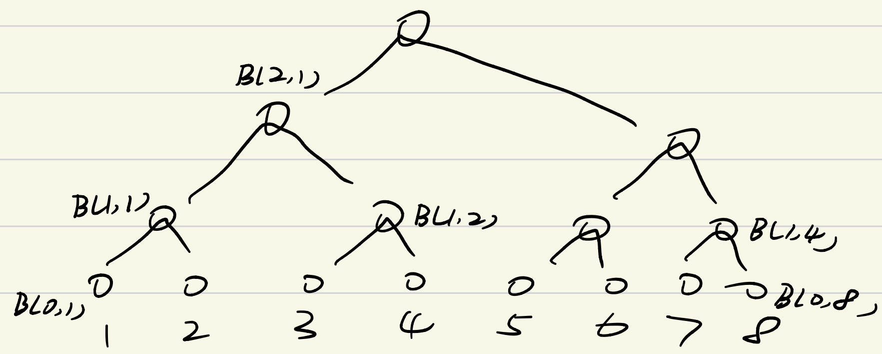
for $h=1$ to $\log_2 n$

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

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

left child right child

return $B(\log_2 n, 1)$



$W = O(n)$

$D = O(\log n)$

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

$T_1(n) = O(n)$

↳ work 总工作量 W: total amount of atomic operations required to complete the alg.

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

↳ Depth D: length of the longest chain of sequential dependencies

反映 how parallel the alg is 并行程度

$T_p(n)$ for arbitrary p

$\frac{W}{P}$? $T_p(n) \geq \max(\frac{W}{P}, D)$ 上界?

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

proof. 每一组内没有依赖关系（组间有）

g_1 …… g_D $\Sigma g_i = W$

$\lceil \frac{g_1}{P} \rceil$ $\lceil \frac{g_i}{P} \rceil$

$$T_p(n) = \sum_{i=1}^{D} \lceil \frac{g_i}{P} \rceil \leq \sum_{i=1}^{D} (\frac{g_i}{P} + 1)$$
$$= \frac{W}{P} + D$$

A_1 W_1 D_1

A_2 W_2 D_2

串行 A_1 A_2: $W = W_1 + W_2$, $D = D_1 + D_2$

并行 for i $1 \leq i \leq 2$ pardo $W = W_1 + W_2$

A_i $D = \max(D_1, D_2)$

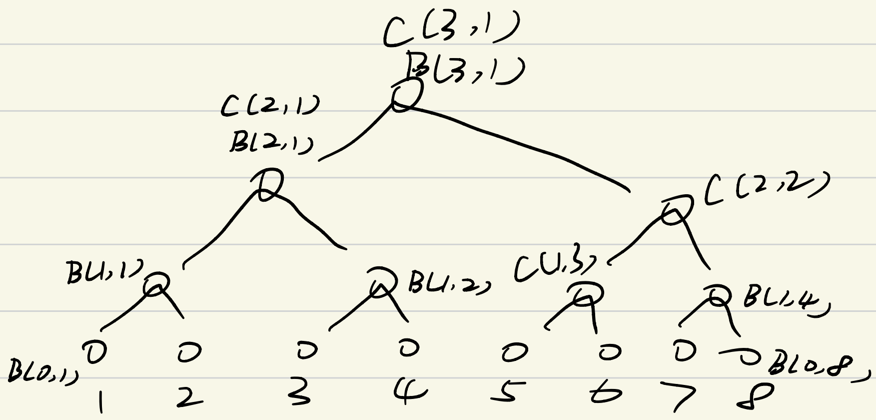
Prefix Sum

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

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

serial : $W = O(n)$ Naive: 并行求n个 prefix sum $W = \sum_{j=1}^{n} O(j) = O(n^2)$

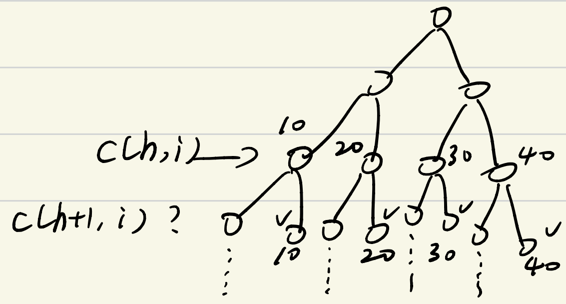
$D = O(n)$ 无法并行 $D = O(\log n)$



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

如: $C(1,3) = A1 + \cdots + A6$

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



叔的左兄弟

if $C(h+1,i)$ is a left child $C(h+1,i) = C(h, \frac{i-1}{2}) + B(h+1,i)$

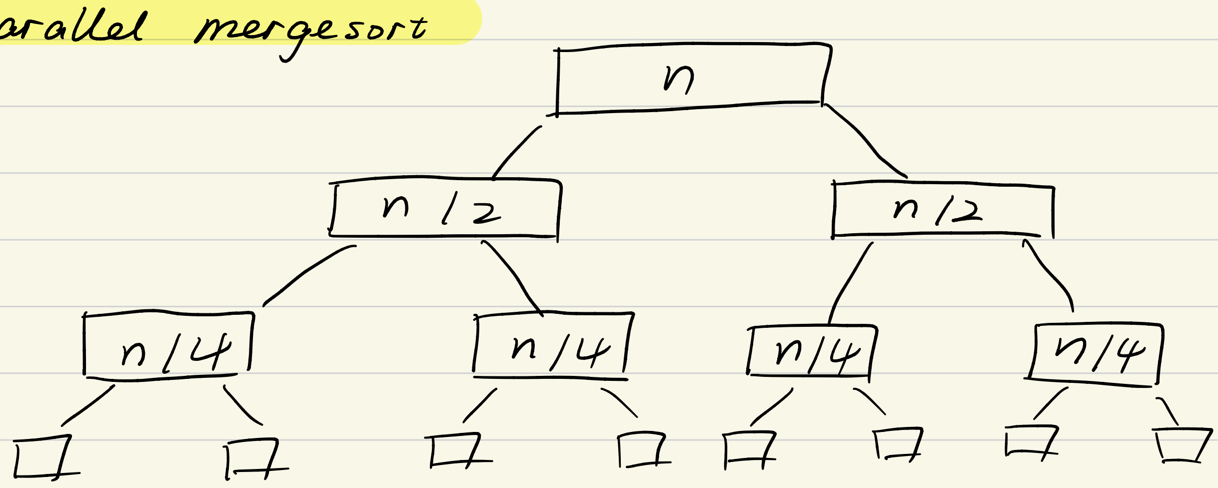
remark: if $i == 1$ $C(h+1,i) = B(h+1,i)$

if $C(h+1,i)$ is a right child $C(h+1,i) = C(h, \frac{i}{2})$

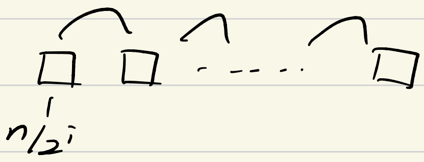
parent

$W_B = O(n)$
$D_B = O(\log n)$
$W_C = O(n)$ $\Rightarrow$ B.C 串行 $W = O(n)$
$D_C = O(\log n)$ $D = O(\log n)$

Parallel mergesort



i-level 2^i nodes



$W_i = O(\frac{n}{2^i}) = D_i$ 第 i 层 $W_i = O(n)$
$D_i = O(\frac{n}{2^i})$

$\therefore W = \sum_i W_i = O(n\log n)$
$D = \sum_i D_i = O(n)$

ranking
parallel merge 后
$D_i = O(\log \frac{n}{2^i})$
$D = \sum D_i = O(\log^2 n)$

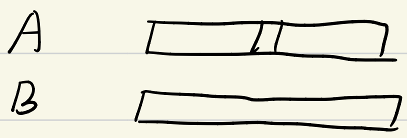
实际上
$D \to O(\log n)$ R.Cole 88
Jcomp

瓶颈在于 merge.

Merge

Input: sorted array A & B (假设 $a_i \neq b_j$)
Output: an sorted array C.

serial: $W = O(n) = D$



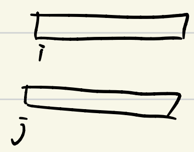
$rank(i, B)$ = rank of $A[i]$ in B.
$rank(i, A)$ = rank of $B[i]$ in A.

如果知道 rank for i $1 \le i \le n$ pardo
$C[i + rank(i,B)] = A[i]$
$C[i + rank(i,A)] = B[i]$
$W = O(n)$ $D = O(1)$

Ranking

Output: $rank(i, B)$ & $rank(i, A)$ for all i.

1. serial ranking 类似 merge



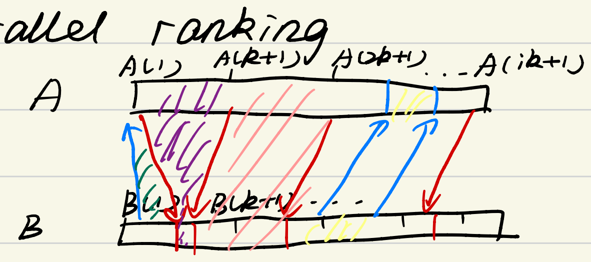
if $a_i < b_j$:
$rank(i, B) = j$
i++
if $a_i > b_j$
$rank(i, A) = i$
j++
$W = O(n)$. $D = O(n)$

idea:
ALG 1 work↓
ALG 2 depth↓ — combine

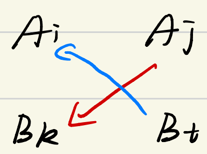
2. binary search
for i, $1 \le i \le n$ pardo
$rank(i,B) = BS(A[i], B)$
$rank(i,A) = BS(B[i], A)$

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

3. parallel ranking



↑↓ 不会交叉



$A_j > B_t$
但 $A_j < B_t$

① using binary search ranking on selected entries
分组 左边的组 < 右边

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

② serial ranking for each group (parallelly)

$W_2 = O(n)$ 每组至多 2k entries

$D_2 = O(k)$

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

$D = O(\log n + k)$ Let $k = \log n$ 则 $D = O(\log n)$

Maximum finding

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

Output: $\max_i A(i)$

0. serial $W = D = O(n)$

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

2. compare all pairs

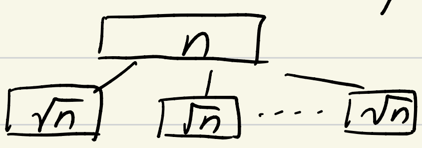
```
for i 1≤i≤n pardo
    B(i)=0

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

? 多个 CPU 可能同时 write
Common CRCW
└ 要求同时写的值一样

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

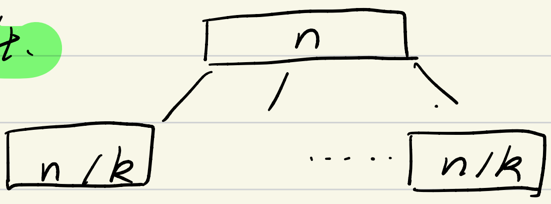
3. Divide-and-conquer



① recursively solve $\sqrt{n}$ subproblems
② find the maximum among the $\sqrt{n}$ numbers by comparing all pairs

$W(n) = \sqrt{n} W(\sqrt{n}) + O(n)$
$D(n) = D(\sqrt{n}) + O(1)$
$\Rightarrow W(n) = O(n \log\log n)$
$D(n) = O(\log\log n)$

4.



① solve subproblems using serial ranking

$W_1 = O(n)$ $D_1 = O(n/k)$

② find the maximum among the k using D&C

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

$D_2 = O(\log\log k)$

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

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

5. Random sampling $W = O(n)$ $D = O(1)$

with highly prob $1 - \frac{1}{n^c}$ return maximum.

$\boxed{A}$ $|A| = n$

↓ random sample

$\boxed{B}$ $|B| = n^{\frac{7}{8}}$ — — 可以降到 $\sqrt{n}$

↓ partition into $n^{\frac{3}{4}}$ group

□ □ □ ⋯ □

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

↓ find the max of each group by comparing all pairs

$W = n^{\frac{1}{4}} \cdot n^{\frac{3}{4}} = O(n)$ $D = O(1)$

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

↓ partition $n^{\frac{1}{2}}$ groups, each of size $n^{\frac{1}{4}}$

□ ⋯ □

↓ $W = O(n)$ $D = O(1)$

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

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

random sample

for i; $1 \le i \le n^{7/8}$ pardo

可能选重复

$B[i]$ = random select from A $W = O(n)$ $D = O(1)$

可能漏掉最大值. 再来一轮

round 2 for i; $1 \le i \le n^{7/8}$ pardo

$B[i]$

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

if $A[i] > m$ Round 1 ans

可能重复

如果 m 大, 那漏的就少

R2 success ↑ prob

throw $A[i]$ into a random space of B.

find a maximum of B. $W = O(n)$. $D = O(1)$

若 $\underbrace{\text{rank}(m) \le n^{\frac{1}{4}}}_{E_1}$ and $\underbrace{\text{all } A[i] > m \text{ are thrown in different places of } B}_{E_2}$

⇓ 就可以

success

不懂 无所谓 ⋯ $Pr(\text{success}) \ge Pr(E_1 \cap E_2) \ge Pr(E_1)\, Pr(E_2 | E_1)$

$(1-\frac{1}{x})^x \le e^{-1}$

E_1: 每次成功 $\frac{n^{\frac{1}{4}}}{n} = \frac{1}{n^{\frac{3}{4}}}$

$Pr(E_1) \geqslant 1 - \underbrace{(1-\frac{1}{n^{\frac{3}{4}}})}_{失败}^{n^{\frac{7}{8}}} \geqslant 1-(1-\frac{1}{n^{\frac{3}{4}}})^{n^{\frac{3}{4}} \cdot n^{\frac{1}{8}}}$

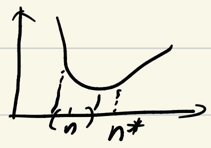
$n^{\frac{1}{4}} <. \to n^{\frac{7}{8}}$

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

$Pr(E_2 | E_1)$

Local Search

$f(n) = (an - b)^2$ (unknown: a, b)



find $\underset{n \in \mathbb{Z}}{\operatorname{argmin}} f(n)$

Optimization problem (e.g. minization)

$C = \{S \mid S \text{ is feasible}\}$ 可行解

cost $c: C \to \mathbb{Z}$ find $\underset{S \in C}{\operatorname{argmin}}\, c(S)$ 找代价最小可行解

① pick a solution S from C

② while S has a better neighbor S' ($c(S') < c(S)$)

③ $S \leftarrow S'$

Vertex Cover

Given a graph $G = (V, E)$

find a minimum vertex cover S ($S \subseteq V$ s.t. every $e \in E$ has at least one endpt in S)

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

$c(S) = |S|$

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

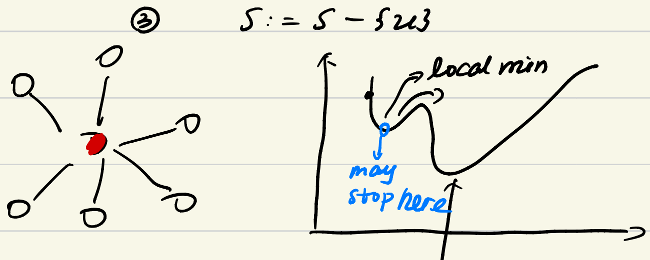
neighborhood of S

LSVC(V, E)

① $S = V$

② if $S - \{u\}$ is a vertex cover for some $u \in S$

③ $S := S - \{u\}$



Metropolis Algorithm

1. Let k. T be 2 constants.
2. pick a solution S from C
3. while true.
4. randomly pick a sol 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 $e^{-\frac{\Delta C}{kT}}$ ($\Delta C = c(S') - c(S)$) 一定prob. 爬出坑 $\Delta C \uparrow$ prob $\downarrow$; $T \downarrow$ prob $\uparrow$ 温度
9. break when some conditions hold.

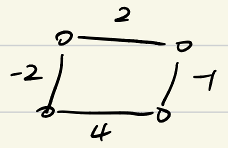
Simulated Annealing 模拟退火

gradually decreasing $T \downarrow$ 先活跃,再稳定

无法分析

Hopfield Network Problem

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



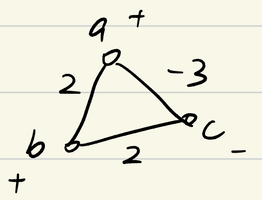
$s: V \to \{+1,-1\}$ 给节点赋值
configuration

e is a good edge (1) if $w_e > 0$, $s(u) \neq s(v)$ } $w_e s_u s_v < 0$
(2) … $<$ $=$

bad otherwise.

Objective. 1. $\max \sum_{e \text{ is good}} |w_e|$

Objective. 2. u. $\max \sum_{\substack{e \text{ good} \\ \text{incident to } u}} |w_e|$



good: $\{bc\}$ $\quad$ $U(c)=2$ $\quad$ 若 $c \to +$ $\quad$ good $\{ac\}$ $\quad$ $U(c)=3$
bad: $\{ac, ab\}$ $\quad$ bad $\{ab, bc\}$

稳定,不会跳槽

Given a configuration S, a node u is satisfied if $\sum_{\substack{e \text{ good} \\ \text{incident to } u}} |w_e| \geq \sum_{\substack{e \text{ bad} \\ \dots}} |w_e|$

A configuration S is stable if every node u is satisfied.

State-flipping

local search
to max objective 1
($\Phi(S)$)

1. pick an arbitrary configuration S
2. while some node u is not satisfied
3. flip the state of u.
4. return S.

stop? ✓

config下
$\Phi(S) := \sum_{e \text{ good}} |w_e|$ → objective 1 每一步都在↑Φ

$S \xrightarrow{\text{flip } u} S'$

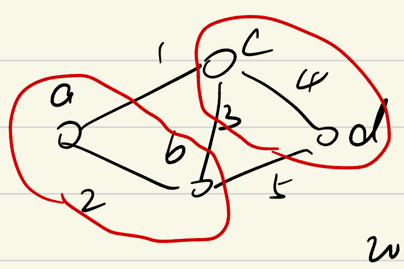
$\Phi(S)$ $\quad$ $\Phi(S') = \Phi(S) - \sum_{\substack{\text{good} \\ u}} + \sum_{\substack{\text{bad} \\ u}} > \Phi(S)$ $\quad$ $\geq \Phi(S)+1$ $\quad$ 每次至少↑1

$\Phi(S) \leq \sum_e |w_e| = W$

Maximum Cut Problem (NP-hard)

Given a 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.



$\delta(A,B) = \{(u,v) \in E \mid u \in A, v \in B\}$ 割边
$w(A,B) = \sum_{e \in \delta(A,B)} w_e$

$w = 9$

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

A special case of Hopfield with $w_e > 0$ for all e. (e is a good edge $\Leftrightarrow$ e is cut edge)

State-flip-Max-Cut:

1. pick an arbitrary cut (A, B)
2. while some nodes u are not satisfied.
   $(\sum_{cut} w_e < \sum_{non\text{-}cut} w_e)$
3. flip the membership of u.

local max in $O(W)$ iterations — input size $\log W$

$\downarrow$ 2-approx; $\hookrightarrow$ pseudo-poly

(A, B) is stable

for $u \in A$, $\sum_{\substack{e=(u,v)\\ v\in B}} w_e \geq \sum_{\substack{e\in(u,v)\\ v\in A}} w_e$

A里的边.

$$2 \sum_{\substack{(u,v)\\ u,v\in A}} w_e = \sum_{u\in A} \sum_{\substack{e=(u,v)\\ v\in A}} w_e \leq \sum_{u\in A} \sum_{\substack{e=(u,v)\\ v\in B}} w_e = w(A,B)$$

类似有 $2 \sum_{\substack{(u,v)\\ u,v\in B}} w_e \leq w(A,B)$

$$\sum_{e\in E} w_e = \sum_{\substack{(u,v)\\ u,v\in A}} + \sum_{\substack{(u,v)\\ u,v\in B}} + \sum_{e\in\delta(A,B)}$$

$$\leq \frac{1}{2} w(A,B) + \frac{1}{2} w(A,B) + w(A,B)$$

我们 $\Rightarrow$ $w(A,B) \geq \frac{1}{2} \sum_{e\in E} w_e \geq \frac{OPT}{2}$

算法得到的割边和

fast?

Idea. update only when there is a big improvement.

flip a node u only when it increases $w(A,B)$ by a fraction of at least $\frac{\varepsilon}{|V|}$

$(1+\frac{\varepsilon}{n})^{\frac{n}{\varepsilon}} \geq 2$

$w(A', B') \geq (1+\frac{\varepsilon}{n}) w(A,B)$

循环 $\frac{n}{\varepsilon}$ 次, W翻倍

$O(\frac{n}{\varepsilon} \cdot \log W)$ iterations ?

$N(A,B) = \{ (A', B') \mid (A', B')$ can be obtained from (A,B) by fliping ~~1~~ node$\}$

$\downarrow$ k (邻域变大)

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

Kermighan and Lin (1970)

$(A, B) \rightarrow \{ (A_1, B_1)\ \ (A_2, B_2) \cdots (A_{n-1}, B_{n-1}) \}$

1. 找改变之后最大的点. [2] 从未被改变点.挑.最大 $O(n^2)$ 找neighbor

而且 neighbor范围广

近似算法

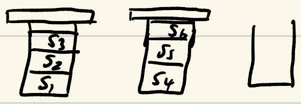
1. all instances
2. polynomial time
3. optimal solution

Binpack Problem

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

Output: packing the items using fewest bins with unit capacity

Next Fit



$B_1 \cdots B_k$

$S(B_i)$

$$\left.\begin{array}{l} S(B_1) + S(B_2) > 1 \\ S(B_2) + S(B_3) > 1 \\ \cdots \\ S(B_{k-1}) + S(B_k) > 1 \end{array}\right\} \to S(B_1) + 2\sum_{i=2}^{B_2} B_i + S(B_k) > k-1$$

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

$$\Rightarrow OPT > \frac{k-1}{2}$$

$$NF = k \begin{cases} k = 2m & OPT \ge m \\ k = 2m+1 & OPT \ge m+1 \end{cases} \Rightarrow \frac{NF}{OPT} \le 2$$

2-approx alg, it has an approx ratio of (at most) 2.

Given an alg A, if for any instance I, $\max\left\{\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right\} \le \rho(|I|)$ ······ 最大/最小

we say A is a $\rho(n)$-approx alg.

$\to$ absolute approx ratio

$2n$ $\left(\frac{1}{2}\ \delta\ \frac{1}{2}\ \delta\ \cdots\cdots\right)$ $(\delta < \frac{1}{n} \to 0)$

$NF: = n$

$OPT: = \frac{n}{2} + 1$ $\quad \frac{NF}{OPT} = 2 - \frac{1}{n+2} \to 2$

AnyFit:

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

put an item i into it.

如何选择?

AnyFit

FirstFit BestFit WorstFit ····

0.7 0.5 0.4 0.1

FF

0.1 0.4

0.7 0.5

BF

0.1

0.4

0.7 0.5

Theorem. $BF(I), FF(I) \le 1.7\,OPT(I)$ for any I (tight)

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

FF decreasing = sort + FF

BF ········ = ········ BF

渐进
asympototic approx ratio

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

tight

$$\frac{FFD(I)}{OPT(I)} \le \frac{\lfloor \frac{11}{9} OPT(I) + \frac{6}{9} \rfloor}{OPT(I)} \overset{?}{\le} \frac{3}{2}$$

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

Theorem: For any binpacking problem
no poly-time alg can achieve an approx ratio
better than $\frac{3}{2}$ unless $P = NP$
no online alg is better than $\frac{5}{3}$.

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 tot value.

暴力: $O(nC)$ $O(nV)$ $(V = \sum_i v_i)$ 伪多项式

Fractional version

A_1 greedy on $\frac{v_i}{w_i}$

Integral version (NP-hard)

A_2 greedy on v_i

	item	value	weight	
$C = 10$	$1 \sim 10$	9	1	$A_2(I) = 10$
	11	10	10	$OPT = 90$

$A^*(I)$

1. run A_1 and A_2 on I
2. return better of $A_1(I)$ and $A_2(I)$

Theorem. A^* has an approx ratio of 2

Proof: I.

$OPT_{Frac}(I)$ $A_1(I)$

$$A^*(I) \ge \begin{cases} A_1(I) \ge OPT_{FRAC}(I) - V_{max} \\ A_2(I) \ge V_{max} \end{cases}$$

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

$O(nV)$ $V = \sum_i v_i \le n \cdot V_{max} \Rightarrow O(n^2 V_{max})$ → pseudo

$v_1 \cdots v_n$ $d = \gcd(v_1 \cdots v_n)$ $v_1/d \cdots\cdots v_n/d$

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

最优解对应集合一样

$$d=\frac{\delta\cdot v_{max}}{n}$$

$$\hat{v}_i=\left\lceil\frac{v_i}{d}\right\rceil \qquad \hat{v}_{max}=\left\lceil\frac{v_{max}}{d}\right\rceil=\left\lceil\frac{n}{\delta}\right\rceil=O\left(\frac{n}{\delta}\right)$$

scaling

$$O(n^2\hat{v}_{max})=O\left(\frac{n^3}{\delta}\right)$$

S

$$v(S)=\sum_{i\in S}v_i$$

$$\hat{v}(S)=\sum_{i\in S}\hat{v}_i=\sum_{i\in S}\left\lceil\frac{v_i}{d}\right\rceil\ge\frac{v(S)}{d}$$

$$\wedge|$$

$$\frac{v(S)}{d}+|S|=\frac{v(S)}{d}+n$$

代入d

$$v(S)+\delta v_{max}=v(S)+n\cdot d\ge d\cdot\hat{v}(S)\ge v(S)$$

OPT under $\hat{v}_i$ — $\hat{S}$ — $v(\hat{S})$

OPT under v_i — S^* — $v(S^*)$

代入$\hat{S}$: $v(\hat{S})+\delta v_{max}\ge d\hat{v}(\hat{S})$ 代入S^*: $d\cdot\hat{v}(S^*)\ge v(S^*)$

$d\cdot\hat{v}(\hat{S})\ge d\hat{v}(S^*)$ （∵ $\hat{S}$是$\hat{v}$最优解）

$$\Rightarrow\begin{cases}v(\hat{S})+\delta v_{max}\ge v(S^*)\\ v(S^*)\ge v_{max}\end{cases}$$

$$\Rightarrow v(\hat{S})+\delta v(S^*)\ge v(S^*)$$

$$\Rightarrow v(\hat{S})\ge(1-\delta)v(S^*)$$

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

近似比可以调节

Definition 11.2 *(PTAS) A polynomial-time approximation scheme (PTAS) is a family of algorithm $\{A_\epsilon\}$, where there is an algorithm for each $\epsilon>0$, such that $\{A_\epsilon\}$ is a $(1+\epsilon)$-approximation algorithm, and its running time is polynomial in the size n of its input instance.*

Definition 11.3 *(FPTAS) A polynomial-time approximation scheme (PTAS) is a family of algorithm $\{A_\epsilon\}$, where there is an algorithm for each $\epsilon>0$, such that $\{A_\epsilon\}$ is a $(1+\epsilon)$-approximation algorithm, and its running time is polynomial in the size n and $1/\epsilon$.*

PTAS：近似比可调

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

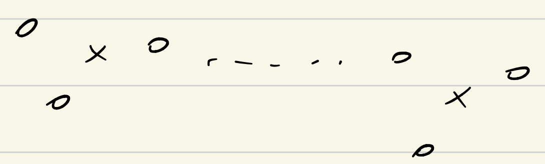
$O(f(\frac{1}{\varepsilon})\cdot poly(n))$ efficient PTAS

$O(\frac{n^3}{\varepsilon})\to$ FPTAS

$O(poly(\frac{1}{\varepsilon})\cdot poly(n))$ FPTAS

---

k-center Problem



Input: n site $s_1 \cdots s_n$

and an integer k

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

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

$dist(x,C)=\min_{y\in C}dist(x,y)$ （x到点集C）

$r(C)=\max_{x}dist(x,C)$

find a set of k center $\overset{C}{}$ to min $r(C)$

$k=1$:

select one site as the center.

site o —r→ o site $r^* \geq \frac{r}{2}$ 2-approx

Assume we know OPT r^* $[0, d_{max})$ 二分距 $\log_2 d_{max}$ 次

已知 r^*

while there exists some sites

pick an arbitrary one as a center

remove all sites within $2r^*$ from the center

C: $r(C) \leq 2r^*$ 但 $|C|$?

assume $|C| \geq k+1$

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

⇓

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

不知 r^* Greedy $(s_1, s_2 \cdots s_n, k)$

选 k 个

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

下证: $r(C_k) \leq 2r^*$

Observation: $C_k = \{c_1, c_2 \cdots c_k\}$

$dist(c_i, c_j)$? $r(C_k)$

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

② $C_k = \{a_1, a_2 \cdots a_k\}$

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

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

反证: Assume $r(C_k) > 2r^*$

k site (center) + 1 site ($r(C_k)$) $\rightarrow r(C_k) =$ —

k 个点覆盖上述 k+1 个点.

$\therefore r^* \geq \frac{r(C_k)}{2} > r^*$

$\alpha \geq 2$ unless $P=NP$

hardness complexity

easy sum $O(n)$

hardest: undecidable (incomputable)

e.g. Halting problem

Given a problem P and an input X.

does P halts on X?

Assume $\exists$.

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

Diagonal(P)

1. if Halt(P, P) 看作string

2. go to step 1

External Sort

External memory model

CPU

通常假设 $(M \geq 3B)$

Internal memory — 参数 M cells (items)

I/O a block: B cells

External memory (disk) — tape 1 …. infinite

2 ….

⋮

→ 只支持 sequential access (不能跳)

I/O 很慢 I/O cost: #blocks moved (tot)

N: input size (# items in the tape) $N \gg M$.

Scan $a_1, \ldots\ldots a_n$

I/O cost: $O(\frac{N}{B})$ → linear time cost ($O(N)$ 不是 linear) $\frac{N}{B}$ 提

one pass: 所有数据扫一遍

Sorting

要用多个 tape: 方便合并

2-way merge: # passes: $1 + \lceil \log_2 \frac{N}{M} \rceil$ 有 $\frac{N}{M}$ 个 runs, 每次

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

passes↓ ← k-way merge: #passes: $1 + \lceil \log_k \frac{N}{M} \rceil$

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

k合一 k { tape 1 … tape k

k blocks

2指用来放输出(交替)

$(k+2)B \leq M$

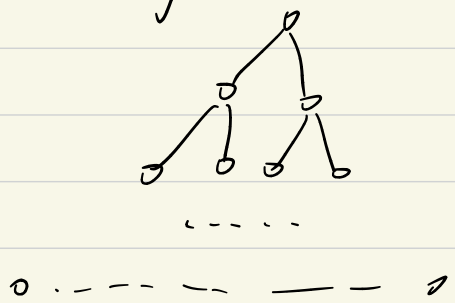
$\Downarrow$

$k \leq \frac{M}{B} - 2$

pass: $1 + \log_{\frac{M}{B}-2} \frac{N}{M}$ cost: $O(\frac{N}{B} \cdot \log_{\frac{M}{B}-2} \frac{N}{M})$

searching

N nodes



最坏: 点都在不同 blocks

$O(\log_2 N)$

↓

B+树

---

avg $= 2M$

Longer run (run少, pass少) 尽可能多读, 直到不行为止 replacement selection

2 4 5 15 huffman tree

2k tapes ×

Fibnacci: $\overset{F_0}{0}$ $\overset{F_1}{1}$ 1 2 3 $F_N = F_{N-1} + F_{N-2}$ $(N \geq 2)$

T_1: F_N → F_{N-1} → ⋯⋯ → 1

F_{N-2} 0 $O(\log_{\frac{\sqrt5+1}{2}} F_N)$ $\log_2 F_N$

0   0   时间↑, 但 tape↓

T_1. $F_{n-1} + \cdots + F_{n-k}$

T_2 $F_{n-1} + \cdots + F_{n-k+1}$

⋮   ⋮   polyphase merge

T_k F_{n-1} k+1 tapes

T_{k+1} 0

$$F_n = F_{n-1} + F_{n-2} + F_{n-3}$$

F_n 0

0 → F_{n-1} →

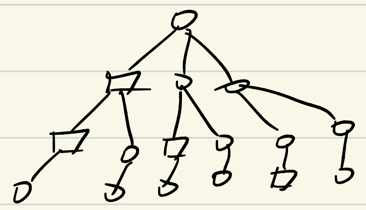
0   F_{n-2}

0   F_{n-3}

$F_{n-1} = F_{n-2} + F_{n-3} + F_{n-4}$

$F_{n-2} = \quad F_{n-3} +$

backtracking



o: good □: bad

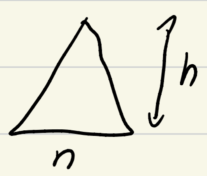
find <u>a good path</u> from root to a leaf

o→o→o→o→o

DFS ✓ or BFS

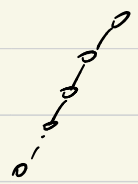
```
dfs(u)   // find a good path from u to a leaf
    if u is bad
        return None.
    else.  // u good
        if u is a leaf
            return u
        else
            for each child v of u
                path = dfs(v)
                if path ≠ None
                    return u→path
```

dfs + pruning = backtracking

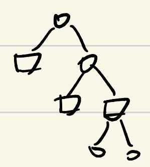


worst case $\Theta(n)$

best case : $O(h)$



or



N Queens Problem

Given a $n \times n$ chessboard

find a <u>feasible</u> placement of n queens

↳ no 2 queens attack (same row/col/digonal)

Fact .

1. for $n > 3$. a feasible placement always exists
2. for special n (prime, $6k+1$ ...)

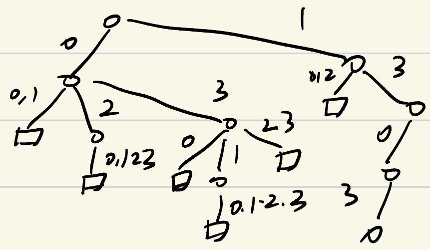
   efficiently solved
3. for general n

   NP-hard

bruteforce $O(n!\,n^2)$?

backtrack worst $O(n!\,n)$

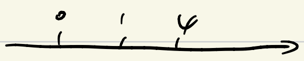
	0	1	2	3
0		0		
1				0
2	0			
3			0	



NQ(n).

```
P[i] = -1 for i=0 to n-1
i=0
while i>=0 and i<n
        findgood = false
        for j=P[i]+1 to n-1        --> 之前放到P[i]
            if j cannot attack P[0].P[1]--.P[i-1]
                P[i] = j;
                findgood = true.
        if findgood = true.
            i = i+1
        else // findgood = false
            i = i-1
    if i==n :
        P is a feasible placement
    if i==0 :
        no feasible placement
```

Turnpike problem



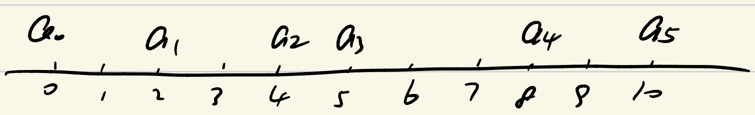
$A=\{0,1,4\}$ $D(A)=\{1,3,4\}$

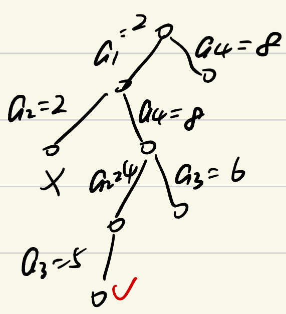
$=\{0,1,2\}$ $\cdot\{1,1,2\}$ → multiset $|D(A)|=\frac{|A|^2-|A|}{2}$

Given D (multiset) find $A=\{a_0\le a_1\cdots\le a_{n-1}\}$ s.t. $D(A)=D$

↓ assume $a_0=0$

$D=\{\cancel{1},\cancel{2},\cancel{2},\cancel{2},\cancel{3},\cancel{3},\cancel{4},\cancel{4},\cancel{5},\cancel{5},\cancel{6},\cancel{6},\cancel{8},\cancel{8},\cancel{10}\}$ $|D|=15 \Rightarrow |A|=6$





$A=\{0,2,4,5,8,10\}$ 不唯一

1. for every d remaining in D, at least one of its nodes not determined

2. for maximum d remaining in D. at least one of its endpoints is a_0 or a_{n-1}

Input: multiset A
Output: multiset B

```
A = {0, max(D)}
D = D - {max(D)}
TP(D, A)        决定中间点
```

```
TP(D,A)
if D empty                              } O(1)
    return true
d = max(D)                              max time
for a* = d - a0 or max(A) - d
    Δ = {dist(a*, a) : a ∈ A}           O(n)
    if Δ ⊆ D                            n·findkey
        D = D - Δ                       n·del time
        A = A ∪ {a*}                    O(1)
        if TP(D, A)                     ) O(1)
            return true
    else
        D = D ∪ Δ                       n·ins time
        A = A - {a*}                    O(1)
return false
```

O(n) + max time + n findkey + n·del + n·ins

time per node O(nlogn) BST

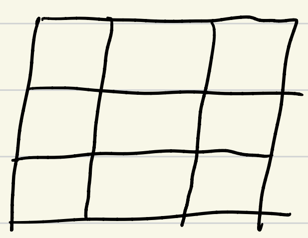
someone proves

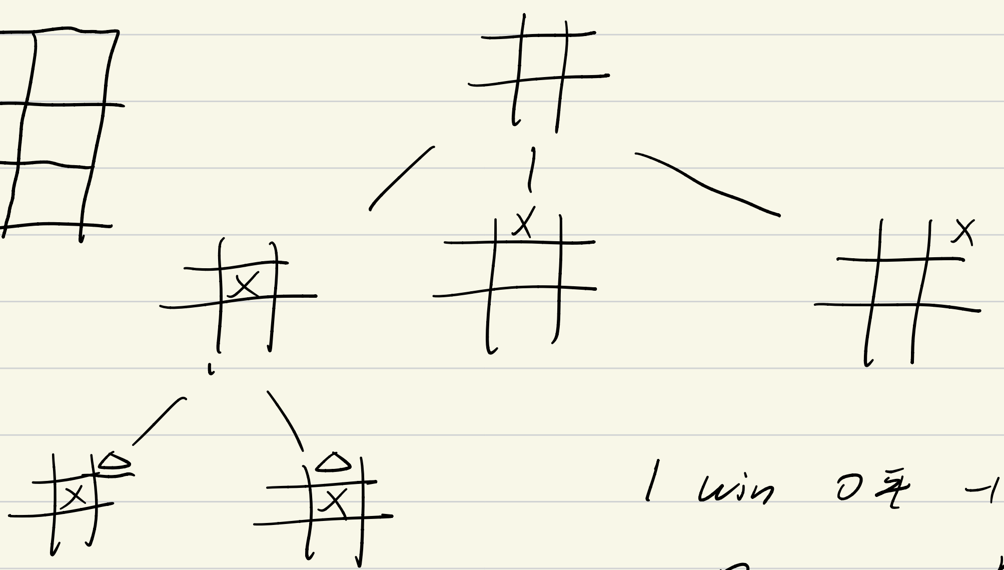
worth case: $\Theta(2^n)$ nodes   rare

best case: $O(n)$ nodes   most instances

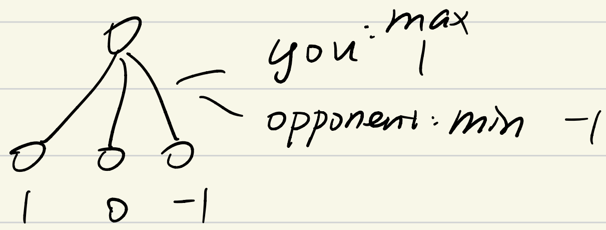
Game

tic-tac-toe





1 win 0平 -1 lose

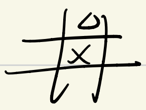


回溯一遍！ 无法快速得到 ☹

布局

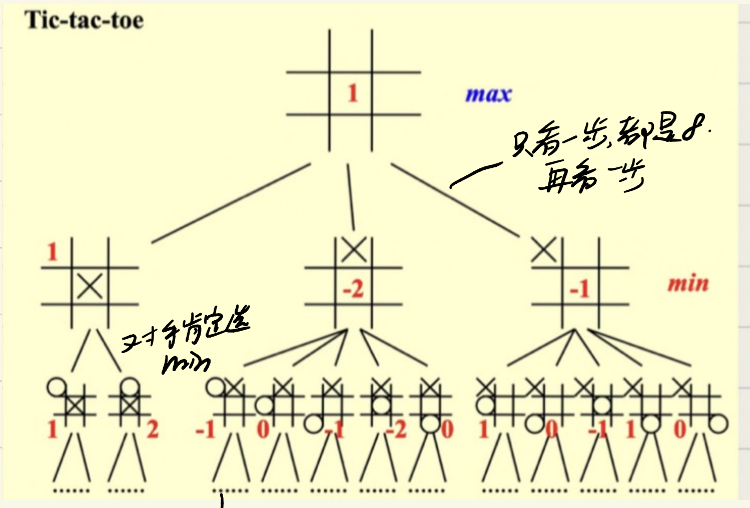
$f(p) = W_{you} - W_{opponent}$

#potential wins



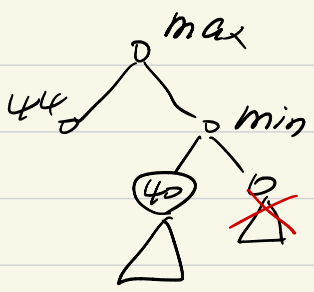
$W_{you} = 6$ （假设对手不走）

$W_{opponent} = 4$　　$f(p) = 2$

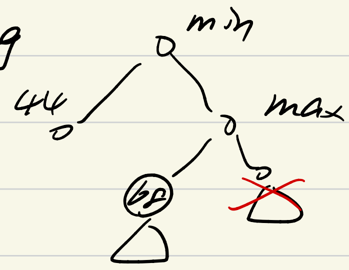


这-1，-2子树不可能大于-1，比1小，剪枝！

α pruning



β pruning



Weighted Independent Set on A Path

Input $v_1 - v_2 - \cdots - v_{n-1} - v_n$ with weight $w_1 \cdots w_n$

Output: an independent set S with maximum weight

$\underbrace{\text{independent set}}_{\text{a subset of vertices s.t. no two are connected by an edge}}$

$v_1 - v_2 - v_3 - v_4$

$1 \quad 4 \quad 5 \quad 4$

$opt(G_3) = 6$

$opt(G_2) = 4$ $\quad opt(G_4) = \begin{cases} 6 \\ 4+4=8 \end{cases}$

Input $v_1 - v_2 - \cdots - v_{n-1} - v_n$

Case 1. $v_n \notin S^*$ $\quad S^* = OPT$ for G_{n-1}

G_{n-1}: $v_1 - v_2 - \cdots - v_{n-1}$

Case 2. $v_n \in S^*$ $\quad S^* = \{v_n\} \cup opt$ for G_{n-2}

v_{n-1}选不了

Subproblems

for $i \in [0, n]$, define

$c[i]$ = total weight of opt for G_i

$c[n] = \max\{c[n-1],\ c[n-2] + w_n\}$

Recurrences

$$\begin{cases} c[i] = \max\{c[i-1],\ c[i-2] + w_i\} \text{ for any } i \in [2, n] \\ c[1] = w_1 \\ c[0] = 0 \end{cases}$$

base case

Computing c[ ]

1. recursion

```
recur(i)
    if i==0 or i==1
        return base case
    else if i>=2
        return max{ __ , __ }
```

$T(n) = T(n-1) + T(n-2) + O(1) \Rightarrow T(n) = O(c^n)$

$c[n] \to c[n-1],\ c[n-2]$; $c[n-1] \to c[n-2],\ c[n-3]$

重复计算

2. recursion with memoization

```
global c[0....n]        c[0]=0, c[1]=w1, c[i]=-1, i>1
recur(i)
    if c[i] >= 0
        return c[i]
    else
        c[i] = max{ __ , __ }
        return c[i]
```

$O(n)$

3. Iteration

$C[0]=0, \; C[1]=w_1$

for $i=2$ to n

$C[i] = \max\{ \_\_ , \_\_ \}$ $\quad O(n)$

Reconstructing OPT solution

$C[0] \; C[1] \cdots C[n]$

S^*

if $C[n] == C[n-1]$

$v_n \notin S^*$

else // !=

$v_n \in S^*$ S_n

```
S* = ∅
i = n
while i >= 2:
    if C[i] == C[i-1]:
        i = i-1
    else
        S* = S* ∪ {v_i}
        i = i-2
if i == 1 :
    S* = S* ∪ {v_1}
return S*
```

```
recon(n)
    if n==0 or 1
        base case
    if n>=2
        if C[n] == C[n-1]
            return recon(n-1)
        else
            return {v_n} ∪ recon(n-2)
```

$O(n)$

Dynamic Programming

1. define subproblems
2. finding recurrence
3. computing the optimal value for (sub)problems
4. reconstructing the optimal solution

Knapsack Problem

Input: n items with weight $w_1 \cdots w_n$
and values $v_1 \cdots v_n$
capacity C

Output: a subset of items with $\max \sum_{i \in S} v_i$ s.t. $\sum_{i \in S} w_i \le C$

case 1: $n \notin S^*$ $\quad S^* :=$ opt for first $n-1$ items with total weight $\le C$

case 2. $n \in S^*$ $\quad S^* = \{n\} +$ opt for first $n-1$ items with total weight $C - w_n$

Subproblems:

for $i \in [0, n]$, for $c \in [0, C]$
define $V[i, c]$ be the max value of a subset of first i items with total weight at most c.

Recurrence $V[n, C] = \max\{V[n-1, C],\ V[n-1, C-w_n]+v_n\}$

for any $i \in [1, n]$, for any $c = [0, C]$

$$\begin{cases} V[i, c] = \max\{V[i-1, c],\ v_i + \underbrace{V[i-1, c-w_i]}_{\text{may} < 0}\} \\ V[0, c] = 0 \quad \text{for } c \in [0, C] \\ V[i, c] = -\infty \quad \text{for } c < 0 \end{cases}$$

Compute $V[\,][\,]$

```
V[0, c] = 0 for c ∈ [0, C]
for i = 1 to n
    for c = 0 to C
        if wi > c
            V[i, c] = V[i-1, c]
        else
            V[i, c] = max {_ , _}
return V[n, C]
```

time $O(nC)$

space $O(nC)$

Reconstructing OPT sol

```
c = C
S = ϕ
for i = n to 1
    if c ≥ wi and V[i, c] = V[i-1, c-wi] + vi
        S = S ∪ {vi}
        c = c - wi
return S
```

time $O(n)$

space $O(nC)$

Remark

time : $O(n\underbrace{C}_{\text{pseudo-polynomial time}})$ → $\log_2 C$ bits to represent C 关于值 ✓ 规模 ×

space : if we only care about OPT values

$O(n+C)$

if requires OPT sol

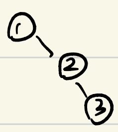
$O(nC)$ (但也降到 $O(n+C)$)

Optimal BST

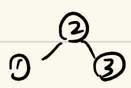
Input: n keys $1, 2, \cdots, n$ with freq $P_1, P_2 \cdots P_n$

Output: a BST with min $\underbrace{\text{avg search time}}_{\sum_{i=1}^{n} P_i(d_i+1)}$

1	0.8
2	0.1
3	0.1

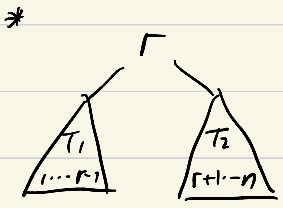


$0.8+0.2+0.3 = 1.3$



$1.6+0.1+0.2 = 1.9$

T^*



search time of k in $T^* = 1 +$ search time of k in T_1

$$\sum_{k=1}^{n} P_k \cdot \text{search time of } k \text{ in } T^* = \sum_{k=1}^{r-1} P_k (\text{search time of } k \text{ in } T_1 + 1)$$
$$+ P_r \qquad\qquad \sum_{k=1}^{n} P_k$$
$$+ \sum_{k=r+1}^{n} P_k (\text{search time of } k \text{ in } T_2 + 1)$$

average search time of T^* $= \sum_{k=1}^{n} P_k +$ average search time in T_1 $\; + \; \cdots\cdots\cdots$ T_2

$c[1,n] \qquad c[1, r-1] \qquad c[r+1, n]$

subproblems

for $i \in [1, n+1]$, $j \in [0, n]$

define $c[i, j]$ be the avg search time of the optimal BST for key $[i \cdots j]$ with freq $P_i \cdots P_j$

不知根节点

$$c[1,n] = \min_{1 \le r \le n} \{ c[1, r-1] + c[r+1, n] + \sum_{k=1}^{n} P_k \}$$

Recurrence

$$\begin{cases} c[i,j] = \min_{i \le r \le j} \{ c[i, r-1] + c[r+1, j] + \sum_{k=i}^{j} P_k \} \\ c[i, j] = 0 \text{ if } i > j \end{cases}$$

Computing $c[\ ,\ ]$

$c[i, i-1] = 0$ for $i = 1$ to n $\qquad O(n^3)$

for $l = 0$ to n

  for $i = 1$ to $n - l$

    $c[i, l+i] = \sum_{k=i}^{i+l} P_k + \min_{i \le r \le j} \{ c[i, r-1] + c[r+1, i+l] \}$

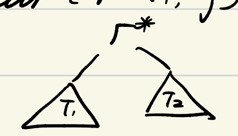
    记录 $r[i, l+i]$

return $c[1, n]$

Reconstruction

recur(i, j)

1. if $i == j$, trivial
2. $r^* = \arg\min \{ c[i, r-1] + c[r+1, j] + \sum_{k=i}^{j} P_k \}$
3. $T_1 = $ recur$(i, r^* - 1)$
4. $T_2 = $ recur$(r^* + 1, j)$
5. return



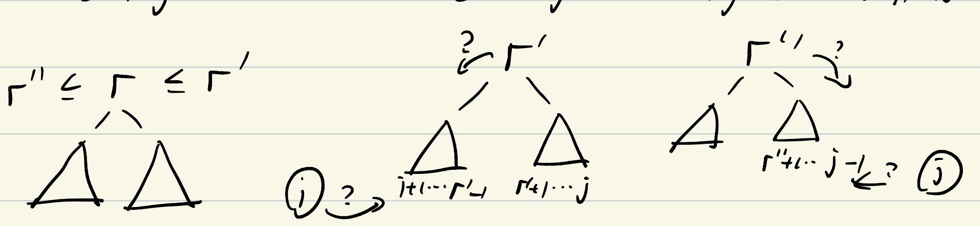
recursive calls: $\Theta(n)$

time for each: $O(n)$

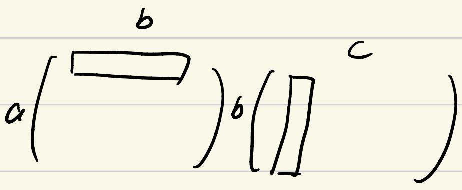
total $= O(n^2) \to O(n)$

$O(n^3) \to O(n^2)$

$c[i, j] \qquad c[i+1, j] \quad c[i, j-1]$ 已经算好了



$M_{a\times b} M_{b\times c}$



$a\times b\times c$

$M_{4\times 3} M_{3\times 2} M_{2\times 1}$ $\quad (4\times 3\times 2)+(4\times 2\times 1) = 32$

$(3\times 2\times 1)+(4\times 3\times 1) = 18$

Input: $\underset{r_0\times r_1}{M_1} \cdot \underset{r_1\times r_2}{M_2} \cdots \underset{r_{n-1}\times r_n}{M_n}$

Output: best order of performing multiplication

b_i = #ways to $\times$ i matrics

$b_1 = b_2 = 1 \quad b_3 = 2 \quad b_4 = b_3 b_1 + b_2 b_2 + b_1 b_3 = 5$

$M_1 \circ M_2 \circ M_3 \circ M_4$

$$b_i = \sum_{j=1}^{i-1} b_j b_{i-j} \qquad b_n = \frac{4^n}{n\sqrt{n}}$$

$\underset{M_{r_0\times r_k}}{(M_1 \cdots M_k)} \Big| \underset{M_{r_k}\times M_{r_n}}{(M_{k+1} \cdots M_n)}$

$O(r_0\times r_k\times r_n)$ + min time mul first k matrices
+ . ― . . . . ― . last $n-k$ ― ― . . .

Subproblem

for $i,j \in [1,n]$

$c[i,j]$ = min cost for perform $M_i \cdots M_j$

$$C[1,n] = \min_{1\le k\le n-1} \{ r_0\cdot r_k \cdot r_n + C[1,k] + C[k+1,n] \}$$

$$\begin{cases} C[i,j] = \min_{i\le k\le j-1} \{ r_{i-1} r_k \cdot r_j + C[i,k] + C[k+1,j] \} \\ C[i,i] = 0 \quad \text{for } i \in [1,n] \end{cases}$$

Input: a set of n activities $\underset{(s_1,f_1)}{I_1} \cdots \underset{(s_n,f_n)}{I_n}$

Output: a max set S of compatible activities

max total weight