Simran Pandey
15/1296

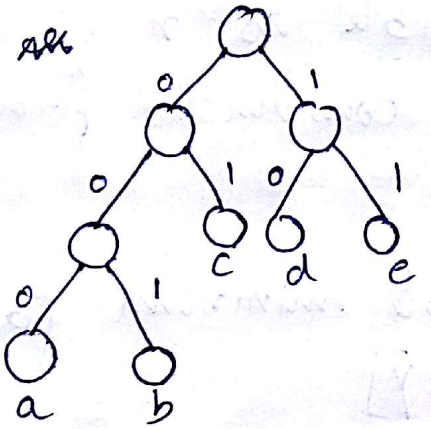
mid-Sem Test (Advanced Algo)

Ans 1 (a) with the help of an example explain how a prefix code is generated using a binary tree.

→ consider that we have a binary Tree T with n leaves
→ an alphabet S having n characters.

now we know that in binary Tree T we follow the path from root to the leaf labelled x, $\forall x \in S$

Each edge is



when we go left from a node from parent node we write 0 and when we go right from parent node we write 1.

In the diagram above

Prefix code of root of a = 000
[taking 3 successive left hand edges starting from root]

Similarly b = 001

c = 01

d = 10

e = 11

(B) Prove that the encoding of s constructed from T is a prefix code.

→ In order for encoding of x_i to be a prefix of the encoding of y, the path from the root to x would have to be a prefix of the path from the root to y.
which is, it means that x lies on the path from the root to y.
but this wont be possible as x is a leaf.
∴ The encoding of s constructed from T is a prefix code.

(ii) Let OPT(i,j) denote minimum cost of an alignment b/w x & y]
where $x_1, x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$ are two sequences.

→ For an optimal alignment M,
atleast one of the following is true,

(i) if $(i,j) \in M$
In this case, the mismatch cost is $\alpha_{x_i y_j}$
and then we align
$x_1, x_2 \ldots x_{i-1}$ and $y_1, y_2 \ldots y_{j-1}$
∴ $OPT(i,j) = \alpha_{x_i y_j} + \alpha_{x_i y_j} + OPT(i-1, j-1)$

(ii) If the i^{th} position of first sequence is not matched.
The gap cost is δ and we align $x_1, x_2 \ldots x_{i-1}$ and $y_1, y_2 \ldots y_j$

~~OPT(m,n) = δ + OPT(m-1,n)~~

$\therefore OPT(i,j) = \delta + OPT(i-1, j)$

(ii) if the j^{th} position of the second sequence is not matched

we pay a gap cost of δ, we get

$$OPT(i,j) = \delta + OPT(i, j-1)$$

The recurrence corresponding to the minimum alignment cost is

$$OPT(i,j) = \min [\alpha_{x_i y_j} + OPT(i-1, j-1),$$
$$\delta + OPT(i-1, j) + \delta + OPT(i, j-1)]$$

[for $i \geq 1$ and $j \geq 1$]

(iii) Now let me know that appending an element in a merged list takes $O(1)$ time

Similarly, Appending n elements into the merged list will take $O(n)$ time

So, the merging procedure takes $O(n)$ time

For simultaneous sorting & counting the no. of inversions in a list of n elements it runs in $O(n \log n)$ time.

acc. to recurrence

$$T(n) \leq 2T(n/2) + n$$

So, the algorithm runs in $O(n \log n)$ time for a list with n elements.

(iv) (A) Amortized cost of insert operation on a Fibonacci heap is $O(1)$

The actual cost is $O(1) \Rightarrow c_i = O(1)$

increase in Potential $= 1$ $(\Delta\phi)$

$$\hat{c}_i = O(1) + 1 = O(1)$$

[where $\hat{c}_i = c_i + \Delta\phi$.]

and $\Delta\phi = \phi(H') - \phi(H)$

(B) Amortized cost of delete min op^n

→ Total amt of work done in extracting the min node → $O(D(n) + t(H))$

→ Potential before → $t(H) + 2m(H)$

" afterwards → $\underbrace{(D(n)+1)}_{\text{at most } D(n+1) \text{ roots remain}} + \underbrace{2m(H)}_{\text{no root is marked.}}$

Amortized cost = $O(D(n) + t(H)) + ((D(n)+1) + 2m(H) - ((t(H) + 2m(H))$

$= O(D(n))$

where $D(n)$ is an upper bound on the max. degree of any node in heap

$O(\lg n) = O(D(n))$

Ans 6. The potential func^n $\phi(H)$ of fibonacci heap H is defined by

$$\phi(H) = t(H) + 2m(H)$$

where $t(H)$ is the no. of trees in the root list of H and $m(H)$ is the no. of marked nodes in H.

T.P:- max degree or $D(n)$ of any node in fib heap is bounded by $\log_\phi D(n) = O(\log n)$.

let $k = x$.degree where x be any node in an n-node fibonnacci heap

→ at first we'll prove that

$$n \geq \text{size}(x) \geq \phi^k$$

where $\phi = (1+\sqrt{5})/2$

→ S_k = min possible size of any node of degree K

→ $S_0 = 1$ and $S_1 = 2$

→ S_k is at most size(x).

now consider a node z s.t

z-degree = K and size(z) = S_k

$S_k \leq$ size(x), we compute a lower bound on S_k.

now let $y_1, y_2, \ldots y_k$ denote the children of z in the order in which they were linked to z.

To bound S_k, we count one for z itself and one for y_1 (size(y_1) ≥ 1). we get:

$$\text{size}(x) \geq S_k \geq 2 + \sum_{i=2}^{k} \text{size}(y_i).\text{degree}$$

$$\geq 2 + \sum_{i=2}^{k} s_{i-2}$$

where [y_i. degree $\geq i-2$]

By induction on k, we show that

$$s_k \geq F_{k+2} \quad \forall \text{ non-ve int } k.$$

Induction step:

we assume that $k=2$ and $s_i \geq F_{i+2}$

for $i = 0, 1, \ldots k-1$

$$s_k \geq 2 + \sum_{i=2}^{k} s_i - 2$$

$$\geq 2 + \sum_{i=2}^{k} f_i$$

$$= 1 + \sum_{i=0}^{k} f_i$$

$$= F_{k+2} \geq \phi^k$$

$\therefore$ size $(x) \geq s_k \geq F_{k+2} \geq \phi^k$

take base $-\phi$ log and we get $k \leq \log_\phi n$

$$\Rightarrow k \leq \lfloor \log_\phi n \rfloor$$

Hence the max degree $D(n)$ of any node in a Fib heap is bounded by $D(n) = O(\lg n)$.

$$D(n) = O(\lg n)$$

Ans 25 Dynamic Table are

- Variable - size list data structure that allows elements to be added or removed.
- They overcome limit of static arrays which have a fixed capacity that needs to be specified at allocation.

→ When should we expand a dynamic table?

- when all space in the array is consumed & an additional element is to be added
- array expanded by constant proportion by allocating a new table.
- new table's size is twice the size of present & copying prev. elements.

→ When should we contract a dynamic table.

- when we remove some elements & if size drops below 1/4 of present size
- we contract the table deallocating the old table & creating new table of 1/2 of size

→ Potential func^n of dynamic table

$$\Phi(T) = 2 \cdot T.num - T.size$$

[T.num = no. of items &
T. size = size of table]

Amortized Cost of i^{th} Table-insert Operation

num_i = no. of items stored in table after i^{th} opⁿ

$size_i$ = total size after i^{th} opⁿ

ϕ_i = potential after i^{th} operⁿ

initially $num_0 = 0$ $size_0 = 0$, $\phi_0 = 0$

without Expansion

$size_i = size_{i-1}$ ⟶ (1)

$size_{for\ i^{th}\ op}$ = size of prev opⁿ

[Amortized cost] $\hat{c}_i = c_i + \phi_i - \phi_{i-1}$

$= 1 + (2 \cdot num_i - size_i) - (2(num_{i-1} - size_{i-1}))$

$= 1 + 2\,num_i - size_i - (2(num_i - 1) - size_i)$

$= 1 + 2\,num_i - size_i - 2\,num_i + 2 + size_i$

$= 3$

With Expansion

$Size_i = 2 \cdot Size_{i-1}$ — ①

$Size_{i-1} = num_{i-1} = num_i - 1$ — ②

↓ Size before Expansion

↓ no. of items → same before Eqn (Table got full) →

from ① & ②

$Size_{i+1} = 2 \cdot (num_i - 1)$

$\hat{c}_i = c_i + \phi_i - \phi_{i-1}$

$num_i + (2 \cdot num_i - Size_i) - (2 \cdot num_{i-1} - Size_{i-1})$

↑ to transfer new elements

$\times [num_i + 2 \cdot num_i - 2num_i + 2 - 2num_i + 2 + num_i + 1]$

$\Rightarrow$

$num_i + (2 \cdot num_i - 2(num_i - 1)) - (2 \cdot (num_i - 1) - (num_i - 1))$

$num_i + 2 \cdot num_i - 2num_i + 2 - 2 \cdot num_i + 2 + num_i - 1$

$4 - 1$

$\hat{c}_i = 3$

TABLE- DELETE

→ delete causes to have 1 element lesser than prev op^n

$num_i = num_{i-1} - 1 \rightarrow ? \; num_i + 1 = num_{i-1}$

if $\alpha_{i-1} \not< 1/2$ [it might or might not contract]

→ if it does not

$Size_i = Size_{i-1}$ [table does not contract]

$$\hat{c}_i = a_i + \phi_i - \phi_{i-1}$$

$$1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right)$$

$$1 + \frac{size_i}{2} - num_i - \frac{size_i}{2} - num_{i-1}$$

$$= 2.$$

if $\alpha_{i-1} < 1/2$ & i^{th} op^n trigger a contraction

(actual cost) $c_i = num_i + 1$

[move num_i items] [delete one item]

① → $Size_i/2 = Size_{i-1}/4$ → table becomes half

② — $num_{i-1} = num_i + 1$

$$\hat{c_i} = c_i + \phi_i - \phi_{i-1}$$

$$= num_i + 1 + \left(\frac{Size_i}{2} - num_i\right) - \left(\frac{Size_{i-1}}{2} - num_{i-1}\right)$$

~~$= num_i + 1 +$~~

from ① & ②

$$\frac{Size_i}{2} = num_i + 1$$

$$\frac{Size_{i-1}}{2} = 2(num_i + 1)$$

On replacing

$$= num_i + 1 + (num_i + 1 - num_i) - (2num_i + 2 - num_{i-1})$$

$$num_i + 1 + num_i + 1 - num_i - 2num_i - 2 + num_i + 1$$

$$3 - 2$$

$$= 1.$$

~~Amortized cost of insert~~

Sol 4. (i) T.P - Binary tree corresponding to the optimal prefix code is full (ii)

Proof :- Let T be the Binary tree corresponding to the optimal prefix code

→ Suppose it contains a node u with exactly one child v.

now we convert T into T' by replacing the node u with v. there can two possibilites

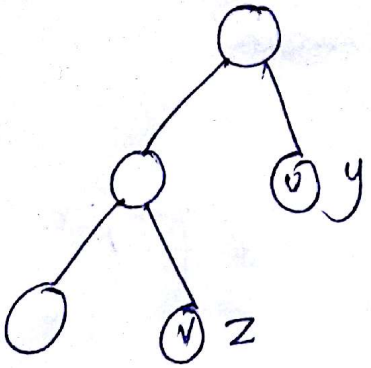
(i) If u is the root of the tree
we delete node u and make v the root of v

(ii) If u is not the root
let w be the parent of u in T
we delete node u in and make v the child of w instead of u.

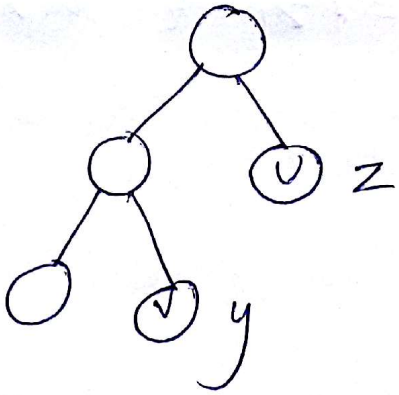
→ This step decreases the no. of bits needed to encode any leaf in the subtree rooted at node u & doesn't affect other leaves.

So the prefix code of T' has a smaller ABL than that of T which contradicts the optimality of T.

(ii) To Proof: $f_y \geq f_z$



suppose $f_y < f_z$. Consider the code obtained by exchanging labels at nodes v & v



On exchanging, the effect of change is as follows, multiplier of f_y increases from depth(v) to depth(v) and multiplier of f_z decreases from depth(v) to depth(v)

ABL after the change is

= ABL Before change − (depth(v) f_y + depth(v) f_z) + (depth(v) f_z + depth(v) f_y)

⇒ Let ABL_2 be ABL After change and ABL_1 be ABL before change, so

$ABL_2 = ABL_1 - \text{depth}(v) f_y - \text{depth}(v) f_z + \text{depth}(v) f_z + \text{depth}(v) f_y$

$= ABL_1 + (\text{depth}(v) - \text{depth}(v))(f_z - f_y)$

$= ABL_1 + (\underbrace{\text{depth}(v) - \text{depth}(v)}_{+ve})(f_y - f_z)$

we know that this term is +ve

if $fy < fz$

then $fy - fz$ is -ve, which would mean

$ABL_2 < ABL_1$

contradicting the supposed optimality of previous code before the change.

$\therefore fy \geq fz_1$ & hence proved.