Page No.
Date : / /

Name: Pratiksingh Rajeshsingh Thakur
Reg. No.: 2019BCS133
Roll No.: A 63
Division : A
Subject: Advanced Algorithm

ISE 2

Q.1 What is meant by amortised analysis? Explain in detail amortised analysis of Binary Counting Problem using accounting and potential methods.

Ans⇒ Amortised Analysis:

Amortised analysis is a worst case analysis of a sequence of operation — to obtain a tighter bound on the overall or average cost per operation in a sequence that is obtained by separately analyzing each operation in the sequence.

There are 2 methods of amortised analysis:
1) Accounting method
2) Potential method

Page No.
Date : / /

★ Binary Counting Problem:

A:	5	4	3	2	1	0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
:						
:						
n						

A binary k-bit counter can be implemented using a k-element binary array. The counter is initially 0.

The only operation is increment(), which adds 1 to the current number in the counter.

```
procedure increment
    i=0;
    while A[i]==1 & i < A.length do A[i]=0
                i=i+1
    end while
    if (i < A.length)
        A[i]=1
```

If we increment starting at 0, all the way upto (n-1). In worst case for each increment the complexity is $\log n$.

Page No.
Date : / /

and for the entire counting problem, the complexity is O(nlogn)

Amortise analysis will consider global picture some incrementing steps are taking too much time it is amostised space over other increment steps which are taking less time each of step is taking constant time.

(i) Accounting method: In the accounting method of amortised analysis we assign differing charges to different operations, with some operation charged more or less then the actual cost. The amount we charge on operation is amortised cost.

The difference to specific objects in the data structure as credit. Credit can pay for later operations whose amortised cost is less than their actual cost.

(i) Assume, the amortised cost of changing 0 to 1 as Rs. 2.

(ii) Similarly Amortised cost of 1 to 0 as Rs.0

(iii) When there is transition from 0 to 1, then Rs 1 convens actual cost Rs 1 stays with the bit which is 1

Page No.
Date : / /

For n increment operation the total amortised cost is O(n) which bounds the total actual buy cost.

(ii) Potential Method:

Instead of representing prepaid work as credit stored with specific object in the data structure, the potential method of amortised analysis represents the prepaid work as "potential energy" or just "potential" which can be released to pay for future operations. We associate the potential with the data structure as a whole rather than with the specific object within the data structure.

The potential method works as follows:

c_i = actual cost for i^{th} operation

D_i = Data structure after applying the i^{th} operation to data structure D_{i-1}

A potential function ϕ maps each data structure D_i to a real number $\phi(D_i)$ which is the potential associated with data structure D_i

$\phi(D_i)$ = potential of D_i

D_i = accumulated charge

Page No.
Date : / /

Amortised cost = a_i

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

for all n no. of operations

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

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

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

If ϕ is selected such that $\phi(D_0) = 0$

and $\phi(D_n) \geq 0$

$$\sum_{i=1}^{n} a_i = \left(\sum_{i=1}^{n} c_i\right) + \left(\phi(D_n)\right)$$

$$\therefore \boxed{\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} a_i}$$

Page No.
Date: / /

Applying to Binary Counting problem:

Define $\phi(D_i) = b_i$

where b_i is no. of 1's in the binary string

t_i = No. of trailing 1's (consecutive)

0	0	0	0	0 0
0	0	0	0	0 1
0	0	0	0	1 0
0	0	0	0	1 1
0	0	0	1	0 0
0	0	0	1	0 1
0	0	0	1	1 0
0	0	0	1	1 1

$\phi(D_i) - \phi(D_{i-1}) = b_i - b_{i-1}$

$= (b_{i-1} - t_{i-1} + 1) - b_{i-1}$

$= 1 - t_{i-1}$

actual cost: $c_i = t_{i-1} + 1$

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

$= t_{i-1} + 1 + 1 - t_{i-1}$

$$\boxed{a_i = 2}$$

i.e Amortised analysis $a = O(n)$ which is upper bound to actual cost.

$$\boxed{\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} a_i \leq 2n}$$

Page No. _____
Date : / /

(Qu 2) Discuss in detail fibonacci heap explain all types of its operations with the help of suitable examples and respective amortised and actual time complexity analysis. Also states its uses in computer algorithm / problems.

Fibonacci heap is a data structure for priority queue operations consisting of a collection of heap-ordered trees. It has better amortised running time than many other priority queue data structure including binary heap & boromial heap.

A fibonacci heap is a collection of rooted trees that are min-heap ordered that is each tree obeys the min heap property the key is greater than or equal to the key of its parent.

Node structure of fibonacci heap:

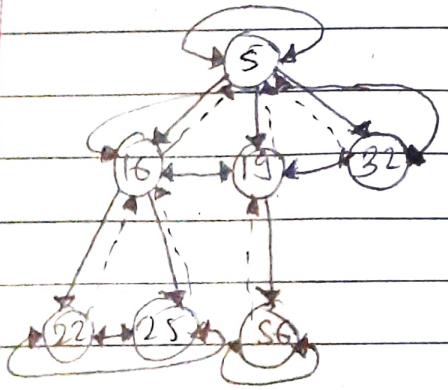
It has 4 types of pointers
(i) parent pointer
(ii) child pointer
(iii) Right sibling
(iv) left sibling

Page No.
Date : / /

It supports the following five operation in which each element has a key:

(i) make-heap(): It makes the heap from scratch. Amortised complexity: O(1)
(ii) insert (H, x): Insert node x in heap H. It has complexity of O(1)
(iii) minimum (H): Returns minimum value over all available values. It has complexity of O(1).
(iv) Meld (H_1, H_2): Merges two heaps into a single heap. It has a complexity of O(log n)
(v) delete_min (H): Extract minimum from the heap and delete it. It has a complexity of O(log n)
(vi) delete (H, x): Delete node x, O(log n)
(vii) Decrease_key (H, x, Δ): assigns to element x within the heap the new key value less by Δ. It has complexity of O(1)

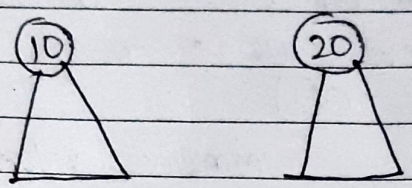
* Structure of Heap: fibonacci heap is a collection of ordered heap tree. A node is less than or equal to all its children.



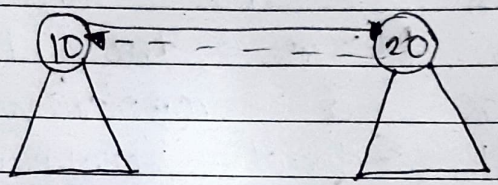
Page No. _____
Date : __/__/__

Simple Sub operations:

(i) Linking:- Given two heap ordered trees, make root with the bigger key the child of the other root.

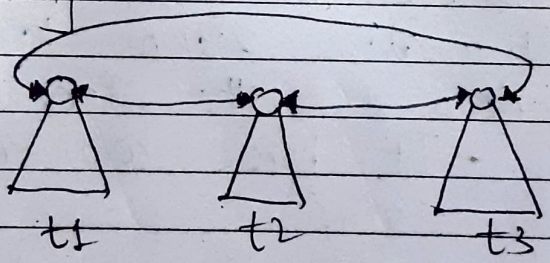


Here we make 20 as a child of 10

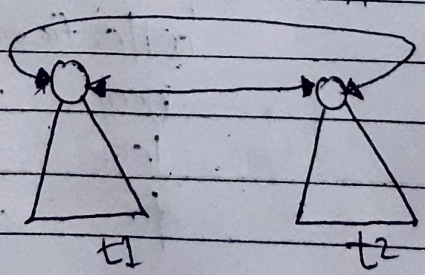


Relation between them is parent, child. It has complexity of O(1)

(2) Unlinking:

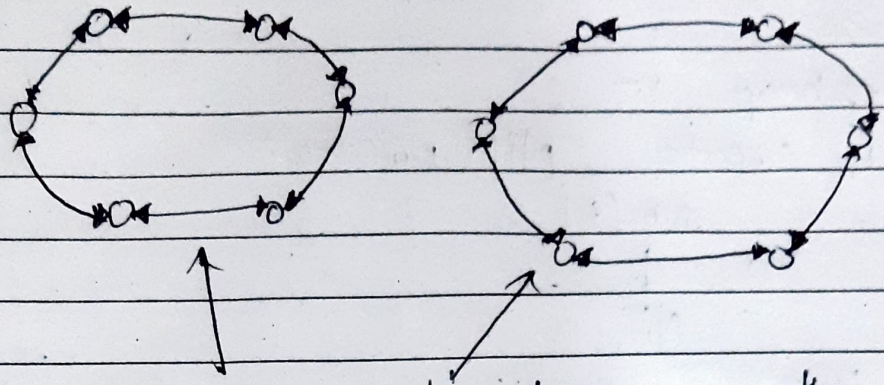


we want to unlink t_2

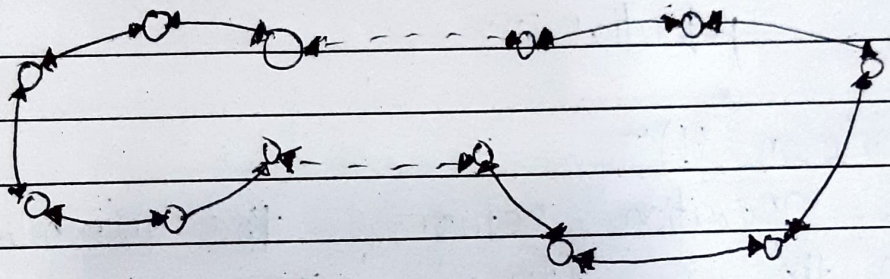


Page No.
Date : / /

(3) Meoging of cycles:



We need to merge the 2 cycles



It has complexity of O(1)

* Potential function of fibonacci series:-

Let $x_1, x_2, \ldots, x_n$ be the sequence of operations. Let $t_i(H)$ = the no. of roots of roots in the root list after operation z_i

$M_i(H)$ = No. of marked nodes in H after the operation z_i (some of the nodes of fibonacci heap are marked

$P_i = t_i(H) + 2 \times m_i(H)$

Page No.
Date : / /

condition:

$P_0 = 0$

$P_n \geq 0$ a

(i) make_heap :-

(i) create Nil pointer

min (H)

↓

Nil

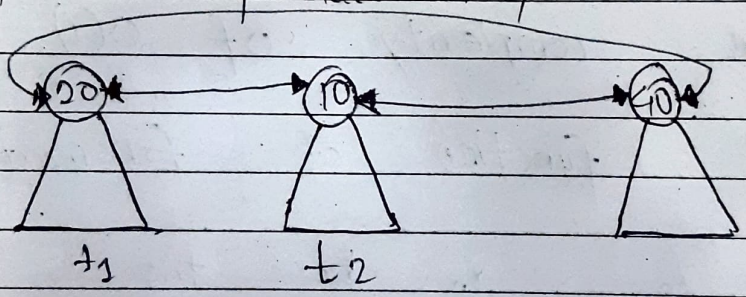
This operation has cost = O(1)

no. of roots = 0 ; No. of marked nodes = 0,

so potential = 0

(2) minimum (H):-

This operation, returns the minimum key from the fibonacci heap.

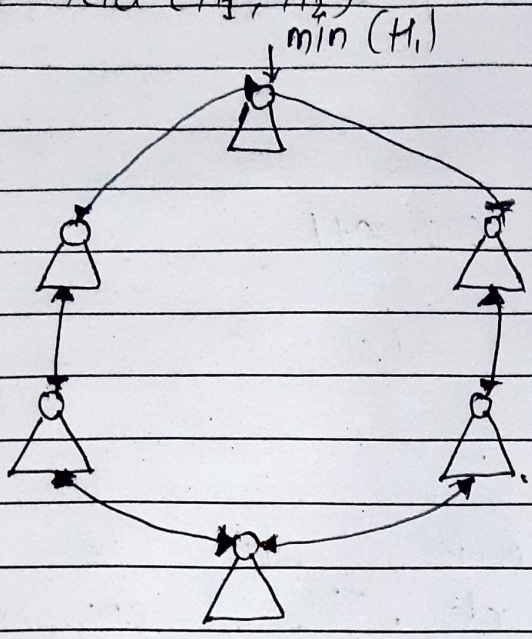


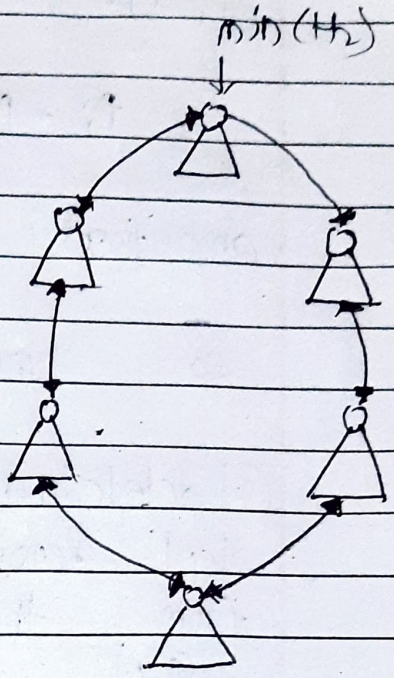
$P_i - P_{i-1} = 0$

amortised cost = actual cost = O(1)

Page No.
Date : / /

(3) meld (H_1, H_2):



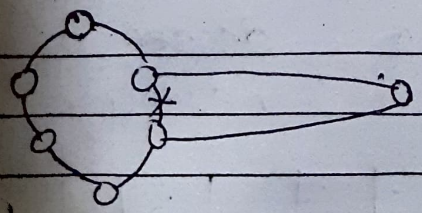


Merge root cycles of H_1 & H_2
Its cost is $O(1)$

$min = min(min(H_1), min(H_2))$
Adjusting $min(H)$ also takes $O(1)$.

(4) Insert (H, x):
step 1: create fibonacci heap H, which has a single node x

step 2: meld $(H, H_2) = H'$



$P_i(H') = t_{i-1}(H) + 1 + 2m_{i-1}(H)$

$\therefore P_i - P_{i-1} = 1$

amortised cost = actual cost + 1

so amortised cost = $O(1)$

(5) delete_min(H):

Step1: Remove the node with the minkey from the root cycle. This operation takes $O(1)$

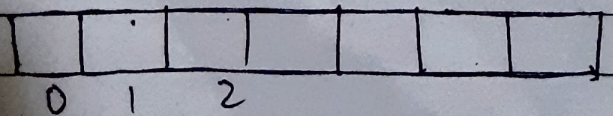
Step2: Merge the rootcycle with the cycle of children node.

Step 3: while two roots r_1, r_2 have same degree
do
link r_1, r_2
end while

It has complexity $O(t_{i-1}(H) + O(n))$

Step 4: Adjust the min key

D(n) returns a degree of node having highest degree in hierarchy of heap

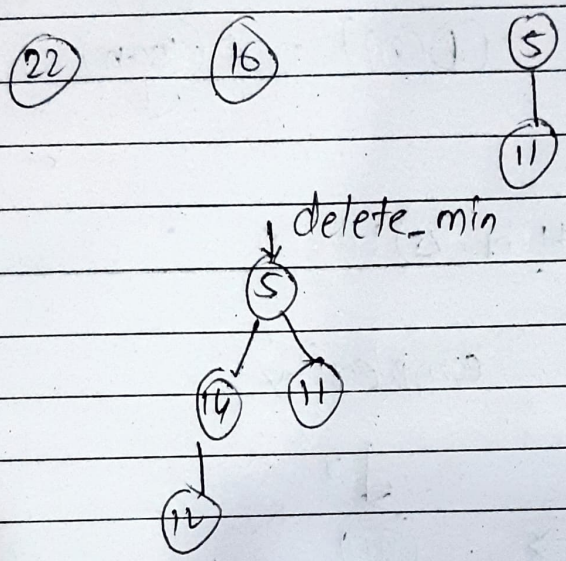


A[i] points to the node with degree(i) otherwise

A[i] = occupied with root x' link x' with x and try to place in A[i+1]

x' = x

Complexity depends on degree D(n)+1 & D(n) is O(log n)



Del Delete_min (H):

step 1: O(1)
step 2: O(1)
step 3: link $O(t_{i-1}(H) + D(n))$
Step 4: Adjust min $O(D(n))$

Total Actual cost := $O(t_{i-1}(H) + D(n))$

Potential Change:

$P_i \Rightarrow$ After potential = $D(n)+1 + 2m_{i-1}(H)$

$P_{i-1} \Rightarrow$ Before potential = $t_{i-1}(H) + 2m_{i-1}(H)$

Page No.
Date: / /

Change in the potential $(P_i - P_{i-1}) = D(n) + 1 - t_{i-1}(H)$

Amortised cost = actual cost + potential Difference

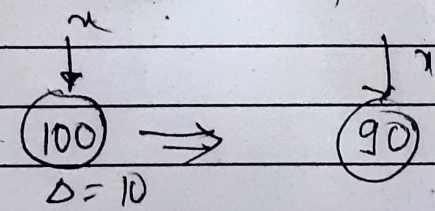
$= O(t_{i-1}(H) + D(n)) + D(n) + 1 - t_{i-1}(H)$

$= O(D(n)) + O(t_{i-1}(H)) - t_{i-1}(H)$

amortised cost = $O(D(n)) = O(\log n)$

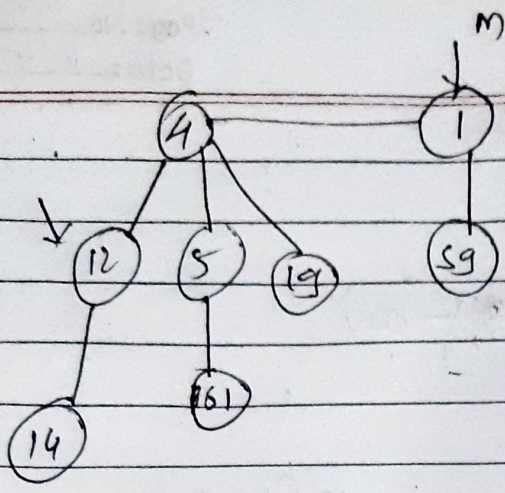
6] decrease_key (H, x, Δ):

It has O(1) complexity



Steps involved in decrease_key operation:

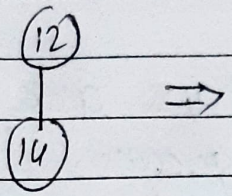
i) Unlink the rooted tree at x [marking is done]
ii) Decrease the key by Δ
iii) Add x to the root cycle
iv) Do cascading cut (marking bit is used)



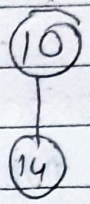
Page No.
Date : / /

$\Delta = 2$

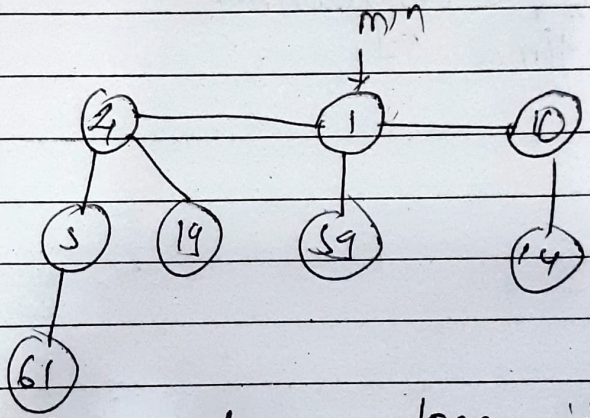
Step 1 =>

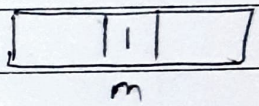


Step 2 =>



Step 3 =>





Whenever a node lose its first child it is marked

amortised cost = actual cost + potential diffe.

actual cost of the operation is guided by cascading cuts

we have made:

c: calls of cascading cuts

∴ actual cost = $O(c)$

Increase in no. of roots = c

Decrease in marking = c-2

change in potential = $(c+2) - (c-2) = 4$

amortised cost = $O(1)$

Page No.
Date: / /

7) Delete operation :-

delete (H, x) :-
1) decrease (H, x, -∞)
2) delete_min (H)

entire complexity is O(log n)

fibonacci heaps are used to implement the priority queues in Djkastra algorithm giving the algorithm a very efficient run time.