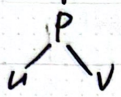
Leftist heap & Skewed heap

Binary heap

1. heap property  $\Rightarrow p<u$ & $p<v$

2. complete binary tree 

支持操作：

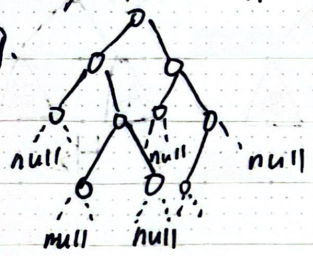
- findmin $O(1)$
- deletemin $O(\log n)$
- insert $O(\log n)$
- given pointers ⟨ decreasekey $O(\log n)$, delete $O(\log n)$
- make-heap $O(n)$

- 现在要求优化 merge (二合一) 操作 - $O(n)$ 太慢 Goal: $O(\log n)$

⇒ 左倾堆 Leftist heap

性质：1. heap property

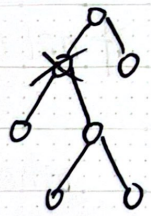
2. binary tree with leftist property

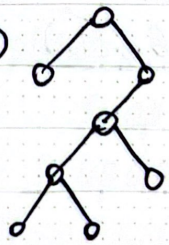
e.g. 

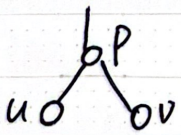
↗ length: npl(u)

the shortest path from u to null

from each u, the right descending path is (one of) the shortest path from u to nu

①  不是

②  是 (每个节点满足左势性质)

npl 关系： 

$npl(p)=\min\{npl(u), npl(v)\}+1$

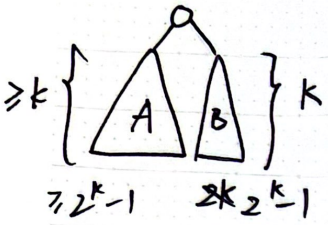
$\rightarrow npl(u)\geq npl(v)$ if leftist.

Lemma: For a leftist heap with n nodes, its right path has at most $\log_2(n+1)$ nodes. ⇔ # nodes in right path = r, # nodes in leftist heap $\geq 2^r$

proof: 归纳法 base $r=1$ $\geq 2^1-1=1$ ✓

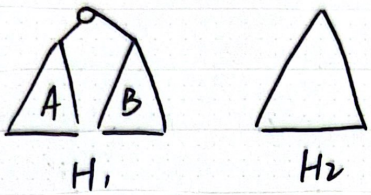
inductive hypothese $r=k$ $\geq 2^k-1$

inductive step $r=k+1$?



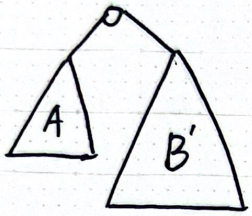
⇒ $\geq 2(2^k-1)+1 = 2^{k+1}-1$

操作：① merge



假设 $H_1.root < H_2.root$

⇒ 合并：



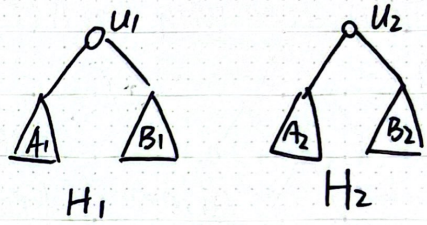
$B' = merge(H_2, B)$

⇒ Swap children if $npl(A.root) < npl(B.root)$

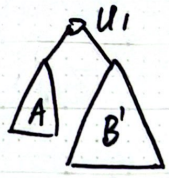
具体思路：

merge (H_1, H_2)

1. if $(u_1 == null)$
   return H_2;
2. if $(u_2 == null)$
   return H_1;



3. if $u_1.key < u_2.key$
   $B' = \underline{merge(B_1, H_2)}$ ⇒ $H =$



if $npl(A) \geq npl(B')$

or

otherwise

递归

update $npl(u_1)$
return H.

else // $u_1.key > u_2.key$ similar

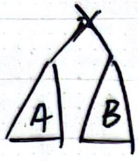
levels of recursion. $\leq r_1 + r_2$ (# nodes on the right path of H_1)

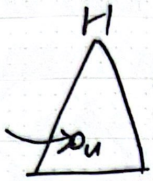
time per level $O(1)$

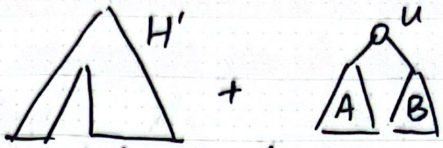
⇒ total time $O(r_1+r_2) = O(\log n_1 + \log n_2) = O(\log n)$

$n = n_1 + n_2$

2. Insertion 看成 merge (H, o) $O(\log n)$

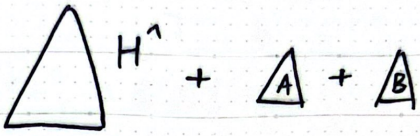
3. Deletemin  ⇒ merge (A, B) $O(\lg n)$

4. Delete  把以u为root的子树 =



↓ fix the leftist property (*)

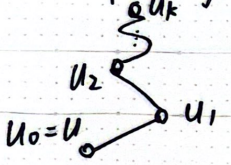
=



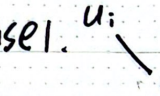
merge =

 $O(\log n)$

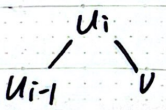
分析(*)步：只有u的 ancestor 需要修复



for i=1 … k :

case1.  ⇒ npl(ui) ↓. min[npl(v), npl(ui-1)]

↳ update npl(ui)

case2. 

(修复完成)

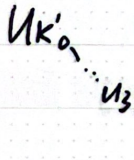
case 2.1 $npl(u_{i-1}) \geq npl(v)$ break;

case 2.2 $npl(u_{i-1}) < npl(v)$

swap children (ui)

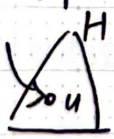
update npl (ui).

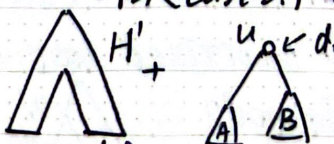
退出前(修复后) u_{i-1} 一定是 u_i 的右孩子：



→ 最长为 $O(\log n)$ → 时间复杂度 $O(\log n)$

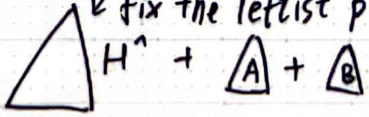
↳ 之后是 Case 2.1 ⇒ break

5. Decreasekey  =



↓ fix the leftist property

=



merge =

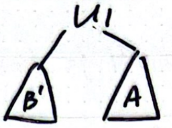
 $O(\log n)$

- findmin $O(1)$
- build $O(n)$ ↑ 常规建堆

Skewed heap — self-adjusting version of leftist heap

merge 操作与 leftist heap 类似，唯一区别：① 合并完之后无条件交换.

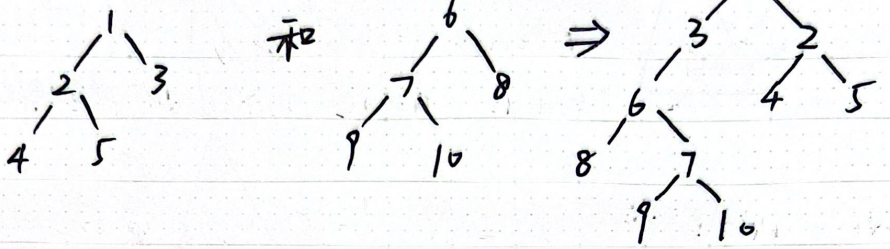
即 $B' = merge(B, H_2) \Rightarrow H =$



仍能去掉~~无法~~保证 $O(\log n)$ 均摊复杂(*)

② 当 u_1 or u_2 == null 时，会 swap child for all nodes on the right path except for the lowest one.

eg. 合并:



insertion } 都是通过 merge 实现 → 均摊意义下 $O(\log n)$

deletemin

delete

decreasekey > not support

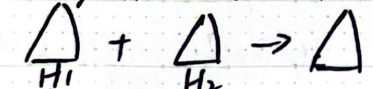
→ 定义 potential function:

Given a node $u \in H$, u is heavy if size(right subtree of u) ≥ size(left subtree of u)

light otherwise.

$\Phi(H)$ = # heavy nodes in H, $\Phi(empty) = 0$, $\forall H, \Phi(H) \geq 0$

考虑 merge:



l_1+h_1 l_2+h_2 ← #light / heavy nodes on the right path of H_1/H_2

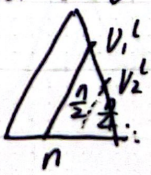
actual cost = $O(l_1+l_2+r_1+r_2)$

$\Phi(H_1)+\Phi(H_2) = h_1+h_2+h$ ← 不在路上的 heavy nodes，merge 后 h 不变

$\Phi(H) \leq h+l_1+l_2$，重的都变轻，轻的可能变重.

$\Delta\Phi \leq l_1+l_2-h_1-h_2$. ⇒ amortized cost = actual + $\Delta\Phi = O(l_1+l_2)$

考虑 right path 上的 light node

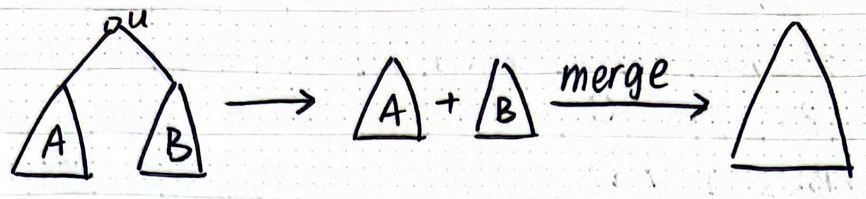


$2^l \Leftrightarrow n$

$\Rightarrow O(\log n_1 + \log n_2)$

$= O(\log n)$

考虑 Deletemin:



actual cost $O(1)$ $\quad O(l_1+l_2+h_1+h_2)$

$\Delta\Phi \le 0$ $\quad O(l_1+l_2-h_1-h_2)$

$\Rightarrow$ amortized $= O(l_1+l_2) = O(\log n)$

(**) actual cost $= O(\#\text{levels of recursion})$

$= O(\#\text{nodes whose children are swapped})$

$l'+h'$

$\Phi(H_1)+\Phi(H_2)=h+h'$, $\Phi(H') \le h+l'$, $\Delta\Phi \le l'-h'$

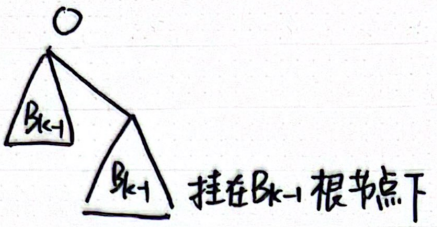
$\Rightarrow$ amortized cost $= O(l') = O(\lg n)$

Binomial heap

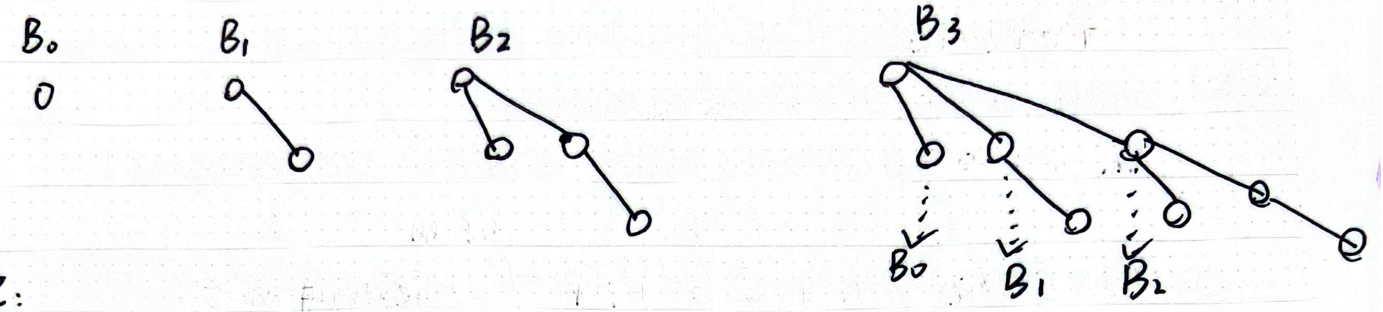
Binomial Tree

$h=0$ B_0

$h=k$ B_k



⇒



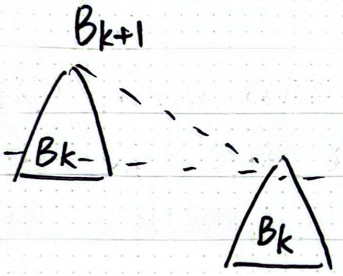
⇒ 结论:

① # nodes in $B_k = 2^k$

② # nodes at level d of $B_k = \binom{k}{d} = \frac{k!}{(k-d)!\,d!}$

proof: 归纳法 - base case $k=0$ true

- assume true for B_k:



d ⇒ $\binom{k}{d} + \binom{k}{d-1} = \binom{k+1}{d}$

□

③ the root of B_k has k subtrees $B_0, B_1, \ldots, B_{k-1}$.

proof: 展开 B_k ⇒ $B_{k-1} + B_{k-1}$

$B_{k-2} + B_{k-2}$

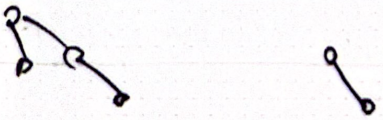
...

A Binomial heap is a forest of

1. binomial trees of distinct height
2. each of which is in heap order

eg. A binomial heap with 6 nodes:

$6 = 2^2 + 2^1 + 0 = 110$ ⇒二进制唯一,形式确定



trees in a binomial heap with nodes $\leq \log_2 n$

└ their height $\leq \log_2 n$ (B_k's height is k)

⇒ find min 遍历根节点找 min $O(\log n) \to O(1)$.

(维护一个指向最小根的指针)

Insertion 对应二进制数+1

insert(H, x)

1. add to H a B_0 with key x
2. i = 0
3. while H has two trees of height i
4. combine them into B_{i+1} (取小的 root)
5. i = i+1

时间复杂度 O(# combine) = O(# trees in H +1) = O(log n)

Merge 对应二进制数加法

merge(H_1, H_2)

1. for i = 0 to max{# trees in H_1, # trees in H_2}
2. if there are more than one trees of height i.
3. combine them into a B_{i+1}

$O(\#\text{ trees in } H_1 + \#\text{ trees in } H_2) = O(\lg n_1 + \lg n_2) = O(\lg n)$, n_1+n_2

DeleteMin 对应减法

delMin(H)

1. find the B_k that contains the min $O(1)$
2. $H = H - B_k$ $O(1)$
3. $H' = B_k$ - root $O(k) = O(\log n)$
4. merge(H, H') $O(\log n)$

⇒ $O(\log n)$

Decreasekey (x, t) 不断向上换 $O(k) = O(\log n)$

Deletion ⇒ decrease key ($x, -\infty$) + delmin $O(\log n)$

Make heap $(k_1, \dots, k_n)$

1. $H = \phi$
2. for $i = 1$ to n
3. insert (H, k_i)

not tight bound $O(n \log n)$

实际 $O(n)$.

$O(n)$ proof: "starting with empty heap, n consective insertions take $O(n)$ time in total."

法1: 总时间为 $O(\#\text{combine}) = O(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + \frac{n}{2^{\log_2 n}}) = O(n)$

($\frac{n}{2}$: B_0+B_0; $\frac{n}{4}$: B_1+B_1)

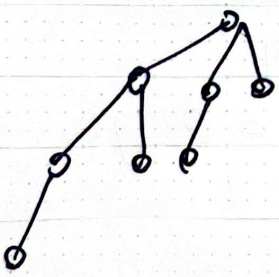
法2: 定义势函数 $\Phi(H) = \#\text{trees in } H$, $\Phi(\text{empty}) = 0$, $\Phi(H) \geq 0$

actual cost: $1 + \#\text{combines}$

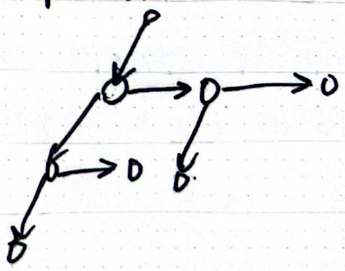
$\Delta\Phi = +1 - \#\text{combines}$

⇒ amortised cost: 2 ⇒ $O(n)$

代码实现: 链表



⇒ left-child-next-sibling



Inverted File Index

① Hashing 快
BST 支持range query

② ⇒合并多个链表同些数据:

n_1 0→0→0
n_2 0→0→0→0 > $O(n_1+n_2)$ > $O(n_1+n_3)$ > ...
...
n_k 0→0→0→0→0..→0

先从小的开始合.

$\Rightarrow O(kn_1+\sum_{i=1}^{k} n_i) = O(\sum_{i=1}^{k} n_i)$

③ Term-partitioned index —— 查找效率高
Document-partitioned index —— 鲁棒性好

④ 筛选文档: - 仅输出高权重文档
- 低频词权重更高

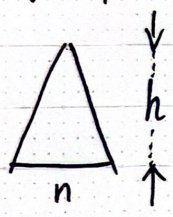
⑤ 评价搜索引擎: - 索引建立速度
- 查询速度
- 支持查找类型.

BackTracking

backtracking 回溯法

```
dfs(u): // find a good path from u to a leaf
1. if u is bad
       return None.
2. else // u is good
3.     if u is a leaf
4.         return u.
5.     else
6.         for each child v of u
7.             path = dfs(v)
8.             if path ≠ None:
9.                 return u → path.
```

⇒ dfs + pruning (剪枝) = backtracking



worst case: $\Theta(n)$

best case: $O(h)$

难点1. 构造树

难点2. 循环 or 递归. 选择

回溯法例题:

1. N Queens Problem

Given a $n \times n$ chessboard,

find a feasible placement of n queens

↓

no two queens can attack each other

same row or same col or digonal

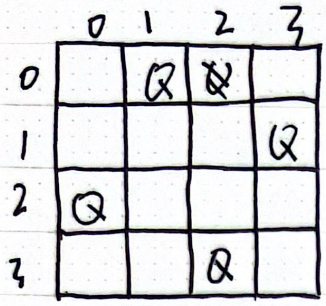
Fact:

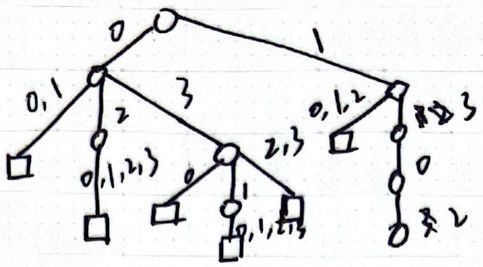
1. For $n > 3$, a feasible placement always exist
2. For some special n (Prime. $6k+1$) efficiently solved.

3. for general n, NP hard.

⇒ bruteforce $O(n! n^2)$

backtrack $O(n! \cdot n)$





NQ(n) ↳第i行Queen位置

```
1. P[i] = -1 for i = 0 to n-1
2. j = 0
3. while i ≥ 0 and i < n
4.         findgood = false
5.         for j = P[i]+1 to n      (前面都试过失败了)
6.             if j cannot attack P[0], P[1], ..., P[i-1]
7.                 P[i] = j
8.                 findgood = true
9.         if find good = true
10.                i = i+1
11.        else  // findgood = false   ⇒回退
12.                i = i-1
13. if i == n :
14.        P is a feasible placement
15. if i == -1 :
16.        ※ no feasible placement.
```

2. Tumpike Problem:

A = [0, 1, 4] D(A) = [1, 3, 4]

A = [0, 1, 2] D(A) = [1, 1, 2]

↳ multiset

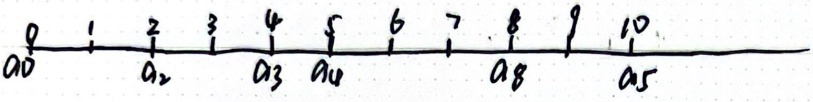
D(A): 距离集合.

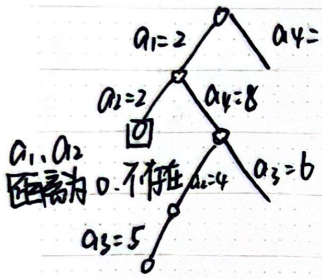
$|D(A)| = \frac{|A|^2 - A}{2}$

$A = [0, 0, 2]$ $D(A) = \{0, 2, 2\}$

Given D, find $A = \{a_0 \le a_1 \cdots \le a_{n-1}\}$, s.t. $D(A) = D$.
(D: multiset; assume $a_0 = 0$)

e.g. $D = \{1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 8, 10\}$ $|D| = 15$
$A = \{a_0 \le a_1 \le \cdots \le a_5\}$ $|A| = 6$
$a_0 = 0$





$\Rightarrow A = \{0, 2, 2, 4, 5, 8, 10\}$

1. for every d remaining in D, at least one of its endpoints is not determined.
2. ☆ for maximum d remaining in D, at least one of its endpoints is a_0 or a_{n-1}

Input: a multiset D
Output: a multiset A s.t. $D(A) = D$
1. $A = \{0, \max(D)\}$
2. $D = D - \max(D)$
3. $TP(D, A)$
   → $TP(D, A)$
   1. if D is empty } $O(1)$
   2. return true }
   3. $d = \max(D)$ max time
   4. for $a^* = d$ or $\max(A) - d$ $2 * \cdots$
   5. $\Delta = \{dist(a^*, a) : a \in A\}$ $O(n)$

6. if $\Delta \subseteq D$ — 对Δ中每一个作findkey ⇒ n·findkey time
7. $D := D - \Delta$ — n·deltime
8. $A := A \cup \{a^*\}$ — 用链表 O(1)
9. if TP (D, A) — O(1)
10. return true
11. else
12. $D := D \cup \Delta$ — n * insertion time
13. $A := A - \{a^*\}$ — O(1)
14. return false. // 两个找a^*都无解.

时间复杂度: 单个节点: $O(n) + maxtime + n * findkey + n \cdot del + n \cdot ins$

要求比较快支持四个操作: avl tree. —— 都是 O(logn)

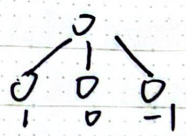
⇒ time per node O(hlogn)

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

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

3. Game Tic-tac-toe 井棋

1 win　0 平　-1 lose

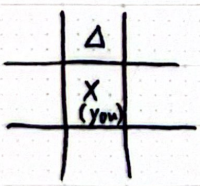


you: max

opponent: min

只有走完才知道 win / lose. ⇒ 只能大致估计布局 $f(P) = W_{you} - W_{oppo}$

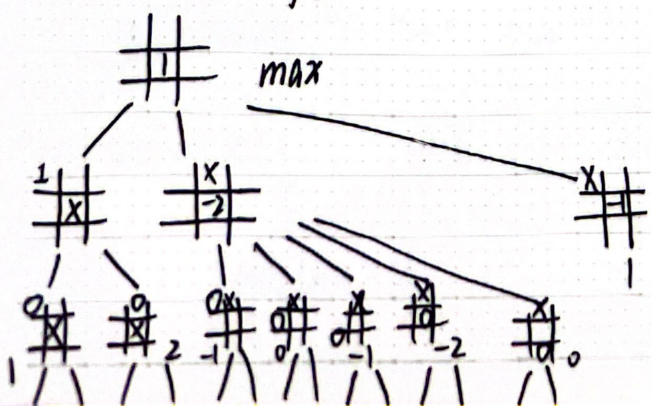
↓ #potential wins



$W_{you} = 6$

$W_{oppo} = 4$

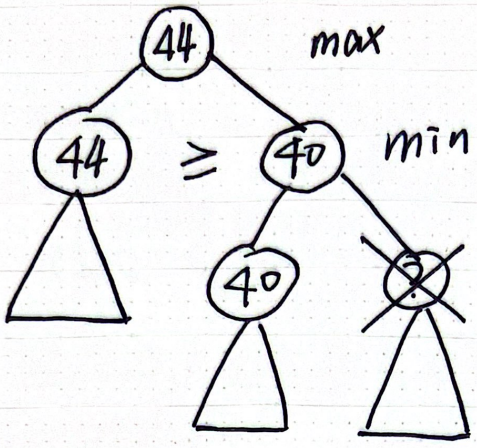
$f(P) = 6 - 4 = 2$.



可以根据max、min剪枝.

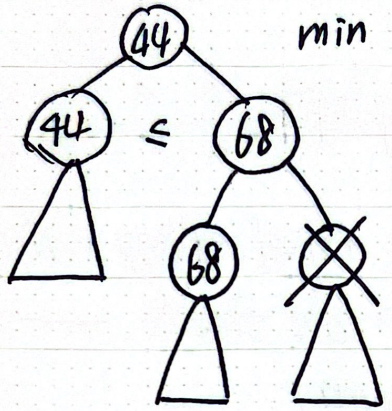
… 略

→ α pruning



← 剪枝. 不需要再判断

→ β pruning



← 剪枝, 不需要再判断

limits the searching to only $O(\sqrt{N})$ nodes (非严谨结论)

Divide and Conquer

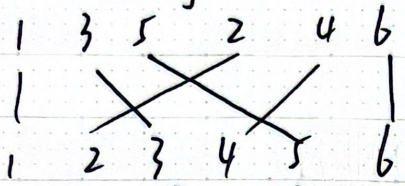
分治法步骤：

① divide　　分成子问题
② conquer　　解子问题
③ combine　　把子解合成总解

eg① counting inversion

定义 inversion：(i, j) is an inversion if $i < j$ and $A[i] > A[j]$

具象化：

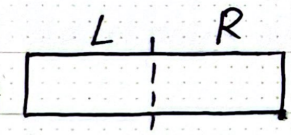


⇒ 交叉点个数代表 inversion 个数

for n elements, 最多 C_n^2 对 inversion

Input: an array A of n distinct integer
Output: the number of inversions in A
brute-force : $O(n^2)$
d&c : $O(n \log n)$

考虑：



- left inversion : $i < j \le \frac{n}{2}$
- right inversion : $\frac{n}{2} < i < j$
- split inversion : $i \le \frac{n}{2} < j$

Count Inv (A)
1. If $|A| \le 2$, trivial.
2. leftInv = Count Inv (left half of A)
3. rightInv = Count Inv (right half of A)
4. splitInv = Count Split Inv (A)
5. return " " + " " + " "

⇐ 递归实现

← 重点.

对 4.　C[1, ..., $\frac{n}{2}$]　　D[1, ... $\frac{n}{2}$]

(排好序的).

SplitInv = 0

i = 1, j = 1

```
for k=1 to n
    if C[i] < D[j]    (or j>n)
        i = i+1
    else          // C[i] > D[j]  (or i > n/2)
        splitInv = splitInv + (n/2 - i + 1)
        j = j+1
return Split Inv.
```

⇒ 两边排好序后O(1)即可完成.

对2,3: 数的同时排序 ⇒ 归并排序

```
Sort-and-CountInv (A)
1. if |A|≤2, trivial
2. (C, leftInv) = Sort-and-CountInv (left half of A)
3. (D, rightInv) = Sort-and-CountInv (right half of A)
4. (B, splitInv) = Merge-and-CountSplitInv (C, D)
5. return (B, " "+" "+" ")
```

```
Merge-and-CountSplitInv (C, D)
1. i=1; j=1; splitInv = 0
2. for k=1 to n
3.     if C[i] < D[j]
4.         B[k] = C[i]
5.         i = i+1
6.     else  // C[i] > D[j]
7.         B[k] = D[j]
8.     splitInv = splitInv + (n/2 - i + 1)
9.         j = j+1                    // 记得处理下标越界
10. return (B, splitInv)
```

$T(n) = 2T(\frac{n}{2}) + O(n)$ ⇒ $T(n) = O(n\log n)$

$T(1) = C$

eg② Closest Pair Problem. → computational geometry

Input: a set of n points on a plane

$p_i = (x_i, y_i)$ (assume $x_i \neq x_j$ and $y_i \neq y_j$ for any $i \neq j$)

Output: the pair (p_i, p_j) with smallest distance $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

考虑 1-D 问题. 排序+遍历最短相邻距离 - $O(n \log n)$

2-D: brute-force - $O(n^2)$

d&c - $O(n \log n)$

1次: 按横坐标将点集左右等分: { left pair / right pair / split pair

ClosestPair(P)

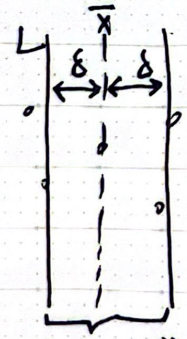
1. if $|P| \leq 3$. trivial.
2. (p_1^l, p_2^l) = Closest Pair (left half of P)
3. (p_1^r, p_2^r) = Closest Pair (right half of P)
4. (p_1^s, p_2^s) = Closest Split Pair (P) - 遍历还是 $O(n^2)$
5. return the closest pair among (p_1^l, p_2^l), (p_1^r, p_2^r), (p_1^s, p_2^s)

对4. $\delta = \min(d(p_1^l, p_2^l), d(p_1^r, p_2^r))$

* Closest Split Pair consider only split pair with distance $< \delta$

↳ $O(n)$ such pair & $O(n)$ time.

首先:



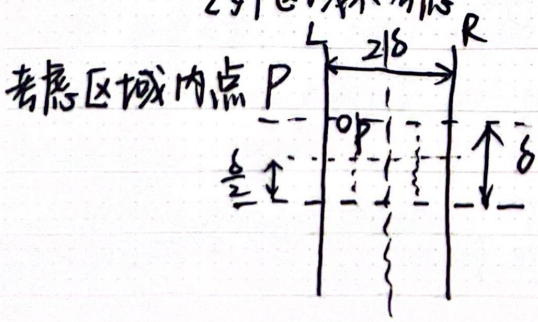
$d(p^l, p^r) < \delta$

$\Rightarrow |x^l - x^r| < \delta$, $|y^l - y^r| < \delta$.

$x^r - \delta < x^l$, $x^r < x^l + \delta$

$\bar{x} - \delta < x^l$, $x^r < \bar{x} + \delta$

之外区域不考虑



考虑区域内点 P

等分成 8 个边长为 $\frac{\delta}{2}$ 的正方形.

每个正方形内最多为一个点(否则 δ 为左右两边分 min 的结论不成立)

⇒ 长方形区域最多 8 个点.

改框架*

假定对纵带内点按 y 坐标排好序 ⇒

~~ClosestSplitPair~~ (Sy) ← list of points in the Strip sorted by y-coordinate $(q_1, q_2, \dots, q_k)$

1. minDist = ∞
2. ClosestSplitPair = None
3. for i=1 to k-1 // 对每个点只需考虑下面最多的7个点.
4. for j=1 to min{7, k-i}
5. if $d(q_i, q_{i+j}) <$ minDist
6. minDist = $d(q_i, q_{i+j})$
7. closestSplitPair = (q_i, q_{i+j})
8. return ClosestSplitPair — O(n)

* ClosestPair (Px, Py)

1. if $|P| \le 3$, trivial
2. Lx = points on left half, sort by x
   Ly .. left .. y
   Rx .. right .. x
   Ry .. right .. y
   } Obtained from Px, Py in O(n) time (已经排好的)
3. (P_1^l, P_2^l) = ClosestPair (Lx, Ly)
4. (P_1^r, P_2^r) = ClosestPair (Rx, Ry)
5. $\delta = \min\{d(P_1^l, P_2^l), d(P_1^r, P_2^r)\}$
6. Sy = points in the strip $(\bar{x}-\delta, \bar{x}+\delta)$, sorted by y - obtained from Py in O(n)
7. (P_1^s, P_2^s) = closestSplitPair (Sy)
8. return the closest pair of ... [3, 4, 7]).

→返回NULL说明? - 所有两边点都很远

$T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$ ⇒ $T(n) = O(n \log n)$

$T(3) = C$

Master ~~Theorem~~ Theory

S&C ⇒ $\begin{cases} T(n) = aT(\frac{n}{b}) + n^d \\ T(1) = C \end{cases}$

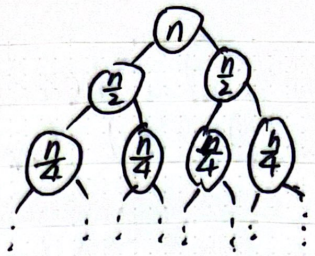
a: # of recursive calls; b: input size shrinkage factor; C: constant

⇒ 如何解这个T(n)?

画图法:

eg1. $\begin{cases} T(n)=2T(\frac{n}{2})+n \\ T(1)=1 \end{cases}$

$\log_2 n$ 层



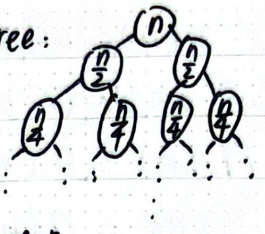
n
n
n

每层一样

$\Rightarrow O(n\log n)$

eg2. $\begin{cases} T(n)=2T(\frac{n}{2})+n^2 \\ T(1)=1 \end{cases}$

recursion tree:



n^2

$(\frac{n}{2})^2\times 2=\frac{n^2}{2}$

$(\frac{n}{4})^2\times 4=\frac{n^2}{4}$

$(\frac{n}{2^i})^2\times 2^i=\frac{n^2}{2^i}$

每层等比例递减

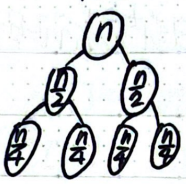
由顶层决定total

$total=\sum_{i=0}^{\log_2 n}\frac{n^2}{2^i}\le 2n^2=O(n^2)$

eg3. $\begin{cases} T(n)=2T(\frac{n}{2})+\sqrt{n} \\ T(1)=1 \end{cases}$

每层等比例递增

由底层决定total



$\sqrt{n}$

$\sqrt{\frac{n}{2}}\times 2=\sqrt{2n}$

$\sqrt{\frac{n}{4}}\times 4=\sqrt{4n}$

$\sqrt{\frac{n}{2^i}}\times 2^i=\sqrt{2^i\cdot n}=(\sqrt{2})^i\cdot\sqrt{n}$

最大 $i=\log_2 n$, $(\sqrt{2})^{\log_2 n}\cdot\sqrt{n}=n$.

$total=\sum_{i=1}^{\log_2 n}(\sqrt{2})^i\cdot\sqrt{n}=\sqrt{n}\cdot\frac{1-(\sqrt{2})^{\log_2 n+1}}{1-\sqrt{2}}=\frac{\sqrt{2}+1}{\sqrt{2}-1}\sqrt{n}\cdot\sqrt{n}=C$

推广得: Four Forms of Master Theorem ⇒

Form 1:

Consider the recurrence $T(n)=aT(\frac{n}{b})+O(n^d)$ with $T(1)=O(1)$

$\begin{cases} \text{If } a=b^d, \text{ then } T(n)=O(n^d\cdot\log n) \\ \text{If } a<b^d, \text{ then } T(n)=O(n^d) \\ \text{If } a>b^d, \text{ then } T(n)=O(n^{\log_b a}) \end{cases}$

Form 2:

Consider the recurrence $T(n)=aT(\frac{n}{b})+f(n)$ with $T(1)=O(1)$

$\begin{cases} \text{If } a\cdot f(\frac{n}{b})=f(n), \text{ then } T(n)=O(f(n)\cdot\log n) \\ \text{If } a\cdot f(\frac{n}{b})=r\cdot f(n) \text{ for some } r<1, \text{ then } T(n)=O(f(n)) \\ \text{If } a\cdot f(\frac{n}{b})=r\cdot f(n) \text{ for some } r>1, \text{ then } T(n)=O(n^{\log_b a}) \end{cases}$

Form 3:

Consider the recurrence $T(n)=aT(\frac{n}{b})+f(n)$ with $T(1)=O(1)$

→

$$\begin{cases} \text{If } f(n)=O(n^{\log_b a}), \text{ then } T(n)=O(f(n)\cdot\log n) \\ \text{If } f(n)=\Omega(n^{\log_b a+\varepsilon}) \text{ for some } \varepsilon>0 \text{ and if } af(\frac{b}{b})\le cf(n). \\ \quad\quad \text{for some constant } c. \text{ then } T(n)=O(f(n)) \\ \text{If } f(n)=O(n^{\log_b a-\varepsilon}) \text{ for some } \varepsilon>0. \text{ Then } T(n)=O(n^{\log_b a}) \end{cases}$$

Form 4:

Consider the recurrence $T(n)=aT(\frac{n}{b})+O(n^d\log^p n)$ ($\to p>0$) with $T(1)=O(1)$.

$$\begin{cases} \text{If } a=b^d, \text{ then } T(n)=O(n^d\cdot\log^{p+1} n) \\ \text{If } a<b^d, \text{ then } T(n)=O(n^d\cdot\log^p n) \\ \text{If } a>b^d, \text{ then } T(n)=O(n^{\log_b a}) \end{cases}$$

另法：猜个（可画图估计）用归纳法证明，对取上下整的情况复加严谨

e.g. $\begin{cases} T(n)=2T(\lfloor\frac{n}{2}\rfloor)+n \\ T(1)=1 \end{cases}$

假设 $T(n)=O(n\log n) \Leftrightarrow \exists c>0, \forall n\ge n_0, T(n)\le c\cdot n\log_2 n$

取 $n_0=2$, $c=2$，归纳：

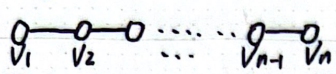
base case $T(2)=2T(1)+2=4\le 2\cdot n\log_2 n \quad (n=2)$

assume $\forall k\in[2,n-1]$, $T(k)\le c\cdot k\log_2 k$

$$\begin{aligned} T(n)&=2T(\lfloor\tfrac{n}{2}\rfloor)+n \\ &\le 2\cdot c\cdot\lfloor\tfrac{n}{2}\rfloor\log_2\lfloor\tfrac{n}{2}\rfloor+n \\ &\le cn\log_2(\tfrac{n}{2})+n=cn(\log_2 n-1)+n\le \text{~~cbo~~} cn\log_2 n \end{aligned}$$

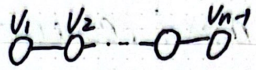
Dynamic Programming

* Weighted Independent Set on a Path

Input: 

with weights: w_1 w_2 w_{n-1} w_n

Output: an independent set S with maximum weight
a subset of vertices s.t. no two are connected by an edge.

Case 1: $v_n \notin S^*$ //最优解未选择 v_n
↳ S^* = opt for G_{n-1} 

Case 2: $v_n \in S^*$ //最优解选择 v_n $\Rightarrow S^* = \{v_n\} \cup$ opt. for G_{n-2}

Subproblems
for $i \in [0,n]$ define $C[i]$ = total weight of opt for G_i
$\rightarrow C[n] = \max\{C[n-1], C[n-2]+w_n\}$

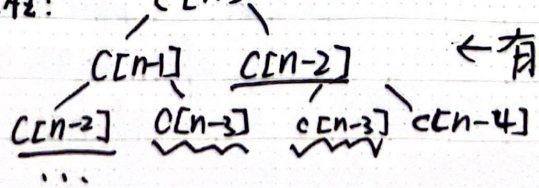
Recurrences
$$\begin{cases} C[i] = \max\{C[i-1], C[i-2]+w_i\} & \text{for any } i \in [2,n] \\ C[1] = w_1 \\ C[0] = 0 \end{cases} \quad \text{> base case}$$

Computing C[]
1. recursion

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

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

考虑计算过程:



←有很多重复计算

2. recursion with memoization

```
global C[0..n]
       C[0]=0, C[1]=w1, C[i]=-1 for i>1
recur(i):
    if C[i]≥0:
        return C[i]
    else
        C[i] = max{recur(i-1), recur(i-2)+wi}
        return C[i]
```

⇒ T(n) = O(n)

3. Iteration

```
C[0]=0
C[1]=w1
for i = 2 to n
    C[i] = max{C[i-1], C[i-2]+wi}
```

⇒ T(n) = O(n)

(通过前三个方法知道了C[i]的值) ⇒

Reconstructing opt solution

```
if C[n] == C[n-1]
        vn ∉ S*
else // C[n] == C[n-2]+wn
        vn ∈ S*
```

with O(n)

用循环:

```
S* = ∅
i = n
while i≥2
    if C[i]==C[i-1]:
        i = i-1
    else // C[i]==C[i-2]+wi
        S* = S* ∪ {vi}
        i = i-2
```

用递归

```
recon(n)
1. if n==0 or 1
2.     base case
3. if n ≥ 2:
4.     if C[n] == C[n-1]
5.         return recon(n-1)
6.     else // case 2
7.         return {vn} // recon(n-2)
```

Dynamic Programming:

1. defining subproblem ← 重点.
2. finding recurrence
3. computing the optimal value for (sub)problems
4. reconstructing the optimal solution

Knupsack Problem:

Input: ① n items with weights $w_1, \dots, w_n$
and values $v_1, \dots, v_n$
② capacity C.

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

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

Subproblems

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

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

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

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

↑ 这是为负数

Computing $V[][]$ time: $O(nC)$, space: $O(nc)$

```
V[0][C] = 0 for c[0,C]
for i = 1 to n
    for C = 0 to C
        if wi > C          // 背包装不下
            V[i][C] = V[i-1][C]
        else
            V[i][C] = max{V[i-1][C], V[i-1][C-wi]+Vi}
```

Reconstructing opt sol

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

time - $O(n)$ space - $O(nC)$

Remark

time: $O(nC)$ → $\log_2 C$ bits to represent C (对应的输入规模) → 指数(相对输入规模)

↘ pseudo-polynomial time 伪多项式 (是相对输入值的多项式)

space: if only cares about opt value

$O(n+C)$ (n: 输入数据; 只需记录当前i和i-1的V[i][c])

if requires opt solution

$O(n+C)$ 分治+dp

Optimal BST

Input: n keys 1, 2, ..., n with

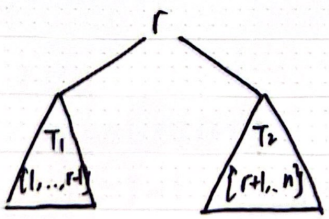
freq. (被查询的频率) $p_1, p_2, \dots, p_n$

Output: a BST with minimum average search time

$\sum_{i=1}^{n} p_i(d_i+1)$

node i 深度

考虑 T^*



(假设 k < r)

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

$\sum_{k=1}^{n} p_k \cdot$ searchtime of k in $T^* =$

$\sum_{k=1}^{r-1} p_k \cdot$ (searchtime of k in T_1 + 1) +

p_r + ($\sum_{k=1}^{n} p_k$) +

$\sum_{k=r+1}^{n} p_k \cdot$ (searchtime of k in T_2 + 1)

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

(T^*: $c[1][n]$; T_1: $c[1][r-1]$; T_2: $c[r+1][n]$)

Subproblems

for $i \in [1, n+1]$, $j \in [0, n]$ 感觉 i,j 反了? (没反). 因为 i>j 要用到.

define $c[i][j]$ be the average search time of the optimal BST for key $[i, \dots, j]$ with freq. $p_i \dots p_j$

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

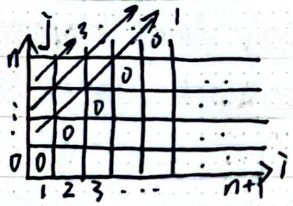
Recurrence $\rightarrow$ $\begin{cases} c[i][j] = \min_{i \le r \le j} \{ c[i][r-1] + c[r+1][j] + \sum_{k=i}^{j} p_k \} \\ c[i][j] = 0 \text{ if } i > j \end{cases}$

computing $c[][]$

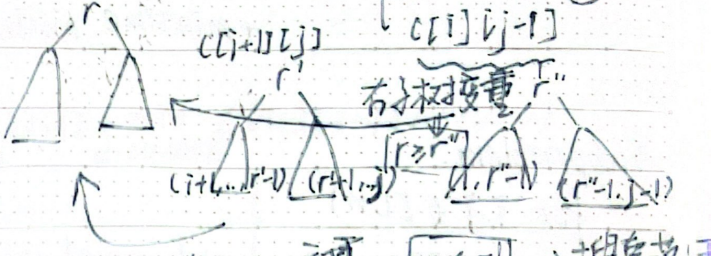
```
c[i][i-1] = 0 for i = 1 to n
for l = 0 to n
    for i = 1 to n - l
        c[i][i+l] = Σ_{k=i}^{i+l} p_k + min_{i≤r≤j(=i+l)} { c[i][r-1] + c[r+1][i+l] }
return c[1][n]
```

$\} O(n^3)$

计算顺序:



算 $c[i][j]$ 的时候, $\begin{cases} c[i+1][j] \\ c[i][j-1] \end{cases}$ 已知.

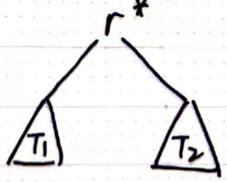


剪枝后 $O(n^2)$

左子树变量 $\Rightarrow r^* \le r'$ ∴搜索范围 $r'' \le r \le r'$

Reconstruction

```
recur(i, j)
1. if i == j, trivial.        // 单点.
2. r* = argmin_{i≤r≤j} { c[i][r-1] + c[r+1][j] + Σ_{k=i}^{j} p_k }
3. T1 = recur(i, r*-1)
4. T2 = recur(r*+1, j)
5. return
```



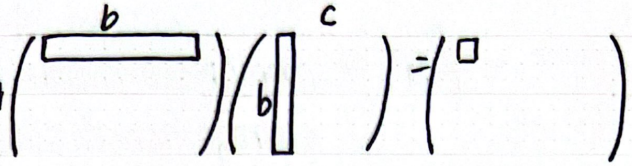
← 在 computing 算过记录即可 $r[i][j] \rightarrow$ 算 $c[i][j]$ 时记录的根节点.

recur(1,n)

recursive calls: $\Theta(n)$ // 每次递归确定一个节点.

$\rightarrow$ total: $\Theta(n^2)$

矩阵乘法

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

正常计算：$a\times b\times c$

考虑 $M_{4\times3}\cdot M_{3\times2}\cdot M_{2\times1}$

先乘前两个：$4\times3\times2+4\times2\times1=32$

先乘后两个：$3\times2\times1+4\times3\times1=18$.

Input: $M_1\cdot M_2\cdot M_3\cdots M_n$

$r_0\times r_1 \quad r_1\times r_2 \quad r_2\times r_3 \quad r_{n-1}\times r_n$

Output: best order of performing multiplication

b_i := # ways to multiply i matrices

$b_1=1$, b_2 非, $b_3=2$, $b_4=b_3b_1+b_2b_2+b_1b_3=5$

$M_1\cdot M_2\cdot M_3\cdot M_4$

( )( ) $b_3\cdot b_1$

{)( ) $b_2\cdot b_2$

( )(} $b_1\cdot b_3$

$\Rightarrow b_i=b_{i-1}b_1+b_{i-2}b_2\cdots b_1b_{i-1}$

$b_n=\frac{4^n}{n\sqrt{n}}$ 枚举较慢. → 如何找动规?

考虑 M_1到M_n 分成 $(M_1\cdots M_k)(M_k\cdots M_n)$

$M_{r_0\times r_k}$ $M_{r_k\times r_n}$

$\Rightarrow O(r_0\times r_k\times r_n)$ + minimum time multiply first k matrices + mintime multiply last ☆ n-k matrices

Subproblem.

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

c[i][j] = min cost for perform $M_i\cdot M_{i+1}\cdots M_j$

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

$\Rightarrow$ $\begin{cases} c[i][j]=\min\limits_{i\le k\le j-1}\{r_{i-1}r_kr_j+c[i][k]+c[k][j]\} \\ c[i][i]=0 \text{ for } i\in[1,n] \end{cases}$

最短路问题

Input: a directed graph $G=(V,E)$ with edge cost $c: E \to \mathbb{Z}$ and a source $s \in V$

Output: the cost $\delta(s,v)$ of the shortest path from s to v for every $v \in V$

Dijkstra's $O(|V|\log|V| + |E|)$ assume $\forall e,\ c(e) \geq 0$
(有负边会出错)

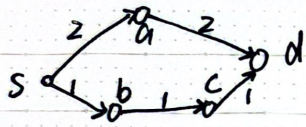
Bellman-Ford if $\exists e,\ c(e) < 0$ Dynamic Programming
assume G has no negative cycle

Subproblem

for $i \geq 0$, for every $v \in V$

let $c[v][i]$ be cost of the shortest path from S to V with *at most* i edges ($c[v][i] = +\infty$ if no such path exists)

e.g.



$c[d][0] = +\infty$ $c[d][2] = 4$

$c[d][1] = +\infty$ $c[d][3] = 3$

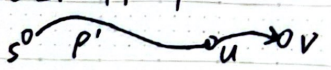
Recurrence

$c[v][i]$ ⇒ 解为 p^* $|p^*| \leq i$ (至多用 i 条边)

case 1. $|p^*| \leq i-1$

$c[v][i] = c[v][i-1]$

case 2. $|p^*| = i$



$c[i][j] = \min_{u:(u,v)\in E} c[u][i-1] + c(u,v)$ ☆

⇒ $c[v][i] = \min \left\{ c[v][i-1],\ \min_{u:(u,v)\in E} c[u][i-1] + c(u,v) \right\}$

base case:

$c[s][0] = 0$

$c[v][0] = +\infty$ for $v \neq s$

Compute:

1. $c[s][0] = 0$
2. $c[v][0] = +\infty$ for $v \neq s$
3. for $i = 1, 2, \ldots |V|-1$ // 不存在负环 ⇒ 无环

for $v \in V$ $c[v][i] = \min \{ c[v][i-1],$

(假设连通)

→ step 4~5: $\sum_V (1 + \text{in-degree}(v)) = |V| + |E| = O(|E|)$

⇒ total $= O(|V||E|)$

若存在 k s.t. $c[v][k] = c[v][k-1]$, 循环可停止

time:

Lemma:

if G has no negative cycle if and only if $c[v][n-1] == c[v][n]$ for any $v \in V$

proof:

⇒): trivial

⇐): for any $k \geq n$ $c[v][k] = c[v][n-1]$ ⇒ G has no negative cycle. (在第n层稳定)

(*) if $c[v][n-1] == c[v][n]$ for any v // 不带负圈

return $c[v][n-1]$ for all v

else

report G has negative cycle.

All-pair shortest path (ASAP) 找每个点对间最短路.

Input: a directed graph $G = (V, E)$ with edge cost $c: E \to \mathbb{Z}$

Output: $\delta(u, v)$ for every $(u, v) \in V \times V$

if $\forall e, c(e) \geq 0$ $|V| \times$ Dijkstra's $- O(|V|^2 \log|V| + |V||E|)$

if $\exists e, c(e) < 0$ $|V| \times$ Bellman-Ford $- O(|V|^2|E|)$

↓ 优化

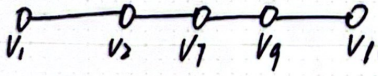
Floyal-warshall $O(|V|^3)$ (稠密时更好) DP

$V = v_1, \ldots, v_n$

P 

rank(P) = largest index of internal nodes of P

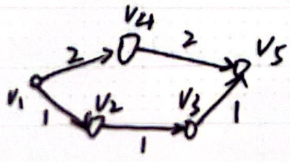
rank(P) = 0 if P has no internal nodes

e.g.  rank = 9

Subproblems.

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

let $c[i][j][k]$ = the cost of shortest path of rank at most k from v_i to v_j.

e.g. 

$c[1][5][0] = +\infty$ $c[1][5][3] = 3$

$c[1][5][1] = +\infty$ $c[1][5][4] = 3$

$c[1][5][2] = +\infty$

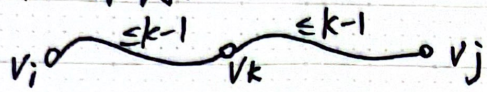
No.

Date

Recurrence

$c[i][j][k] \to p^*$ $\quad rank(P^*) \le k$ ← 根据编号是否取到来讨论

case 1: 取到 k



$\Rightarrow c[i][k][k-1] + c[k][j][k-1]$

case 2: 没有取到 k: $\quad c[i][j][k-1]$

$\Rightarrow c[i][j][k] = \min\{c[i][j][k-1],\ c[i][k][k-1] + c[k][j][k-1]\}$

按 k 从 1 到 n 遍历即可

base case $\begin{cases} c[i][i][0] = 0 \\ c[i][j][0] = +\infty \quad (i \neq j) \end{cases}$

n^3 subproblems

Remark: G has a negative cycle if and only if $c[i][i][n] < 0$ for some i.