15/12/18

Wed 11:15 + Slot A + Wed 4-5 optional

CS2005D_19

Topics:

1) ~~Asymptotic~~
2) recurrences
3) analysis of algorithm.
4) proving correctness of algorithm.

Hash tables

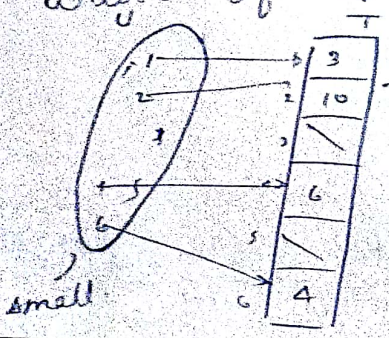
To do operations on a dictionary
i. search, insert & delete

Let $U = \{1, 2, 3 \ldots m\}$ universal set of keys (data / unique name of nodes) like key in binary search. eg. sl.no of students

① direct approach: store each data in an array corresponding to each key

(if U is small & unique)

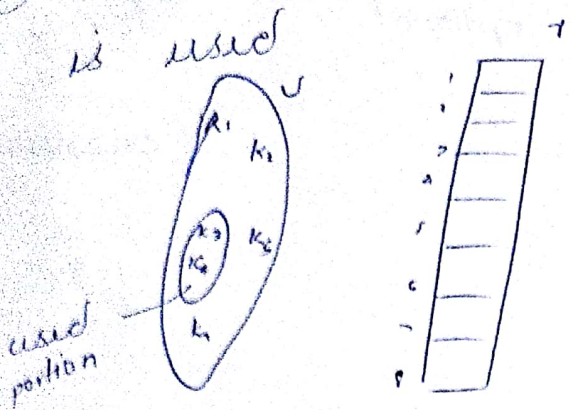
If U is large & no. of nodes is less then waste of space occurs.



→ direct-address table

but all of U is used

(2) U is large but only a small part of it is used



∫ worst case of BST, skewed to left/right → O(n)

In (1) case (running time is constant ∀ operation)

a) SEARCH(T, K)
return T[K]; but in linked list, its O(n)

b) INSERT(T, x)
T[x.key] = x.
x - node to be inserted

c) DELETE(T, x)
T[x.key] = NIL

To store data in (2) case in efficient manner we introduce hash fn.

ie $U = \{1, \ldots, 100\}$
but $|T| = 10$

for ex, define a fn that takes $\frac{\%10}{|T|}$ for each

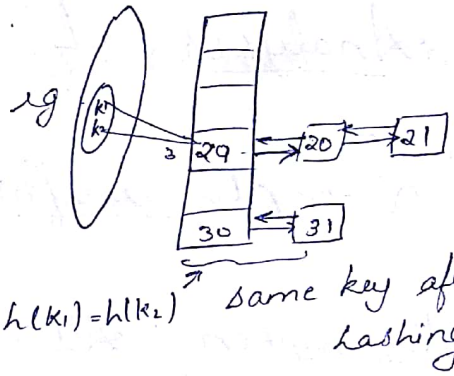
key before mapping to T
so that time of all operation remains O(1) itself

Issues

(ii) Collision: More than 1 key can map to same location after hashing

Soln:

@ chaining: keep a linked list for data with same hash value.



To search, we need to traverse the linked list corresponding to T[h(k)]. {h() is hash fn}

Here, insertion requires O(1) (@ begining of list)
search requires O(n) is worst case
deletion requires O(1) {given x in fn is pointer of list is doubly linked}

Chained-hashInsert(T, x)

insert x at the head of the list T[h(x.key)]

Chained-hashSearch(T, k)

Search for element with key k in list T[h(k)]

Chained-hashDelete(T, x) address/node → O(1)

Delete x from list T[h(x.key)]

Analysis of hashing with chaining

Simple uniform hashing

Assumption: All the key have = prob. of being mapped to a location in T (= 1/|T|) ind of the rest

⇒ for j = 0, 1, ... m−1, n_j ⇒ length of list T[j]

$n = n_0 + n_1 + \dots + n_{m-1}$ (total keys)

⇒ load factor (α)

Avg. no. of elements stored in a chain
(Avg. no. of elements with same hash)

$= n/m$

$\therefore E[n_j] = \frac{n}{m} = \alpha = n/m$

Let time req. to compute $h(k) = O(1)$.

Theorem:

In hashing with chaining, on an avg an unsuccessful search takes $O(1+\alpha)$ under simple uniform hashing

Proof:

on unsuccessful search, we need to traverse the list $T[h(k)]$, but it can have $E[n_{j}]_{j=h(k)} = \alpha$ terms or 0 terms

$$\therefore \text{avg} = E[T(h(k))] = \frac{1}{m}[T(h(1)) + \ldots]$$

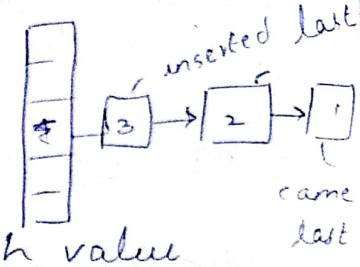
$$= \frac{1}{m}[n_0 + n_1 + \ldots n_{m-1}]$$

$$= \frac{n_0}{m} + \frac{1}{m}[n_1 + n_2 + \ldots n_{m-1}]$$

$$= \frac{n_0}{m} \alpha + c$$

$\therefore$ avg. of search $= O(1+\alpha)$ [search (α) + compute hash fn. (c)

Theorem: In chained hashing, a successful search takes an avg-case time $\Theta(1+\alpha)$ under assumption of SUH.



Proof:

→ elements of a list have same hash value

→ head of list will be inserted last & tail of list is inserted first

→ since we are searching from head, we are actually searching among those elements inserted after the key, having same hash value.

eg: if 2 is key, we search 3 & 2, but 3 is inserted after 2 but have same $h(k)$

Let x_i denote the i^{th} element inserted into table for $i = 1, 2 \ldots n$ &

Let $k_i = x_i.key$. for keys k_i & k_j we define indicator rand. var.

$$X_{ij} = \begin{cases} 1 & \text{if } h(k_i) = h(k_j) \\ 0 & \text{o/w.} \end{cases}$$

?

∴ Under SUH, (each key has eq. prob. of being mapped to a location)

$$P_r[X_{ij}=1] = P_r[h(k_i) = h(k_j)] = P_r[i \& j \in \text{same list}] = \frac{1}{m}$$

i.e., we check whether $h(k_i) = h(k_j)$ iff $\underline{j > i}$

i.e., k_j occurs after k_i

∴ expected no. of elements examined in a successful search

$$= E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} X_{ij}\right)\right] \Rightarrow ?$$

(expected no. of elements examined in @ list; same list; no. of elts after i^{th} elmnt & $\in$ same list)

X = no. of elts in a list

$$X = \sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}$$

$$\therefore E[X] = E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{1}{m} = \frac{1}{2m}\left[n^2 - n\right]$$

i.e., [no. of elements examined in a search for x is 1 more than the no. of elements inserted after x in same list]

$$= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} E[X_{ij}]\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \frac{n-i}{m}\sum_{j=i+1}^{n}\frac{1}{m}\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left[1 + \frac{n-(i+1)+1}{m}\right]$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[1 + \frac{n-i}{m} \right]$$

$$= \frac{1}{nm} \left[n + \frac{1}{m} \left[n^2 - \frac{n(n+1)}{2} \right] \right]$$

$$= 1 + \frac{1}{nm} \left[\frac{n^2 - n}{2} \right]$$

$$= 1 + \frac{1}{2m} \left[n - 1 \right]$$

$$= 1 + \frac{1}{2} \left(\frac{n}{m} \right) - \frac{1}{2n} \left(\frac{n}{m} \right)$$

$$= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}$$

∴ no. of elements examined + ~~hash fn~~ =

total time req $= \Theta\left[1 + \alpha/2 - \alpha/2n\right]$

$= \underline{\Theta(1+\alpha)}$

Hash function

eg $K \% m$ · m is size of table &
K is the key

(1) Division method

$K \% m$ → hashed value b/w 0 & m

Now, the key may not always be a number, then how k%m?

(2) Multiplication method

$h(k) = \lfloor km \rfloor \quad 0 < k < 1$

I] Now, if keys are not always a number

eg: if keys are names

then take ascii of alphabets & sum it

but then all the permutation of a key gives same hashed value.

∴ let the ascii of each alphabet b multiplu with its radix & then add.

II] Now, to divide m,

m must not be a power of 2

i.e 2^r as it ↑ no. of collision.

eg: let k = 127, 254, 161 $\quad m = 2^6$

$h = k\%m = 63$

$= 126\%64 = 62.$

$= 33$

find last p bits of k in binary

63

127 → 01111111

254 → 01111110

3) 161 $\Rightarrow$ 01 0 100001
33

2 | 161 | 1
2 | 80 | 0
2 | 40 | 0
2 | 20 | 0
2 | 10 | 0
2 | 5 | 1
2

ii the last p bits of k gives the hash value.

but its not a good hash fn.

bcoz, if k has n bits,

by keeping p last bits of k as same, we can have $(k-p)!$ keys which will have same hashed value.

$k = \underbrace{b_n\ b_{n-1} \ldots b_p}_{(n-p)! \text{ ways.}}\ \underbrace{b_{p-1} \ldots b_1}_{\text{used of } k\%m \text{ if } k = 2^p}$

Good choice $\longrightarrow$ if m is a prime no: & is not near to a power of 2

eg: for $|U| = 2000$, $\alpha = 3$, then m can be 701

(2) if $0 < A < 1$

then k ~~k~~ $= kA \% 1 = kA - \lfloor kA \rfloor$ = fractional pa of kA

Now, a good choice for A

$= \frac{\sqrt{5} - 1}{2}$ by knuth (golden ratio ?)

$= 0.61803...$

here it is good to choose table size as a power of 2 (not necessary to choose so)

$\therefore \boxed{h(k) = \lfloor km \rfloor = \lfloor (\underbrace{kA \% 1}_{\text{fractional part}}) m \rfloor}$

eg: let word size of m/c = w bits and

k fits into a single word.

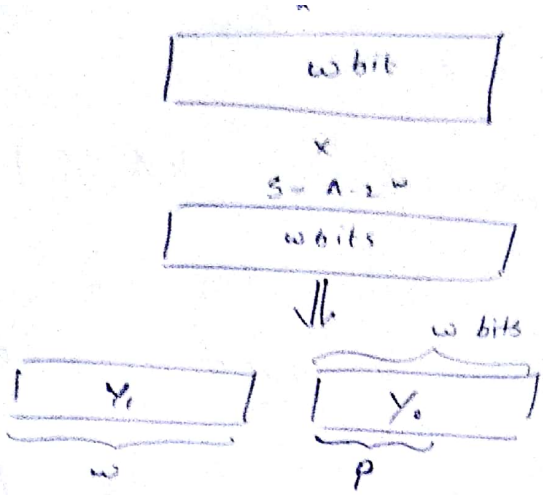
A is a fraction of the form $A = \frac{s}{2^w}$

where $0 < s < 2^w$ so that $0 < A < 1$

since 2^w is a w bit word,

$s = A \cdot 2^w$ = w bit word / integer

$\therefore$ $k \times s$ = word of max. $2 \times w$ bit size



Now extract

p bits from MSB of Y_0

= h(k)

if $A = \frac{\sqrt{5}-1}{2}$.

$S = A \cdot 2^w$ — word size of m/c

∴ K*S & extract p bits from MSB to get h(k)

eg: k = 123456, p = 14, $\frac{m}{2^p} = 16384$

$w = 32$, $A = \frac{S}{2^w} = \frac{S}{32}$.

$A = 2654435769 / 2^{32}$

$k \cdot S = 327706022297664$

$= (76300 \cdot 2^{32}) + 17612864$

Here $Y_1 = 76300$ & $Y_0 = 17612864$

14 higher order bits of Y_0 = h(k) = 67

Another approach is universal hashing.
i.e., choosing a hash fn from by random from a pool of hash fns.

Open Addressing

To avoid collision

Compute h(k). if h(k) is free in table insert it or else insert in next available space by probing for space (all data in table)

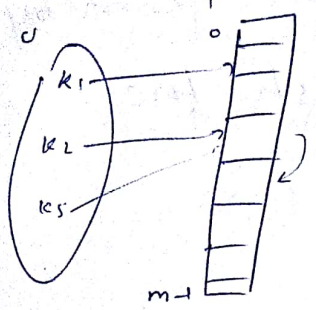
```
Hash_Insert(T, k)
    i = 0                    // probe value.
    repeat
        j = h(k, i)
        if (T[j] == NIL)
            T[j] = k
            return j
        else
            i = i + 1        // probe sequence
    until i == m;
    error "hash table overflow"
```

$h(k)$

$h: U \times \{0, 1, \ldots m-1\} \longrightarrow \{0, 1, \ldots m-1\}$

k $\quad$ i

U - set of keys



for every key k,

conditions { probe sequence $< h(k,0), h(k,1) \ldots h(k,m-1) >$

is a permutation of $< 0, 1, \ldots m-1 >$

① Linear probing

$i \in [0, m-1]$

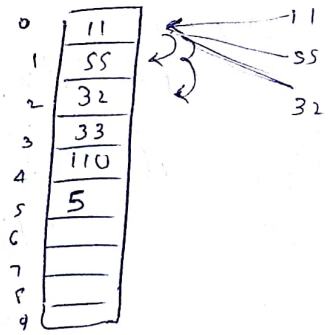
m

eg: $h' : U \longrightarrow \{0, 1, \ldots m-1\}$

$h(k,i) = (h'(k) + i) \% m$

auxiliary hash fn.

$h'(k) = k \% (m+1)$ $\quad m = 10$



$\phi k = 5, 11, 55, 32, 33, 110.$

```
Hash_search (T, k)
    i = 0
    repeat
    j = h(k, i)
    if T[j] == k
        return j
    else
        i = i + 1
    until T[j] == NIL or i == m      (NIL: not found & T not full; i == m: not found & T full)
    return NULL
```

If there is a NULL, before encountering key, then the key is absent as it was supposed to come there if present.

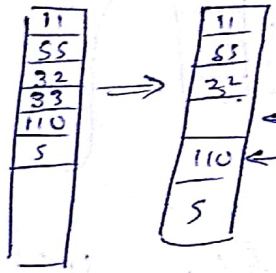
Problem:

① In deletion, if we just delete a value then search for the coming values may get affected.

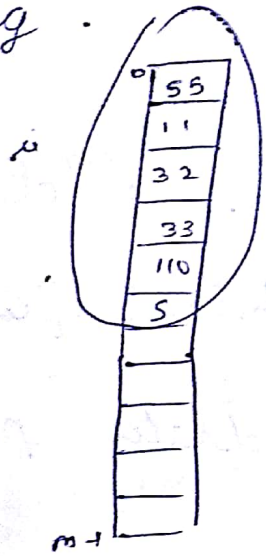
Soln: Use special symbol for deleted r

then search = 0

eg: del 33.



② another problem with linear probing is clustering.



⇒ 1° clustering

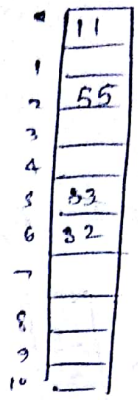
Soln: To avoid clustering use quadratic probing

i.e., $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$.

$c_1 = c_2 = 1$

let $h'(k) = k \% (m+1)$, $m = 10$

$h(k,i) = [(k \% 11) + i + i^2] \bmod 10$



issue

33 can't be mapped to any slot even though free slots are there.

ii, we must C_1 & C_2 wisely

issue 2: Can have 2° clustering.

Another approach:

(3) Double hashing: ensure diff probing seq. even if $h_1(k_1) - h_2(k_2)$ [which is not the case for linear & quadratic probing.

eg: $h(k,i) = (h_1(k) + i\,h_2(k)) \bmod m$.

Good choice

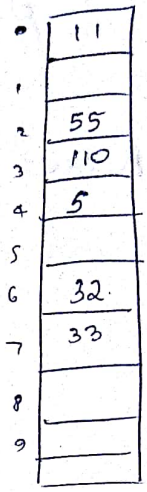
m = prime or
(m,i) are relatively prime } to map to all slots

Let $h_1(k) = k \bmod 11$

$h_2(k) = (1 + k \bmod 9)$ $\quad m = 10.$

$\therefore h(k,i) = (k \bmod 11 + i(1 + k \bmod 9)) \bmod 10$

$U = \{5, 11, 55, 32, 33, 110\}$



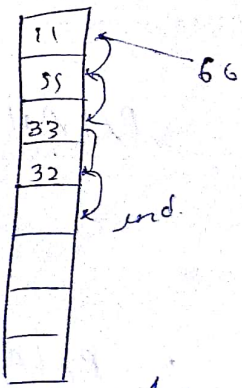
Analysis of open address hashing

Assumption

(i) S.U.H.

(ii) A unique key must have fixed probe sequence

Theorem: Given an open address hash table with $\alpha = n/m \leq 1$ (i.e., $n < m$). The expected no. of probes in an unsuccessful search is atmost $\frac{1}{1-\alpha}$ under S.U.H.



Proof:

In an unsuccessful search every probe $(h(k,i))$ but the last, access an occupied slot that does not contain the desired key and last slot

probed is empty.

Now, X — no. of probes in an unsuccessful search.

A_i — event that i^{th} probe occurs and it's to an occupied slot

Then $\{X \geq i\}$, i.e. no. of probes searched $\geq i \Rightarrow$

1, 2 ... $i-1$ probes are already occupied & its an empty slot.

i.e. $A_1 \cap A_2 \cap \ldots A_{i-1}$

∴ We can bound $Pr\{X \geq i\}$ by bounding

$$Pr\{A_1 \cap A_2 \cap \ldots A_{i-1}\} = Pr\{A_1\} \cdot Pr\{A_2/A_1\} \cdot Pr\{A_3/A_1 \cap A_2\} \ldots Pr\{A_{i-1}/A_1 \cap A_2 \ldots A_{i-2}\}$$

$Pr\{A_i\}$ = Pr{i^{th} probe occurs to an occupied slot}

$= \frac{n}{m}$ — no. of occupied slot / tot no of slot

($i = 0$ ⇒ probe happens)

$Pr\{A_2\} = \frac{n-1}{m-1}$

$Pr\{A_i / A_1 \cap A_2 \ldots A_{i-1}\} = \frac{n-(i-1)}{m-(i-1)}$ — no. of occupied slot left / tot slot unchecked

i^{th} probes occur & its to an unoccupied slot if $i-1$ probes occur

---

$\Rightarrow \frac{n-i}{m-i} \leq n/m \quad \forall i$ s.t $0 \leq i \leq m$ and $0 \leq i \leq m$

$\therefore P_r(X \geq i) = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+1}{m-i+1}$

$\leq \left(\frac{n}{m}\right)^{i-1}$

$= \alpha^{i-1}$

$\sum_{i=0}^{m-1} \frac{n-i}{m-i}$

search is unsuccessful! $\frac{n-i}{m-i}$

$E[X] = \sum_{i=1}^{n} P[X > i]$

$\leq \sum_{i=1}^{\infty} P_r[X \geq 1]$

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

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

$1 + \alpha + \alpha^2 + \cdots \alpha^{m-1}$

$P_r[A_1]$ $P_r[A_2]$ $P_r[A_3]$ $P_r[A_{m-1}]$

1st probe (i=0) sure to occur. 2nd probe $\Rightarrow$ 1st probe fails

but = $P_r[A_1] \cdot P_r[A_2]$

$= 1 \cdot \frac{n}{m}$ for any occupied slot

$\alpha < 1$

$E[X] = O(1/1-\alpha)$

[probing seq same for searching & inserting]

Corollary (for prv. theorem)

Inserting an element into an open-address hash-table with load factor α requires atmost $\frac{1}{1-\alpha}$ probes on average assuming S.U.H

Theorem: Given an open address hash table with load factor $\alpha < 1$, the expected no. of probes in a successful search is atmost

$$\frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$$

→ under SUH

→ & assuming that each key in table is equally likely to be searched for

(not like eg for 2 clustering)

Proof: If k was the $(i+1)^{th}$ key inserted to the table, then the expected no. of probes made in a search for k is atmost $\frac{1}{1-\alpha}$, $\alpha = \frac{i}{m}$ ← no. of el. before k. $i = 0, 1, \ldots, m-1$

from corollary; searching $\neq k$ inserting k. ≡ we search all elements inserted before k.

$$= \frac{m}{m-i}$$

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

Averaging over all n keys in the hash table to get the expected no. of probes in a successful search

$$= \frac{1}{n} \sum_{i=0}^{m-1} \frac{m}{m-i} = \frac{m}{n} \sum_{i=0}^{n-1} \frac{1}{m-i-\alpha}$$

$$= \frac{1}{\alpha} \sum_{k=m-n+1}^{m\alpha} \frac{1}{k}$$

$$\leq \frac{1}{\alpha} \int_{m-n+1}^{m\alpha} \left(\frac{1}{x}\right) dx$$

$$\leq \frac{1}{\alpha} \ln\left[\frac{m\alpha}{m-n+1}\right]$$

$$= \frac{1}{\alpha} \ln\left[\frac{1}{1-\alpha}\right]$$

Review of asymptotic notation

① $f(n) = O(g(n)) \Rightarrow f(n) = Og(n)$ & $f(n) = \Omega(g(n))$

also $g(n) = O(f(n))$

eg $10n^2 - 3n = O(n^2)$

$\Rightarrow c_1 n^2 \leq 10n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$

$\Rightarrow c_1 = 1$

$c_2 = 10$

$n_0 = 1$

② $3n^3 \in O(n^4) \Rightarrow 3n^3 = \Omega(n^4)$ but such c does not exist

Q. $2^{2n} \in O(2^n)$

$c_1 2^n \leq 2^n \leq c_2 2^{2n}$

$\therefore \lim_{n\to\infty} \frac{2^n}{c_2 2^{2n}} = \lim_{n\to\infty} \frac{1}{c_2 2^n} = 0$

c_2 does not exist

O -

$f(n) = O(g(n))$

$\Rightarrow f(n) \leq c\,g(n) \quad \forall\, n > n_0 \;\&\; c \in \mathbb{R}$

Ω

$f(n) = \Omega(g(n))$

$\Rightarrow f(n) \geq g(n) \quad \forall\, n > n_0 \;\&\; c \in \mathbb{R}$

$O(g(n)) \subseteq \Omega(g(n))$

$3n^3 = o(n^4)$

$f(n) \neq O(g(n)) \not\Rightarrow f(n) = \Omega(g(n))$

i $f(n) \leq cg(n) \not\Rightarrow f(n) \geq c'(g(n))$

as $f(n)$ can fluctuate

eg: sin, cos.

→ $f(n) = o(g(n)) \iff g(n) = \omega(f(n))$

(Ty) $f(n) = o(g(n)) \iff g(n) = \omega(f(n))$

Q. True / False?

$o(f(n)) \cap \omega(f(n)) = \phi$ A: T.

let $g(n) \in o(f(n)) \cap \omega(g(n))$

$\Rightarrow g(n) \in o(f(n))$

$\Rightarrow g(n) < cf(n) \ \forall c$

but

$g(n) \in \omega(f(n))$

$\Rightarrow g(n) > cf(n) \ \forall c$

not possible

$\therefore \phi$

But $O(f(n)) \cap \Omega(f(n)) = \Theta(f(n))$

Master's theorem

Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a fn and let $T(n)$ be defined on non-negative integers by $T(n) = a\,T(n/b) + f(n)$

then $T(n)$ has following asymptotic bound.

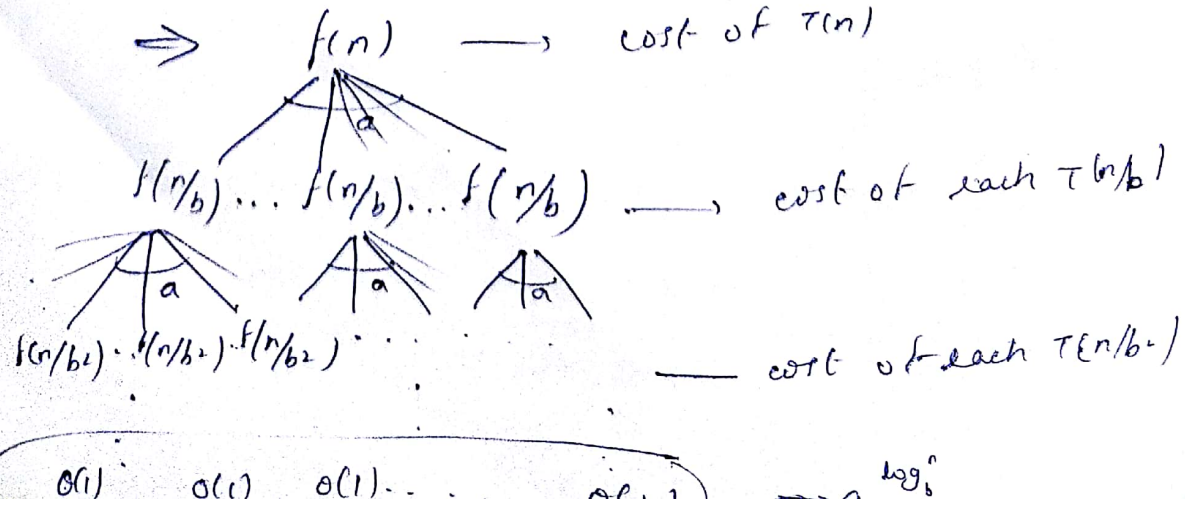
(i) If $f(n) = O\left(a^{\log_b n} / n^{\epsilon}\right)$ for some $\epsilon > 0$,

then $T(n) = O\left(a^{\log_b n}\right)$

(ii) If $f(n) = \Theta\left(a^{\log_b n}\right)$, then $T(n) = \Theta\left(a^{\log_b n} \cdot \log n\right)$

(iii) If $f(n) = \Omega\left(a^{\log_b n} \cdot n^{\epsilon}\right)$ for $\epsilon > 0$, then (and if) $af(n/b) \leq cf(n)$ for some $c < 1$ & for large n,

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



here $T(n) = \underbrace{aT(n/b)}_{\text{divide cost}} + \underbrace{f(n)}_{\text{conquer \& combine cost}}$

In each case, we compare dividing & conquering cost.

⇒ At each level, we have a^i ~~leaves~~ nodes

i.e. $a^0, a^1, a^2 \ldots$

⇒ no. of levels = $\log_b n$

∴ no. of leafs = no. of nodes at $\log_b n^{th}$ level

$= a^{\log_b n}$

∴ total cost = $\boxed{\sum_{i=0}^{\log_b n} a^i f(n/b^i) = g(n)}$

$= \underbrace{\Theta(a^{\log_b n} \cdot 1)}_{\text{for leaf node}} + \underbrace{\sum_{i=0}^{\log_b n - 1} a^i f(n/b^i)}_{\text{for internal node}}$

$f(1) = \Theta(1)$

Note: for master theorem, branching must be equal

Now, the cost is determined by.

$$g(n) = \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i)$$

① If $f(n) = O(a^{\log_b n}/n^{\varepsilon})$, $g(n) = O(a^{\log_b n})$

Proof: if $f(n) = O(n^{\log_b a - \varepsilon})$

$$g(n) \; f(n/b^i) = O\left(\left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon}\right)$$

$$\Rightarrow g(n) \le \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon}$$

$$\le \sum_{i=0}^{\log_b n - 1} \left(\frac{a\, b^{\varepsilon}}{b^{\log_b a}}\right)^i \cdot n^{\log_b a - \varepsilon}$$

$$\le n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n - 1} b^{i\varepsilon}$$

$$\le n^{\log_b a - \varepsilon} \cdot \frac{[b^{\varepsilon \log_b n} - 1]}{b^{\varepsilon} - 1} \qquad \frac{a^n - 1}{a - 1}$$

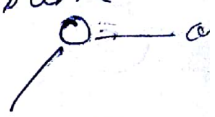
$$= n^{\log_b a - \varepsilon} \left[\frac{n^{\varepsilon} - 1}{b^{\varepsilon} - 1}\right] \qquad n = (\log_b n - 1)/0 \text{ is finite}$$

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

$$\left[\text{as } \lim_{n\to\infty} \frac{g(n)}{n^{\log_b a}} = \lim_{n\to\infty} \frac{n^{\log_b a}}{n^{\varepsilon} \cdot n^{\log_b a}} \left[\frac{n^{\varepsilon} - 1}{b^{\varepsilon} - 1}\right] = \lim_{n\to\infty} \frac{1 - 1/n^{\varepsilon}}{b^{\varepsilon} - 1}\right.$$

$$\left. \Rightarrow g(n) = O(n^{\log_b a}) \quad = \frac{1}{b^{\varepsilon} - 1} = c \right]$$

series $g(n)$ & cost for leaf are almost same



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

$\log_b^n + (1)$

② if $f(n) = \Theta(a^{\log_b^n})$, $g(n) = \Theta(a^{\log_b^n} \cdot \log n)$

Proof:

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

$$\Rightarrow f(n/b^i) = \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b^a}\right)$$

$$\therefore g(n) = \sum_{i=0}^{\log_b^n - 1} a^i \cdot \frac{n^{\log_b^a}}{b^{i \log_b^a}}$$

$$= n^{\log_b^a} \sum_{i=0}^{\log_b^n - 1} \left(\frac{a}{b^{\log_b^a}}\right)^i$$

$$= n^{\log_b^a} \sum_{i=0}^{\log_b^n - 1} 1$$

$$= n^{\log_b^a} \cdot (\log_b^n + +)$$

$$= n^{\log_b^a} \cdot \frac{\log n}{\log b}$$

$$= \Theta(n^{\log_b^a} \cdot \log n)$$

$$\therefore \text{Total} = \Theta(n^{\log_b^a}) + \Theta(n^{\log_b^a} \cdot \log n)$$

$$= \Theta(n^{\log_b^a} \cdot \log n)$$

(3) If $g(n) = \Omega(a^{\log_b^n} \cdot n^2)$, $g(n) = \Theta(f(n))$

provided $af(n/b) \leq cf(n)$

Proof: $g(n) = \Omega(f(n))$ $af(n/b) \leq cf(n)$ $c \leq 1$ & $a, b > 1$

$\Rightarrow f(n/b) \leq \left(\frac{c}{a}\right)^i f(n)$. $\Rightarrow \frac{c}{a} < 1$

$\Rightarrow a^i f(n/b^i) \overset{?}{\leq} c^i f(n)$

$\Rightarrow g(n) \leq \sum_{i=0}^{\log_b^n - 1} a^i f(n/b^i)$

$\leq \sum_{i=0}^{\log_b^n - 1} c^i f(n)$

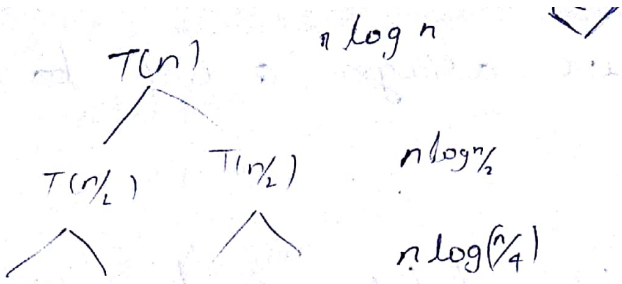
$\leq f(n) \sum_{i=0}^{\infty} c^i$

$\leq f(n) \cdot \frac{1}{1-c}$

$= \Theta(f(n))$

Use masters theorem iff $f(n)$ & $a^{\log_b^n}$ are polynomially compared. ($>, <, =$)

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



$$n \sum_{i=0}^{\log n} \log\left(\frac{n}{2^i}\right) = n \sum_{i=0}^{\log n} \log n - \log 2^i$$

$$= n\left[(\log n)^2 - \log 2 \frac{(\log n+1)(\log n)}{2}\right]$$

$$= O(n(\log n)^2)$$

Amortized analysis

```
Funct(n)
k = 0
for i = 1 to n.
    while (i<k>0) and (A[i] ≠ B[i])
        k = k-1
    A[i] = c;
    k = k + 1
```

Asymptotic ⇒ O(n)
but difficult to explain

eg: let op1 — O(n) — worst
— O(1) — on av. If we perform the m times av. case won't be tight bound.

→ In this analysis, we assign a cost to each operation.

① Aggregate analysis $\left(\frac{\text{tot. cost of } n \text{ op}}{n}\right)$

MULTIPOP(S, k) —— $O(n) = \min(s, k)$

```
while not (stack_empty(S) and k>0)
    POP(S)
    k = k-1
```

PUSH(S, x) — O(1)

POP(S) — O(1)

stack_empty(S) — O(1)

What is total cost for performing n such op.

→ We can pop(n) n times only if push is called n times

∴ we can't call multipop(S, n) for n times as our stack has only n elements.

eg ② Incrementing a binary counter

```
Increment(A)
i = 0
```

```
while i < A.length and A[i] == 1
    A[i] = 0
    i = i+1
if i < A.length
    A[i] = 1;
```

} works till 1st zero

} the 1st zero from LSB is made 1

bit 7 6 5 4 3 2 1 0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1
⋮

(a) Aggregate analysis

0th bit changes every time
1st bit every other time
2nd bit every 4th time
⋮

$\therefore$ total cost $= \sum_{i=0}^{k-1} \lfloor n/2^i \rfloor$

$< n \sum_{i=0}^{\infty} \frac{1}{2^i}$

$= 2n = O(n)$

where n = total no. of case
$= 2^k$
no. of times to increment '' is called to get a unique output

$\therefore$ Amortised cost $= \frac{\text{cost}}{\text{no. of operation}}$

$= \frac{cn}{n} = \underline{O(1)}$

(b) Accounting method

while we assign cost to an operation, some might be overcharged and some undercharged

let $\hat{c_i}$ — Amortized cost of i^{th} operation

c_i — Actual cost

$$\text{Credit} = \hat{c_i} - c_i$$

used to cover undercharged op

We require;

$$\underbrace{\sum_{i=1}^{n} \hat{c_i}}_{\text{amortized cost}} \geq \underbrace{\sum_{i=1}^{n} c_i}_{\text{actual cost}}$$

$$\therefore \text{Total credit} = \sum_{i=1}^{n} \hat{c_i} - \sum_{i=1}^{n} c_i \geq 0$$

In Multipop,

let Amortized cost of push = 2.

" pop = 0

" multipop = 0

then at each n push, 2 is credited

& popping & multipop will have 0 cost as its cost is carried by the credits got from push

---

In increment algorithm,

during each increment, $0 \rightarrow 1$ occurs only once & $1 \rightarrow 0$ occurs as many as $1 \rightarrow A.length - 1$

$\therefore$ Let amortized cost of $0 - 1 = 2$.

& $1 \rightarrow 0 = 0$ (credited by $0 \rightarrow 1$ flip.)

$\therefore$ total cost = 2 (no. of op) (no. of times incremented)

(2 is cost for $0 \rightarrow 1$ only; at each op)

$= 2 \cdot n$

$= O(n)$

$\therefore$ Amortized cost = $\frac{O(n)}{n} = \underline{\underline{O(1)}}$

(c) Potential method.

$\phi(D_i)$ ← potential for Data structure after i^{th} operation

$\phi(D_i) - \phi(D_{i-1}) \Rightarrow$ potential difference

In case of stack op, ϕ can be size of stack.

Operation ⇒ invoking a fn or step

Amortized cost $= \hat{c_i} = c_i + P \cdot D$

$= c_i + [\phi(D_i) - \phi(D_{i-1})]$

(c_i: actual cost)

$$\sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} [c_i + [\phi(D_i) - \phi(D_{i-1})]]$$

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

---

eg: In multipop, let

$\phi(D_i) \Rightarrow$ no. of elements in stack after i^{th} operation.

$$\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})$$

let $\phi(D_{i-1}) = s$

for push, $\hat{c_i} = c_i + (s+1) - s$

$= c_i + 1$

if actual cost $c_i = 1$

$\underline{\hat{c_i} = 2}$

for P, $\hat{c_i} = c_i + (s-1) - s$

$= c_i - 1$

if $c_i = 1$, $\hat{c_i} = 0$ //

for multipop :-

$$\hat{c_i} = c_i + \left(s_{i-1} - \min(s, k)\right) - s_i$$

$$= c_i - \min(s, k)$$

$$= k - k = 0$$

s - size of stack
k - elements to be popped

[c_i = actual cost = 1 × min (s, k) (no. of pops
* actual cost of 1 pop

Note :

(i) $\phi(D_n) \geq \phi(D_0)$ (defn ϕ so that its true)

(ii) $\sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i$

for $\phi(D_i) \geq \phi(D_0)$, $\forall i$, we have to pay in advance

(iii) Usually, $\phi(D_0) = 0$

---

eg ② Incrementing a binary counter

Let $\phi(D_i)$ = no. of 1's in counter after i^{th} op

= b_i

Now, Assume that i^{th} operation resets t_i bits.
but each op. have only 1 set operation

∴ Actual cost of i^{th} op = 1 set + t_i resets

let cost of $0 \rightarrow 1 \Rightarrow 1$, $1 \rightarrow 0 \Rightarrow 1$ $\Rightarrow c_i = 1 + t_i$

$\Rightarrow$ If $b_i = 0$, i.e. no. of 1's after i^{th} op = 0

$\Rightarrow$ i^{th} operation resets all bits

$\Rightarrow$ $b_{i-1} = t_i - k$.

b_{i-1}: 1's before i^{th} op; t_i: bits reset after i^{th} op; k: for no. of bits.

$\Rightarrow$ else if $b_i > 0$ (atleast one 1)

$\Rightarrow$ $b_i = b_{i-1} - t_i + 1$

b_i: 1's after i^{th} op; b_{i-1}: 1's before i^{th} op; t_i: no. of bits reset $1 \rightarrow 0$; bit set

b_{i-1} − no. of 1's set to 0 from initial no. of 1's (+1) = no. of 1's after i^{th} op

11111 − 11110 − t_i + 1 (4 − 0 + 1)

11000 (2) 10(11) − 3 + 1 (= 5 − 4 + 1; b_{i-1}, t_i)

$\Rightarrow$ $\phi(D_i) - \phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$

∴ Amortized cost

$$\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})$$

$$\leq (1 + t_i) + (1 - t_i)$$

$$= 2.$$

If the counter starts at zero, then $\phi(D_0) = 0$ since $\phi(D_i) \geq 0 \; \forall i$, total amortized cost of a sequence of n increment operation is an upper bound for total actual cost.

Worst case of n increment op. is $O(n)$.

Assume that the counter is not starting from zero and let it have b_0 1's initially and after n increment operation it has b_n 1's where $0 \leq b_n, b_0 \leq k$.

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

We have $\hat{c_i} \leq 2. \; \forall \; 1 \leq i \leq n$, $\phi(D_n) = b_n$ & $\phi(D_0) = b_0$

Total actual cost $= \sum_{i=1}^{n} c_i - \sum_{i=1}^{n} \hat{c}_i - (\phi(D_n) - \phi(D_0))$

$\leq \sum_{i=1}^{n} 2 - b_n + b_0$

$= 2n - b_n + b_0$

$\therefore O(n)$

Dynamic tables

```
Table. Insert (T, x)
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.num == T.size
    allocate new-table with 2 × T.size slots
    insert all items in T.table into new-table
    free T.table
    T.table = new-table
    T.size = 2 × T.size
insert x into table
T.num = T.num + 1
```

if x is inserted into table with free slot

$c_i = O(1)$

else. if x is inserted in i^{th} operation & table is full.

then we have to copy $i-1$ elements into new table & insert x

$\therefore$ cost $= O(i)$

$\therefore c_i = \begin{cases} i & \text{if } (i-1) \text{ is exact power of 2} \\ & \text{(table is full)} \\ 1 & \text{o/w} \end{cases}$

$\therefore$ total cost of n table-inserts.

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} (1) + \sum_{j=0}^{\log n} 2^j \quad \text{(log of no of } 2^i\text{)}$$

$$= n + 2n.$$

$$= 3n$$

$\therefore$ Amortized cost for an operation

$$= \frac{3n}{n} = O(3)$$

In terms of accounting method.

$\hat{c_i} = 3$

1 for inserting n.

rest for copying x into new table

whenever account of x becomes 0, then new x/2 elements account provide this cost

In potential method

Let $\phi(T) = 2 \times T.num - T.size$

When table is full $\phi(T) = T.size$ $(T.num = T.size)$

When table is newly allocated.

$\phi(T) = 0$ $\left[T.num = \frac{T.size}{2}\right]$

else $\phi(T)$ constantly ↑ with T.num

	c_i (actual)	$\phi_i - \phi_{i-1}$ (PD)	$\hat{c}_i = c_i + P.D$ (amortized)
ith insertion doesn't expands table	1	2 $[2 \cdot 3 - 4 - 2 \cdot 2 + 4$ $2 \cdot T.num_i - T.size_i -$ $(2 \cdot T.num_{i-1} -$ $T.size_{i-1})$ but $size_i = size_{i-1}$ $num_i = num_{i-1} + 1$	3
Table expands	num_i	$num_i = num_{i-1} + 1$ $size_i = 2 \cdot size_{i-1}$ $size_{i-1} = num_{i-1}$ $\therefore$ $3 - num_i$	3

Amortized analysis

If we have a multi-push operation along with multipop, what will be the amortized cost

It can't follow the earlier approach

worst case can be $\underline{O(n^2)}$ - $O(n)$

Ill^rg If we can have inc & decrement op in binary counter. & case where all bits flips every time ($1111 \underset{-1}{\overset{+1}{\rightleftharpoons}} 0000$)

1
0
10
11
10

Also, if we consider deletion also as a part of dynamic table, the we have to follow diff approach (if we ↓ size when T.num = T.size/2 then ∃ a situation where addition & deletion of element to a complete table which take i in each step)

∴ Use GRAPHS

$G = (\{V\}, \{E\})$

Directed & undirected graph.

⇒ adjacent ⊂ 2

⇒ Degree { In / out

Let $V = \{V_0, V_1, \ldots V_n\}$

⊙ path ⇒ if ∃ edge b/w V_i & V_{i+1}.

⊙ Simple path ⇒ every vertex is distinct.

⊙ cycle ⇒ 2 or more same vertices of a path.

⊙ Simple cycle ⇒ $V_0 = V_n$ & simple path.

⇒ isolated vertex

⇒ subgraph $G' = (V', E')$

⇒ $V' \subseteq V$ & $E' \subseteq E$ & E' is formed from V'

⇒ Connected graph ⇒ path b/w every pair of vertex.

⇒ Strongly connected ⇒ In <u>directed graph</u>

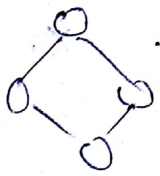
A → B → C

weakly connected as

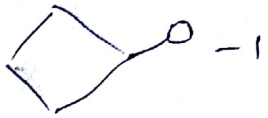
A, C has a path but

C, A doesn't have one

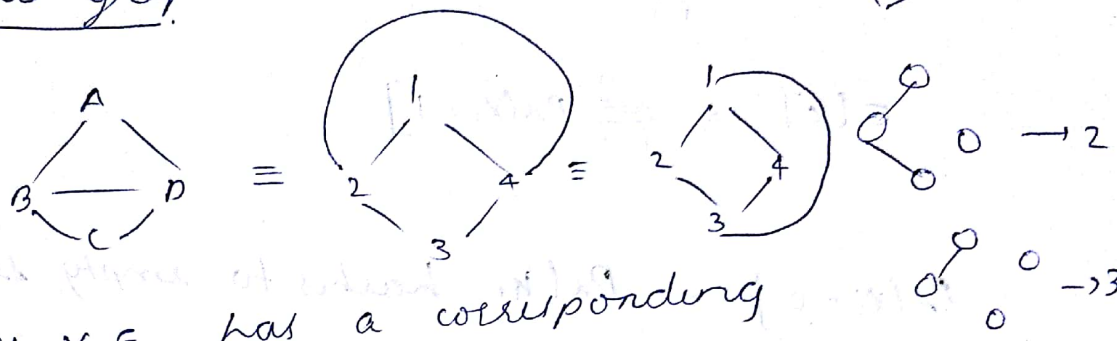
Connected component →



isolated comp. is connected.



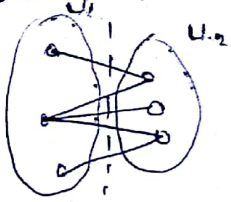
Isomorphic graphs.



every $v_i \in$ has a corresponding $v'_i \in$

Bipartite graph : ÷ graph x 2 sets U_1 & U_2

s.t $\forall \in (v_i, v_j)$ $v_i \in U_1$ & $v_j \in U_2$



Complete graph : an edge b/w every vertices

$(K_1, K_2 \ldots)$ ($^nC_2$ edges)

comp. graph with 1 vertices

Q In a chained hash table of m size, if there are n elements, what is expected no. of collision ?

A: let $k_1 \ldots k_n$ be the keys. let

X be a rand. var indicating no. of collisio

$X_i = \begin{cases} 1 & \text{if } k_i \text{ had a collision} \\ 0 & \text{o/w} \end{cases}$

$$X = \sum_{i=1}^{n} X_i$$

$$E[X] = \sum_{i=1}^{n} P_r[X_i = 1]$$

$P_r[X_i = 0]$ = $P_r[k_i$ hashes to empty slot$]$

= {expected no. of empty slots after $i-1$ elements were inserted}

= $m \cdot P_r[j$ is empty before i^{th} insertion$]$

$$= \left(\frac{m-1}{m}\right)^{i-1}$$

$$\therefore E[X] = \sum 1 - \sum P_r[X_i = 0]$$

$$= n - \sum \left(\frac{m-1}{m}\right)^{i-1}$$

$$= n - \frac{\left[1 - \left(\frac{m-1}{m}\right)^{n}\right]}{1 - \frac{m-1}{m}}$$

$$= n - m\left[1 - \right.$$

$\frac{1}{1}$

$\frac{1}{m} + \frac{2}{m}$

$\frac{1}{m} + \frac{2}{m} + \frac{3}{m}$

$1 + 2 + \dots + i$

$\frac{i(i+1)}{2}$

$\sum_{i=1}^{n} \frac{i(i+1)}{2}$

$\frac{1}{2}\left[i^2 + i\right]$

$\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}$

$\frac{n(n+1)}{2}\left[\frac{2n+1}{3} + 1\right]$

$= \frac{n(n+1)(n+2)}{3}$