From MIT: An introduction to algorithm

Sunday, January 26, 2020 7:55 PM

Asymptotic Notation

$$\boxed{\begin{array}{l} f(n) = O(g(n)) \\ \text{means. } \exists \text{ const } c>0.\ n_0>0. \\ \quad \text{s.t. } 0 \le f(n) \le c\,g(n)\ \forall n>n_0 \end{array}}$$

<u>macro convention</u>

asymmetric

<u>e.x.</u> $n^2 + O(n) = O(n^2)$.

means. $\forall f(n) \in O(n)$. $\exists g(n) \in O(n^2)$.
s.t. $n^2 + f(n) = g(n)$

Moreover, Ω notation.
$f(n) = \Omega(g(n))$

$\exists c>0,\ n_0>0$.
s.t. $0 \le g(n) \le f(n)$.

Θ notation.
$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$.

Strict notations: $O \to o$. $\Omega \to \omega$.
$\le \to <$. $\ge \to >$
$\exists c.n_0 \to \forall c.\ \exists n_0(c)$.

(也就是不包含同级幂.).

Solving Recurrences

1. Substitution.
   Gauss $\Rightarrow$ prove by induction.

Ex. $T(n) = 4T(n/2) + n$.

Gauss n^2.
Assume $T(k) \le ck^2$ for $k<n$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c\cdot(\tfrac{n}{2})^2 + n \\ &= cn^2 + n \quad \text{失败} \end{aligned}$$

$\Rightarrow$ Assume $T(k) \le c_1k^2 - c_2k$ for $k<n$.

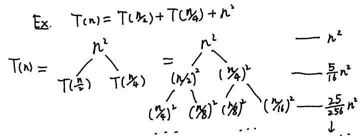
$$\begin{aligned} T(n) &= 4T(\tfrac{n}{2}) + n \\ &\le 4c_1(\tfrac{n}{2})^2 - 4c_2(\tfrac{n}{2}) + n \\ &= c_1n^2 - (2c_2-1)n \le c_1n^2 - c_2n. \end{aligned}$$

Let $c_2 \ge 1$. $2c_2 - 1 \ge c_2$.

$\Rightarrow T(n) \le c_1n^2 - c_2n^2 \Rightarrow T(n) = O(n^2)$

2. Recursion-tree method.

Ex. $T(n) = T(n/2) + T(n/4) + n^2$



Count level-by-level $\Rightarrow$ upper bound.

$T(n) \le 1 + \frac{5}{16} + \dots + (\frac{5}{16})^k + \dots$

$\Rightarrow T(n) = O(n^2)$

3. Master Theorem

$T(n) = aT(n/b) + f(n)$
where $a \ge 1$. $b<1$. $f(n)$: <u>asymptotically positive</u>
$(\exists n_0>0.\ \forall n>n_0\ f(n)>0)$.

Compare $f(n)$ with $n^{\log_b a}$

<u>Case 1</u> $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$.
$\Rightarrow T(n) = \Theta(n^{\log_b a})$

<u>Case 2</u> $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some $k \ge 0$
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$

<u>Case 3</u> $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$.
& $af(n/b) \le (1-\varepsilon)f(n)$ for some $\varepsilon > 0$
$\Rightarrow T(n) = \Theta(f(n))$

Divide and conquer

<u>Mergesort</u>

$T(n) = 2T(n/2) + \Theta(n)$.
$\Rightarrow T(n) = \Theta(n\log n)$ by master theorem

<u>Fibonacci Sequence</u>

①. recursive.
$T(n) = T(n-1) + T(n-2) + \Theta(1)$
$T(n) = \Omega(\varphi^n)$. $\varphi = \frac{1+\sqrt{5}}{2}$.

Quicksort

Hoare 1962

- Divide & conquer
- very practical

1. partition array into 2 subarrays around pivot x

$\boxed{\le x}\boxed{x}\boxed{\ge x}$

2. Conquer (Recursive)
3. Combine.

```
def Partition (A,p,q): // array A from p to q
    pivot = A[p]
```

① recursive.

$T(n) = T(n-1) + T(n-2) + \Theta(1)$

$T(n) = \Omega(\varphi^n)$. $\varphi = \frac{1+\sqrt{5}}{2}$

② bottom-up iteration.

Compute $F_0, F_1, \cdots, F_n$ $\Rightarrow T(n) = \Theta(n)$

③ $F_n = \frac{\varphi^n}{\sqrt{5}}$ nearest integer. $T(n) = \Theta(\lg n)$

④ Recursive Squaring

$$A^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

$\Rightarrow T(n) = T(n/2) + \Theta(1)$

$T(n) = \Theta(\lg n)$

Matrix Multiplication

Naive: A.B $n \times n$. AB $\Rightarrow \Theta(n^3)$

Divide & conquer (Block matrix)

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$

Again n^3.

$\Downarrow$

Strassen's algorithm

$P_1 = a(f-h)$ $\quad P_5 = (a+d)\cdot(e+h)$

$P_2 = (a+b)\cdot h$ $\quad P_6 = (b-d)\cdot(g+h)$

$P_3 = (c+d)\cdot e$ $\quad P_7 = (a-c)\cdot(e+f)$

$P_4 = d(g-e)$

$r = P_5 + P_4 - P_2 + P_6$

$s = P_1 + P_2$

$t = P_3 + P_4$

$u = P_5 + P_1 - P_3 - P_7$

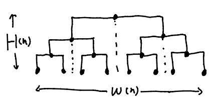
$$C = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

$\Rightarrow T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$

$T(n) = \Theta(n^{\log_2 7})$

VLSI Layer

Embed a complete binary tree.



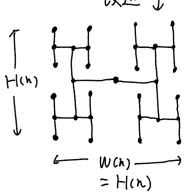
Area(n) = $H(n) \cdot W(n)$

$H(n) = H(\frac{n}{2}) + 1$ $\quad H(n) = \Theta(\lg n)$

$W(n) = 2W(\frac{n}{2}) + 1$ $\quad W(n) = \Theta(n)$

Area(n) = $\Theta(n \lg n)$

改进 $\downarrow$



$H(n) = 2H(\frac{n}{4}) + 1$

$\Rightarrow H(n) = \Theta(\sqrt{n})$

Area(n) = $\Theta(n)$

3. Combine

```
def Partition (A,p,q): // array A from p to q
    pivot = A[p]
    for j = p+1 to q
        if A[j] <= x
            i = i+1
            exchange A[i] with A[j]
    exchange A[p] with A[i]
    return
```

Analysis

worst case: input sorted or reverse sorted.

$T(n) = T(n-1) + \Theta(n)$

$= \Theta(n^2)$

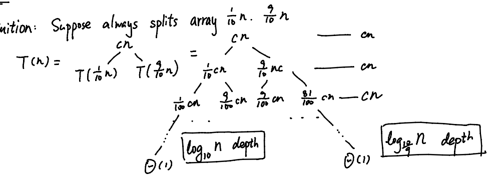
Best case: splits the array $\frac{n}{2}$.

$T(n) = 2T(n/2) + \Theta(n)$

$= \Theta(n \lg n)$

Average Case

Intuition: Suppose always splits array $\frac{1}{10}n$. $\frac{9}{10}n$



$\Rightarrow cn \log_{10} n \le T(n) \le cn \log_{\frac{10}{9}} n$

$\Rightarrow T(n) = \Theta(n \lg n)$

Randomized Quicksort

- running time independent from input ordering
  - $\rightarrow$ no assumption to input distribution
- Randomly choose pivot elem.

Analysis

for $k = 0, 1, 2, \cdots, n-1$.

let $X_k = \begin{cases} 1 & \text{if generates } k : n-k-1 \text{ partition} \\ 0 & \text{else.} \end{cases}$

(indicator variable)

$E(X_k) = \frac{1}{n}$

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \\ \cdots \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \end{cases}$$

$$T(n) = \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n) \right)$$

$$E(T(n)) = \sum_{k=0}^{n-1} E(X_k) \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

partitions, random choice made. independent.

$= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] \times 2 + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)$

$= \frac{2}{n} \sum_{k=0}^{n-1} E[T(k)] + \Theta(n)$

prove $E[T(n)] \le an \lg n$

$\boxed{\text{Use } \sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2}$

$E[T(n)] \le \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$

$\le \frac{2a}{n} (\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2) + \Theta(n)$

$= an \lg n - \frac{a}{4} n + \Theta(n)$

$\le an \lg n$

Linear time sorting

Comparison sorting model

- only use comparisons to determine order
- No comparison sorting algorithm runs faster than nlgn

$<a_1, a_2, \dots, a_n>$ for decision tree.

each internal nodes compare two elems

⇒ split the tree into two subtrees whenever making a comparison

⇒ running time = length of a node-leaf path
worst case: the height of the tree

Prove that the height of the decision tree is $\Omega(n\lg n)$

Proof. # leaves must be $n!$
height h ⇒ # leaves $\le 2^h$

⇒ $n! \le 2^h$ $\qquad \frac{n!e^n}{n^{n+\frac{1}{2}}} \sim \sqrt{2\pi}$

$h \ge \lg n!$

$\ge n\lg n$

⇒ $h = \Omega(n\lg n)$

Therefore, all comparison sorting algos are $\Omega(n\lg n)$

Counting sort model

Input $A[1,\dots,n]$
each $A[i] \in \{1,2,\dots,k\}$

Count how many times $x \in \{1,2\dots,k\}$ appears among $A[1,\dots,n]$. auxiliary space $O(k)$
Then rearrange the input array using the counter. Time $O(k+n)$

Stable sort preserve the relative order of elements.

Radix Sort

Hollerith 1890

- digit by digit sorting
- start from the least significant digit ⇐ must be stable sort
- when sorting i^{th} digit, the $(i+1)^{th} \sim k^{th}$ digits (suffix) are sorted

ex.

8 6 3		4 3 1		4 3 1			
3 6 7	⇒	8 6 3	⇒	5 3 9			
5 3 9		3 6 7		8 6 3			
4 3 1		5 3 9		3 6 7			

Now the last two digits perfectly in order!

Suppose: n integers in b bits
splits into b/r digits, each r bits long

Time: $O(b/r \cdot (n+2^r))$

Choose $r = \lg n$. ⇒ $O(\frac{b\cdot n}{\lg n})$

Or ⇒ handle $0,1,\dots,n^d-1$ in $O(dn)$ time.

Order statistics

- given n elems, find k^{th} smallest (elem of rank k)

naive: Sort A and return $A[k]$ ⇒ $O(n\lg n)$

Randomized order algo

Expected linear

Randomselect(A, p, q, i):
  if $p=q$ then return $A[p]$
  r = randomQuicksort(A, p, q) // pivot position
  $k = r-p+1$ // rank$(A[r])$.
  if $i=k$ then return $A[r]$
else if $i<k$ then return Randomselect$(A, p, r-1, i)$
  else then return Randomselect$(A, r+1, q, i-k)$

A: p [$\le A[r]$ | r | $\ge A[r]$] q

Analysis

Assume: elems are distinct.

$$T(n) \le \begin{cases} T(\max\{0,n-1\}) + \Theta(n) & \text{if } 0 : n-1 \\ \dots & \dots \\ T(\max\{n-1,0\}) + \Theta(n) & \text{if } n-1 : 0 \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left[T(\max\{k, n-k-1\}) + \Theta(n)\right]$$

$$\Rightarrow E(T(n)) = \frac{1}{n}\sum_{k=0}^{n-1} E[T(\max\{k,n-k-1\})] + \frac{1}{n}\sum_{k=0}^{n-1}\Theta(n)$$

$$\le \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} E(T(k)) + \Theta(n)$$

「奇数多加一个.

Prove: $T(n)$ is linear (expected)

$E(T(k)) \le ck$ for $k<n$

Then: $E[T(n)] \le \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} E(T(k)) + \Theta(n)$

$\le \frac{2c}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} k + \Theta(n)$

$\le \frac{2c}{n}\cdot\frac{3}{8}n^2 + \Theta(n)$

$= cn - (\frac{1}{4}cn - \Theta(n))$

$\le cn$. for c sufficiently large.

⇒ $E(T(n)) = \Theta(n)$.

Worst-Case Linear-time order statistic

(Blum, Floyd, Pratt, Rivest, Tarjan 1973)

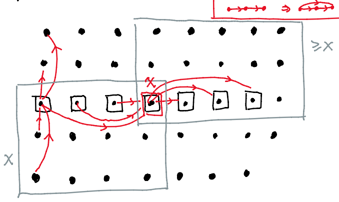
idea:
- guaranteed good pivot

Steps

1. Devide the array into $\lfloor n/5 \rfloor$ groups containing (at most) 5 elems.
   And Find the medium of each group $\Theta(n)$

notation: $a < b$

2. Recursively Select the medium of $\lfloor n/5 \rfloor$ medium. ⇒ $T(\lfloor n/5 \rfloor)$



Same as Randomized one.

3. Partition with x as pivot
   Let k = rank(x)
4. if $i=k$ then return x
   if $i<k$ then recursively select the i^{th} smallest in the lower part of A
   else recursively select the $(i-k)^{th}$ smallest in the upper part of A

Analysis $3\lfloor\lfloor n/5 \rfloor/2\rfloor$ elems $\le x$ / $\ge x$
(at least)

$= 3\lfloor n/10 \rfloor$

Simplication for $n \ge 52$, $3\lfloor n/10 \rfloor \ge \frac{n}{4}$

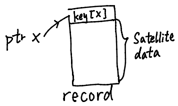
⇒ $T(n) \le T(n/5) + T(\frac{3}{4}n) + \Theta(n)$

⇒ $T(n) = \Theta(n)$

Hashing

Symbol-table problem

Table S holding n records



Operations:

- Insert (S, x): $S \leftarrow S \cup \{x\}$ } dynamic sets
- Delete (S, x): $S \leftarrow S - \{x\}$ }
- Search (S, k): returns x if key[x] = k; null if no such x.

Direct access table

Suppose: keys drawn from $U = \{0, 1, \ldots, m-1\}$

Assume keys are distinct.

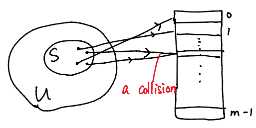
Set: $T[k] = \begin{cases} x & \text{if } x \in S \text{ and key}[x] = k \\ \text{null} & \text{otherwise.} \end{cases}$

Operations take $\Theta(1)$ time.

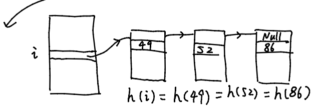
Suffer great limitation.

Hashing

Hash function h maps keys "randomly" into slots of table T



☆ Resolve collision by chaining — create lists



$h(i) = h(49) = h(52) = h(86)$

Analysis

Worst Case: every key hashes to same slot. Access takes $\Theta(n)$ time.

Average Case

Assumption of simple uniform hashing.

- each key $k \in S$ is equally likely to be hashed to any slots in T.
- independent from where other keys are sloted

Def. Load factor

n keys, m slots.

$\alpha = \frac{n}{m}$ = average length of each list

$\Rightarrow$ Expected unsuccessful Search time:

$= \Theta(1+\alpha)$

Expected search time $= \Theta(1)$ if $\alpha = O(1)$

$\Leftrightarrow$ if $n = O(m)$

Choosing a hash function

- should distribute keys uniformly into slots.
- Regularity in key distributions should not affect uniformly

Division method

$h(k) = k \mod m$

- Don't pick m with small divisor d.

  Ex. $d = 2 \Rightarrow$ if keys are all even, all odd slots never used.

Thus, choose m = prime.

Multiplication method

faster than division method.

$m = 2^r$, computer has w-bit words.

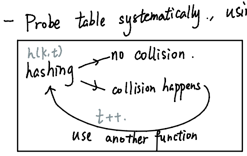
$h(k) = (A \cdot k \mod 2^w)$ right shift $(w-r)$

odd integer $2^{w-1} < A < 2^w$.

- A not too close to 2^{w-1} or 2^w

Resolving collisions by open addressing

- No storage for links.
- Probe table systematically, using a sequence of hashing functions.



$h(k) \Rightarrow h(k, t)$

key — times of failed hashing

- follow same sequence while searching
- cannot delete a record safely
- n records, m slots. $n \leq m$

Probing Strategies (for open hashing)

1. Linear: $h(k, i) = (h(k, 0) + i) \mod m$

   Problem: clustering phenomena.

2. Double hashing: $h(k, i) = (h_1(k) + i \cdot h_2(k)) \mod m$

   usually pick $m = 2^r$ and h_2 to be odd.

Analysis of open hashing

- Assumption: each key equally likely to have any one of the m! perms as its probe sequence indep. of others.

- Theorem. $\mathbb{E}(\text{times of probes}) \leq \frac{1}{1-\alpha}$ if $\alpha < 1$.

Proof: 1 probe always necessary.

With probability n/m, collision $\Rightarrow$ another probe.

Now with prob. $\frac{n-1}{m-1}$ collision.

...

Note: $\frac{n-i}{m-i} < \frac{n}{m} = \alpha$. $(i > 0)$.

$\mathbb{E}(\#\text{ probe}) = 1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(1 + \frac{n-2}{m-2}(1 + \cdots)\right)\cdots\right)$

$\leq 1 + \alpha(1 + \alpha(1 + \cdots)\cdots)$

$\leq 1 + \alpha + \alpha^2 + \cdots$

$= \frac{1}{1-\alpha}$.

$\Rightarrow$ if $\alpha < 1$ is constant. $\Rightarrow O(1)$ probes.

- Take $\alpha = \frac{1}{2}$

Universal hashing

Weakness of hashing

For any choice of hash function, exists a bad set of keys.

$\Rightarrow$ Choose a hash function at random $\Rightarrow$ indep. from input

Universal hashing

Def. $\mathcal{U}$: universe of keys.

$\mathcal{H}$: finite collection of hash functions mapping $\mathcal{U}$ to $\{0, 1, \dots, m-1\}$

$\mathcal{H}$ is universal, if $\forall x, y \in \mathcal{U}$. $(x \neq y)$, $\left|\{h \in \mathcal{H}, h(x) = h(y)\}\right| = \frac{|\mathcal{H}|}{m}$

Theorem

Choose h randomly from $\mathcal{H}$, hashing n keys into m slots.

For a given key x. $E(\#\text{ collisions with } x) < n/m$

Proof.

C_x : random variable denoting total # collisions of keys in T with x.

Let $C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise.} \end{cases}$

$E(C_{xy}) = \frac{1}{m}$. $C_x = \sum_{y \in T - \{x\}} C_{xy}$

$\Rightarrow E[C_x] = E\left[\sum_{y \in T-\{x\}} C_{xy}\right] = \sum_{y \in T - \{x\}} \frac{1}{m}$

$= \frac{n-1}{m} < \frac{n}{m}$.

Construction of a universal hash function

Let m be prime: Decompose key k into $r+1$ digits

$k = \langle k_0, k_1, \dots, k_r \rangle_m$ where $0 \le k_i \le m-1$ (base m).

Pick $a = \langle a_0, a_1, \cdots a_r \rangle_m$ each a_i chosen randomly from $0 \sim m-1$.

Hash function: $h_a(k) = \left(\sum_{i=0}^{r} a_i k_i\right) \bmod m$

$|\mathcal{H}| = m^{r+1}$

Prove the $\mathcal{H}$ is universal:

Let $x = \langle x_0, \cdots, x_r \rangle$. $y = \langle y_0, \cdots, y_r \rangle$. $x \neq y$.

W.L.o.g. (without loss of generality). differ at position 0. $(x_0 \neq y_0)$

If collide, $h_a(x) = h_a(y)$.

$\Rightarrow \sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \pmod m$

$\Leftrightarrow \sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \pmod m$

$\Leftrightarrow a_0 (x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i (x_i - y_i) \pmod m$

> Number theory fact.
> m prime. $\forall z \in \mathbb{Z}_m$. and $z \neq 0$.
> $\exists$ unique $z^{-1} \in \mathbb{Z}_m$ s.t. $z z^{-1} \equiv 1 \pmod m$.

$\Leftrightarrow a_0 \equiv \left(-\sum_{i=1}^{r} a_i (x_i - y_i)\right) (x_0 - y_0)^{-1} \pmod m$

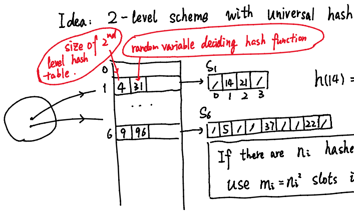
(exists $(x_0 - y_0)^{-1}$)

a_0 depends on $a_1 \sim a_r$ $\Rightarrow m^r = \frac{|\mathcal{H}|}{m}$ choices to cause collision

Perfect hashing

– Given n keys, construct a static hash table of size $m = O(n)$
s.t. search takes $O(1)$ time in the <u>worst case</u>

Idea: 2-level scheme with universal hash at both level, with no collision at 2^{nd} level



$h(14) = h(21) = 1$, $h_{31}(14) = 1$. $h_{31}(21) = 2$

If there are n_i hashed to level-1 slot i,
use $m_i = n_i^2$ slots in level-2 table S_i

Analysis

Thm. Hash n keys into $m = n^2$ slots using random h in universal $\mathcal{H}$

$E[\#\text{ collisions}] < \frac{1}{2}$

Pf. Prob. 2 given keys collide: $\frac{1}{m} = \frac{1}{n^2}$

$\binom{n}{2}$ pairs of keys

$\Rightarrow E[\#\text{ collisions}] = \binom{n}{2}\frac{1}{n^2} = \frac{n-1}{2n} < \frac{1}{2}$

Markov ineq.

For random variable $X \geq 0$, $Pr\{X \geq t\} \leq \frac{E[X]}{t}$

Proof. $E(x) = \int_0^t u\, P_r(X=u)du + \int_t^{\infty} u\, P_r(X=u)du$

$\geq 0 + \int_t^{\infty} t\, P(X=u)du$

$= P_r(X \geq t)\, t$.

Corollary $Pr\{\text{no collision}\} \geq \frac{1}{2}$

$Pr\{\geq 1 \text{ collision}\} \leq \frac{E(\#\text{ collisions})}{1} < \frac{1}{2}$

$\Rightarrow P_r\{\text{no collision}\} \geq \frac{1}{2}$

To find level-2 hash function, just try a few to find one, which is proved to be easy.

For level 1. choose $m = n$.
For level 2. choose $m_i = n_i^2$.

$\Rightarrow E[\text{total storage}] = n + E[\sum_{i=0}^{n-1} \Theta(n_i^2)]$

$= \Theta(n)$.

Binary Search Trees (BST)

BST sort

– Build BST
– do an inorder traversal (中序) } Algorithm.

– Same as Quick Sort in disguise (make same comparisons, but in different order)

$\boxed{\text{Randomized BST Sort.}} = \boxed{\text{Randomized Quick Sort}}$

<u>Theorem</u>

$E[\text{height of random built BST}] = O(\log n)$

<u>Jensen's Inequality</u>

f: convex function

$f[E(X)] \leq E[f(X)]$

Proof. convex: $X_1, X_2 \in \mathbb{R}$.

$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2)$

$\Rightarrow f(\sum_{i=1}^{k} \alpha_i x_i) \leq \sum_{i=1}^{k} \alpha_i f(x_i) \quad (\alpha_i \geq 0, \sum_{i=1}^{k}\alpha_i = 1)$

proved by induction.

$f(\sum_{i=1}^{n} \alpha_i x_i) = f[\alpha_n x_n + (1-\alpha_n)\sum_{i=1}^{n-1} \frac{\alpha_i}{1-\alpha_n} x_k]$

$\leq \alpha_n f(x_n) + (1-\alpha_n) f[\sum_{i=1}^{n-1} \frac{\alpha_i}{1-\alpha_n} x_k]$

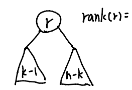
$\Rightarrow f(E(X)) = f(\int_{-\infty}^{\infty} u g(u) du) \leq \int_{-\infty}^{\infty} f(u) g(u) du$

$\int_{-\infty}^{\infty} g(u) = 1 \qquad = E(f(x))$

Expected height

X_n = r.v. of height of randomly built BST

$Y_n = 2^{X_n}$



if $\text{rank}(r) = k$,

$X_n = 1 + \max\{X_{n-k}, X_{k-1}\}$

$Y_n = 2\max\{Y_{n-k}, Y_{k-1}\}$

<u>deal with "plus 1":</u>
calculate 2^{X_n} instead of X_n

Let $Z_{nk} = \begin{cases} 1 & \text{rank}(r) = k \\ 0 & \text{otherwise.} \end{cases}$

$E(Z_{nk}) = \frac{1}{n}$

$\Rightarrow Y_n = \sum_{k=1}^{n} Z_{nk}(2\cdot\max\{Y_{n-k}, Y_{k-1}\})$

$E(Y_n) = E[\sum_{k=1}^{n} Z_{nk}(2\max\{Y_{n-k}, Y_{k-1}\})$

$= \frac{2}{n}\sum_{k=1}^{n} E[\max\{Y_{n-k}, Y_{k-1}\}]$

$\leq \frac{2}{n}\sum_{k=1}^{n} E[Y_{n-k} + Y_{k-1}]$

$= \frac{4}{n}\sum_{k=0}^{n-1} E[Y_k]$

By substitution method

Claim: $E[Y_n] \leq cn^3$

If $E[Y_k] \leq ck^3$ whenever $k < n$

$E(Y_n) \leq \frac{4}{n}\sum_{k=0}^{n-1} E[Y_k] \leq \frac{4c}{n}\sum_{k=0}^{n-1} k^3 = \frac{4c}{n}\cdot\frac{n^2(n-1)^2}{4} \leq cn^3$

$\Rightarrow E(Y_n) = O(n^3)$

$2^{E(X_n)} \leq E[2^{X_n}] = E(Y_n) = O(n^3)$

By Jenson's Inequality

$\Rightarrow E(X_n) \leq 3\lg n + O(1)$

Balanced Search Tree

– Search tree data structure maintaining dynamic set of n elements using tree of height $\Theta(\log n)$

Examples:

– AVL trees (1962)
– 2-3 trees
– 2-3-4 trees
– B-trees
– Red-black trees
– Skip List

Red-Black trees

<u>Red-Black Properties</u>

1. Every node is either red or black
2. The root & leaves (nil's) are black
3. Every red node has black parent
4. All simple paths from a node x

Augmenting Data Structure

Methodology:
1. Choose a DS.
2. Determine additional info.
3. Verify info. maintained while operating
4. Use info to develop new operations.

Dynamic order Statistics

Select (i) : return i^{th} smallest elem.
Rank (x) : return the rank of x.

(In Balanced Tree)

Select i^{th} smallest in tree rooted at x.

```
k = (x→left)→size + 1
if (k==i) return x
else if (i<k) return Select(x→left, i)
     else     return Select(x→right, i-k)
```

memorize size of each subtree. and update when Inserting/Deleting

Analysis:
$O(\lg n)$

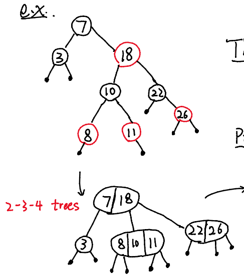
Rank Also $O(\lg n)$

Update size of each subtree

— 2-3-4 trees
— B-trees
— Red-black trees
— Skip List
— Treaps

2. The root & leaves (nil's) are black
3. Every red node has black parent
4. All simple paths from a node x to its descendant leaf have same # black nodes = black-height(x)
does not contain x itself

e.x.

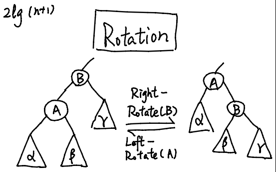


Theorem
Red-black tree with n keys has height: $h \le 2\lg(n+1)$.

Proof Sketch
Merge every red nodes with their parents
→ By Property 4, every leaves have same depth.
$\Rightarrow 2^{h'} \le$ # leaves $\le 4^{h'}$
leaves = n+1. (In either tree).
$\Rightarrow h' \le \lg(n+1)$
$h \le 2h' \le 2\lg(n+1)$

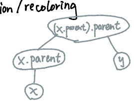
How to preserve Red-black property?
- BST operations (Delete, insert, etc.)
- color changes
- reconstructuring of links via rotations

Rotation

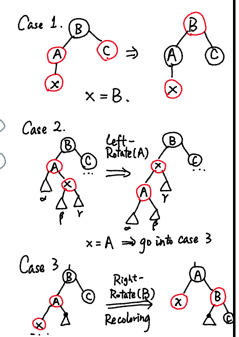


preserve BST properties:
left ≤ root ≤ right

Insert (x):
1. Tree-Insert (x)
2. Choose color red
3. If its parent is red ⇒ break property 3.
Then move violation up via recoloring until we can fix violation via rotation/recoloring



while (x ≠ root and x.color = Red)
do if x.parent == (x.parent).parent).left
if y.color == Red <Case 1>
else if x == (x.parent).right <Case 2>
else <case 3>
else Symmetric (left ↔ right)
root.color = Black.



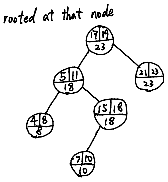
else if (i<k) return Select(x→left, i)
else return Select(x→right, i−k)

Update size of each subtree
E.x. Insert (x)
- plus one at each node on the path.
Now the tree may no longer hold Red-Black Properties
- Update Size during Rotation
⇒ Insert, Delete (and update size) still $O(\log n)$ time.

Interval trees

Maintain a set of intervals.
a b
low high endpoint.
Operations:
- Query: Find an interval in the set that overlaps with x.

1. Red-Black tree, use low endpoint as keys
2. Store in node the largest value of the subtree rooted at that node
3. Modify operations
Insert:
- Update max along the path
- fix rotation Takes O(1) time
Totally takes $O(\log n)$



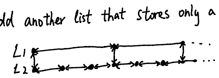
4. Develop new operation:
Interval Search (i): // Find an interval that overlaps with i.
x = root
while (x ≠ nil and (i.low > x.high or x.low > i.high))
do
if (x.left ≠ nil and i.low ≤ (x.left).max)
then x = x.left
else x = x.right
return x

Analysis
Time = $O(\lg n)$
List all overlaps: $O(k \lg n)$ (k: # overlaps)

Correctness
$L = \{i' \in x.left\}$ $R = \{i'' \in x.right\}$ $K = \{i' \text{ overlaps with } i\}$
If search goes right. $L \cap K = \phi$
If search goes left. If $L \cap K = \phi$, then $R \cap K = \phi$

Skip Lists

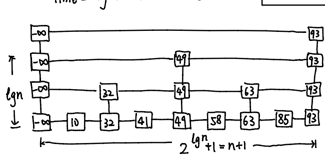
— dynamic search structure
Starting from a sorted list ⇒ $\Theta(n)$ time search
⇓
Add another list that stores only a few elems



Worst Case: Time $\approx |L_1| + \frac{|L_2|}{|L_1|} \ge 2\sqrt{n}$.
⇓
Add more lists:

# list	Time
1	n
2	$2\sqrt{n}$
3	$3\sqrt[3]{n}$
...	...
k	$k\sqrt[k]{k}$

⇒ Let $k = \lg n$,
Time = $\lg n \sqrt[\lg n]{n} = 2\lg n$



An Ideal skip list

How to Maintain?
Insert (x)
- Search to find the place to input.
- Insert x into the bottom list.

Amortized Analysis

Amortized: Average performance in the worst case

How large should a hash table be?
- a tradeoff between time & space.
- what if we don't know n in advance?

Solution: Dynamic tables.
If table is "full", Resize and move elems into the new table.
resize: $2^k \to 2^{k+1} \to \ldots$

Analysis (Amortized)
Worst Case of 1 Insert: $\Theta(n)$ (copying)
Worst Case of n Insert: $\Theta(n)$ NOT $\Theta(n^2)$.
c_i: time cost of i^{th} insert.
$$c_i = \begin{cases} i-1 & \text{if } i = 2^k+1 \\ 1 & \text{otherwise.} \end{cases}$$
⇒ Cost of n Inserts: $\sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j \le n + 2^{\lfloor \lg(n-1) \rfloor + 1} \le 3n$
⇒ Average Cost = $\frac{\Theta(n)}{n} = \Theta(1)$

Types of Amortized Analysis:
- Aggregate. - Accounting - Potential
(The above)
Accounting: charge i^{th} operation with a fictitious amortized cost $\hat{c}_i$.
sum of the amortized cost − sum of all operations up to that point ≥ 0

Insert (x)

- Search to find the place to input.
- Insert x into the bottom list.
- flip a coin (50% prob.) : make x go up a level
  ↳ (do this recursively)

Delete (x)

- Just delete x at anywhere.

Theorem

- Every search in n-element skip list costs $O(\lg n)$ with high prob. : $\forall \alpha \geq 1$, Prob. $\geq 1 - O(1/n^{\alpha})$

Proof.

$Pr\{\#\text{ lists} > c\lg n\} \leq n \cdot \frac{1}{2^{c\lg n}} = \frac{1}{n^{c-1}}$

- Now start from the target node at the bottom list. backtrack.
- each time goes left or up depending on coin flip (50% prob.)

$\#$ up moves $< \#$ levels
$\leq c\lg n$ with high prob. $(\geq 1 - \frac{1}{n^{c-1}})$

$\#$ moves $\leq \#$ coins fliped till reaching $c\lg n$. w.h.p. $(\geq 1 - \frac{1}{n^{c-1}})$

If flip $kc\lg n$ coins,

$Pr\{\leq c\lg n \text{ Heads}\} \leq \binom{kc\lg n}{c\lg n}\left(\frac{1}{2}\right)^{(k-1)c\lg n}$

$\boxed{\binom{y}{x} \leq \left(e\frac{y}{x}\right)^{x}}$

$\leq 2^{\lg(ke)\cdot c\lg n - (k-1)c\lg n}$

$= \frac{1}{n^{\alpha}}$ $\quad (\alpha = c(k-1) - \lg(ke))$

$\Rightarrow$ Time $= O(\lg n)$ w.h.p. (of $1 - \frac{1}{n^{\alpha}}$)

Accounting: charge i^{th} operation with a fictitious amortized cost $\hat{c}_i$.

Sum of the amortized cost − sum of all operations up to that point ≥ 0

$\Rightarrow \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \ (\forall n)$

Dynamic table

- $\hat{c}_i = 3$ for i^{th} insert.
  1 for immediate insert, 2 stored for table doubling.
- When table doubles,
  1 for moving old elems.
  1 for moving new elems

0	0	2	2

$2+2=4$ to move 4 elems, (2 old, 2 new.)

0	0	0	0				

$\Rightarrow \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = 3n$

Potential method

Framework:

- start with data structure D_0
- Operation i transform D_{i-1} to D_i, cost c_i
- Define Potential Function $\Phi: \{D_i\} \to \mathbb{R}$.
  s.t. $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0 \ \forall i$.
- Define amortized cost $\hat{c}_i$ w.r.t. Φ is :

$$\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{=\Delta\Phi.\ \text{potential difference.}}$$

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

$\Rightarrow$ e.x. Table doubling

Def. $\Phi(D_i) = 2i - 2^{\lceil \lg i \rceil}$

Assume $2^{\lceil \lg 0 \rceil} = 0$.

$\Phi(D_i) \geq 2i - 2^{\lg i + 1} = 2i - 2i = 0$.

$\hat{c}_i = c_i + \Delta\Phi_i$

$= \begin{cases} i & \text{if } i = 2^k+1 \\ 1 & \text{otherwise} \end{cases} + (2 + 2^{\lceil \lg(i-1) \rceil} - 2^{\lceil \lg i \rceil})$

① $i = 2^k+1$, $\hat{c}_i = i + 2 + i - 1 - 2(i-1) = 3$

② $i \neq 2^k+1$, $\hat{c}_i = 1 + 2 + 2^{\lceil \lg(i-1) \rceil} - 2^{\lceil \lg i \rceil} = 3$

$\Rightarrow \sum_{i=1}^{n} \hat{c}_i = 3n$.

Competitive analysis

Def. A sequence S of operations is provided one at a time.
For each op. an online algo. A must execute it immediately.
An Offline algo B may see the sequence in advance.

Goal: min cost of A. $\Rightarrow C_A(S)$.

☆ An online alg. A is α-competitive if $\exists$ const k,
st. $\forall$ seq. S of ops,
$C_A(S) \leq \alpha \cdot C_{OPT}(S) + k$

Self-organizing list

- search (x): cost time k. if $\text{rank}(x) = k$
- transpose 2 adjacent elems.

Worst Case of Online

Always choose the last elem : $C_A(S) = \Omega(|S| \cdot n)$

Average Case analysis

Suppose elem x is accessed with prob. $p(x)$

$E[C_A(S)] = \sum_{x \in L} p(x) \cdot \text{rank}_L(x)$

minimized when L is sorted in decreasing order w.r.t p.

Heuristic: Keep count of # times each elem is accessed

Practice:

"Move-to-front" heuristic. ⇐ responds well to the hotness.
After accessing x, move x to head.

Theorem. "MTF" is 4-competitive for self-organizing lists

Proof. Let L_i be MTF's list after i^{th} access
L_i^* be OPT's

c_i = MTF's cost for i^{th} op.
$= 2\,\text{rank}_{L_{i-1}}(x)$ // search + transpose.

Dynamic programming

DP hallmark #1

Optimal substructure: An opt. solution to a problem contains opt solutions to subproblems

DP hallmark #2

Overlapping subproblems: A recursive solution contains a "small" number of distinct subproblems repeated many times.

Longest common subsequence (LCS)

- Given two seq. $x[1 \cdots m]$ and $y[1 \cdots n]$, find a longest seq. common to both

e.x.
x: A B C B D A B
y: B D C A B A

BDAB, BCAB, BCBA $= LCS(x, y)$

Strategy: Consider prefixes of x and y

Define $c[i,j] = |LCS(x[1 \cdots i], y[1 \cdots j])|$

Now ☆ $c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max\{c[i, j-1], c[i-1, j]\} & \text{otherwise} \end{cases}$

LCS: subproblem space contains $m \cdot n$ distinct problem.

Recursive tree:

(m, n)
(m-1, n) (m, n-1)
(m-2, n) (m-1, n-1) (m-1, n-1) (m, n-2)
... ...

$m+n$

$\Rightarrow O(2^{m+n})$

Memorization Alg.

```
LCS(x, y, i, j)
  if (c[i,j] == nil)
    then if (x[i] == y[j])
      then c[i,j] = LCS(x, y, i-1, j-1) + 1
```

L_i^* be OPT's

C_i = MTF's cost for i^{th} op.

$= 2\,\text{rank}_{L_{i-1}}(x)$ // search + transpose.

$C_i^* = \text{rank}_{L_{i-1}^*}(x) + t_i$ ← if OPT performs t_i transposes.

Define potential function $\Phi: \{L_i\} \to \mathbb{R}$ by

$\Phi(L_i) = 2 \cdot \left|\{(x,y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}\right|$

x precedes y in L_i list. y precedes x in L_i^* list

$= 2 \cdot \#$ inversions.

Note that: (i) $\Phi(L_i) \geq 0, \forall i$ (ii) $\Phi(L_0) = 0$.

Transpose creates or destroys an inversion.

$\Rightarrow \Delta\Phi = \pm 2$.

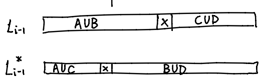
When op i access x, Def.

$A = \{y \in L_{i-1} : y \prec_{L_{i-1}} x \text{ and } y \prec_{L_{i-1}^*} x\}$

$B = \{y \in L_{i-1} : y \prec_{L_{i-1}} x \text{ and } y \succ_{L_{i-1}^*} x\}$

$C = \{y \in L_{i-1} : y \succ_{L_{i-1}} x \text{ and } y \prec_{L_{i-1}^*} x\}$

$D = \{y \in L_{i-1} : y \succ_{L_{i-1}} x \text{ and } y \succ_{L_{i-1}^*} x\}$



$\Rightarrow \text{rank}_{L_{i-1}}(x) = |A| + |B| + 1 = r$

$\text{rank}_{L_{i-1}^*}(x) = |A| + |C| + 1 = r^*$

$\Rightarrow x$ move-to-front: create $|A|$ inversions, destroy $|B|$ inversions.

Thus, $\Phi(L_i) - \Phi(L_{i-1}) \leq 2(|A| - |B| + t_i)$

Amortized cost:

$$\begin{aligned}\hat{C}_i &= C_i + \Phi(L_i) - \Phi(L_{i-1}) \\ &\leq 2r + 2(|A| - |B| + t_i) \\ &= 2r + 2(|A| - (r - |A| - 1) + t_i) \\ &= 4|A| + 2 + 2t_i \\ &\leq 4(r^* + t_i) \quad \text{since } r^* = |A| + |C| + 1 \geq |A| + 1 \\ &= 4C_i^*\end{aligned}$$

Thus, $C_{MTF}(S) = \sum_{i=1}^{n} C_i = \sum_{i=1}^{n} (\hat{C}_i + \Phi(L_{i-1}) - \Phi(L_i)) \leq 4\sum_{i=1}^{n} C_i^* + \Phi(L_n)$

If we count transpose that move x to the front of L as free, cost time $O(1)$.

then: MTF is 2-competitive

```
LCS(x, y, i, j)
    if (c[i,j] == nil)
        then if (x[i] == y[j])
                then c[i,j] = LCS(x, y, i-1, j-1) + 1
                else c[i,j] = max( LCS(x, y, i-1, j), LCS(x, y, i, j-1))

    return c[i,j]
```

$\Rightarrow$ Time $= \Theta(mn)$

Space $= \Theta(mn)$

DP algorithm

- compute the table bottom-up.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

$\Rightarrow$ BCBA

Time $= \Theta(mn)$

Reconstruct LCS by backtracking

Space $= \Theta(\min\{m, n\})$

(Always store only a row/column.

Graphs (review)

- Digraph (Directed) $G = (V, E)$
    - Set V of vertices
    - Set $E \subseteq V \times V$ of edges
- Undirected graph

$|E| = O(|V^2|)$

If G is connected $\Rightarrow |E| > |V| - 1$

$\Rightarrow \lg|E| = \Theta(\lg|V|)$

Graph representation

- Adjacency matrix of $G = (V, E)$.

$V = \{1, 2, \cdots, n\}$. $A_{n \times n}$.

$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$

$\Rightarrow \Theta(|V|^2)$ storage.

good for dense representation

e.x. for sparse graph: a chain, a tree ...

- Adjacency list of $v \in V$

Adj[v]: a list of vertices adjacent to v

In undirected graph: $|\text{Adj}[v]| = \text{degree}(v)$

In digraph, $|\text{Adj}[v]| = \text{outdegree}(v)$

Handshaking Lemma

$\sum_{v \in V} \text{degree}(v) = 2|E|$

$\Downarrow$

$\Theta(|V| + |E|)$ storage

Minimum Spanning Trees (Greedy Algorithm)

Input: Connected, undirected graph $G = (V, E)$ with weight function: $w: E \to \mathbb{R}$

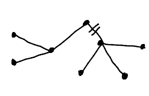
- Assumption: all edge weight distinct (w injective)

just for simplicity

Output: A spanning tree T (connects all vertices) of minimum weight.

$$w(T) = \sum_{(u,v) \in T} w((u,v))$$

Optimal Substructure → DP?



MST T:

- Remove $(u,v) \in T$ from T. $\Rightarrow$ split into T_1, T_2.
- T_1, T_2 are MST of G_1, G_2 respectively.

Hallmark for Greedy algorithms

Stronger property that leads to Greedy algorithm

- Greedy-choice property: A locally optimal choice is globally optimal

Let T be MST of $G = (V, E)$, $A \subseteq V$. Suppose $(u,v) \in E$ is least weight

Shortest path I

- Consider digraph $G = (V, E)$, with edge weight w
- path: $p = v_1 \to v_2 \to \cdots \to v_k$

$w(p) = \sum_{i=1}^{k-1} w(v_i \to v_{i-1})$

If there are negative edge weights

$\Rightarrow$ a negative cycle may make shortest path not exist. ($\delta(u,v) = -\infty$)

- Shortest-path weight (from u to v):

$\delta(u,v) = \min\{w(p) : p \text{ from } u \text{ to } v\}$

If u, v are not connected. ($\delta(u,v) = \infty$).

Optimal substructure

A substructure of a shortest path is a shortest path.

Triangle Inequality

$\delta(u,v) \leq \delta(u,x) + \delta(x,v)$

Single-Source shortest path

- from a given point s, find $\delta(s,u)$. $\forall u \in V$
- Assume $w(u,v) \geq 0$.

$\Rightarrow$ Dijkstra's Algorithm

① maintain set S of vertices whose $\delta(s,u)$ known.

② At each step, add one more vertex to S, whose distance is minimum.

③ Update distance of u in $V - S$ by $\min_{x \in S}\{\delta(s,x) + w(x,u)\}$

use priority queue

(update the adjacent vertices of the newly added vertex).

Analysis

- $|V|$ extract-min, $|E|$ decrease-key.

Array $\Rightarrow O(|V|^2)$.

Binary heap $\Rightarrow O((V+E)\lg V)$

Fibonacci heap $\Rightarrow O(E + V\lg V)$

- For unweighted graphs, $w(u,v) = 1$ $\Rightarrow$ BFS $\Rightarrow$ Time $= O(V+E)$
- use queue instead of priority queue

Shortest path II

- allow negative edge weight.
- some $\delta(u,v)$ may be $-\infty$ if negative weight cycle exists

Bellman-Ford Algorithm

$d[s] = 0$

Hallmark for Greedy algorithms

Stronger property that leads to Greedy alg…

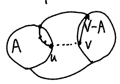
- Greedy-choice property: A locally optimal choice is globally optimal

Let T be MST of $G=(V,E)$, $A \subseteq V$. Suppose $(u,v) \in E$ is least weight edge connecting A to $V-A$. Then $(u,v) \in T$

Proof. suppose $(u,v) \notin MST$.

There is a unique simple path from u to v. (property of tree)

swap (u,v) with the first edge in this path that connect A and $V-A$



$\Rightarrow$ create a tree T'. $w(T') < w(T)$

$\Rightarrow$ contradiction.

Prim's algorithm

Idea: Maintain $V-A$ as a priority queue Q
key each vertex in Q with least weight connecting it to A

```
Q = V
key[v] = ∞   ∀ v ∈ Q
choose an arbitrary s ∈ Q, key[s] = 0.
while (Q ≠ ∅)
do   u = Extract-Min (Q)
     for each v ∈ Adj[u]
         do if (v ∈ Q and w(v) < key[v])
               then key[v] = w(v)   ⇒ implicity decrese-key operation
                    π[v] = u                (realisation : min heap)
```

$\Rightarrow \{(v, \pi[v])\}$ forms MST

Analysis

Handshaking $\Rightarrow \Theta(E)$ implicit Decrease-Keys

Time $= \Theta(V \cdot t_{ExtractMin} + E \cdot t_{DecreaseKey})$

- array $O(V)$, $O(1) \Rightarrow O(V^2)$.
- binary heap $O(\lg V)$. $O(\lg V) \Rightarrow O(\lg V \cdot E)$
- Fib heap $O(\lg V)$, $O(1) \Rightarrow O(V \lg V + E)$ (amortized).

- some $\delta(u,v)$ may be …

Bellman-Ford Algorithm

```
d[s] = 0
for each v ∈ V - {s}
    do d[v] = ∞
for i = 1 to |V|-1 .
    do for each edge (u,v) ∈ E
          do if d[v] > d[u] + w(u,v) .
                then d[v] = d[u] + w(u,v)
for each edge (u,v) ∈ E
    do if d[v] > d[u] + w(u,v).
          report: a negative-weight cycle.
```

$\Rightarrow$ Time $= O(VE)$.