End Semester Examination (5th sem), 2021

→ Subject Name :→ Algorithms.

→ Subject Code :→ IT 3104.

→ Date of Examination :→ 08.12.2021

→ Name :→ Aniket Majhi.

→ Examination Roll No :→ 510819019.

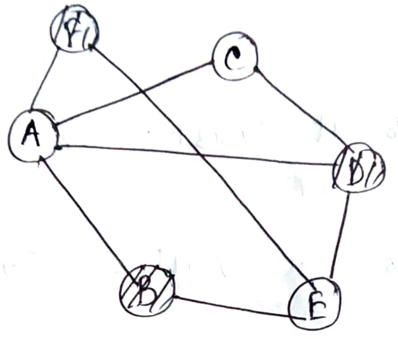
→ GSuite ID :→ 510819019.aniket@students.iiests.ac.in

→ Number of sheets uploaded :→ 14

5

(a) Independent Set :→ In a graph lets say $G = (V, E)$ we say a set of nodes $S \subseteq V$ is independent if no two edges in S are joined by an edge.

Example :→ lets consider a graph,



Here in this graph the independent sets are,

$\{B, D, F\}$, $\{A, E\}$, $\{C, E\}$ etc.

Proof of :→

We have to prove for a graph $V = (G, E)$ if S is an independent set if and only if its complement $V - S$ is a vertex cover.

As, the statement is like if and only if so we have to prove this by starting from both the side to reduce other side.

Firstly, lets start ~~frome the~~ by assuming S is an independent set.

lets Consider an arbitrary edge $e = (u, v)$.

Since S is independent, it cannot be the Case that both u and v are in S.

So one of them must be in the V-S.

It follows that,

Every edge has at least one end in V-S,

~~and~~ so,

V-S is a vertex cover.

⇒ By assuming S is a ~~set~~ independent set we proved V-S is a vertex cover. (Proved).

1

Secondly,

lets ~~start~~ assume V-S is a vertex cover.

Considering any two nodes u and v in S.

If they joined by edge e then neither of them ~~must mu~~ would lie in V-S.

It Contradicts with our assumption that V-S is a vertex cover.

It follows that no two nodes in S are joined by an edge, so S in an independent set.

We proved, S is an independent set by assuming V-S are a vertex cover (Proved)

(2)

By combining (1) and (2) We can say that,

For a graph, $G = (V, E)$,

S is an independent set if and only if V-S is a vertex cover (Proved).

(b.) Proof of Vertex cover $\leq_P$ Independent set :→

If we have a blackbox to solve independent set, then we can decide whether G has a vertex cover of size at most k by asking the black box whether G has a independent set of size at least n-k.

(4)

(a) Branching factor of B-Tree :→

The branching factor in a B-Tree is defined as

- The maximum number of children a node within the B-Tree can have.

  So if a node in a B-Tree having large number childen has the large branching factor.

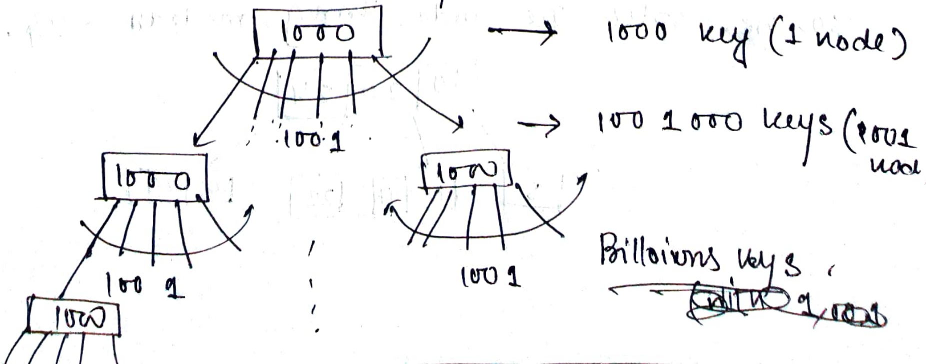
☒ Large branching factor reduced the height of B-Tree :-

Yes, a large branching factor means that the tree height may be considerably less than that of other trees with comparativetively smaller branching factor.

The effect of the larger branching factor is that no of disk accesses required to find a key.

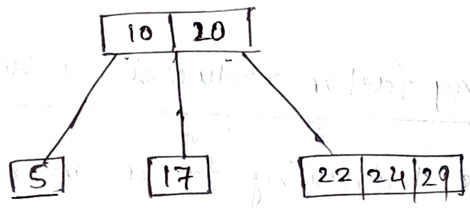
So, if the branching is large then the height of the tree is less the the number of disk accesses will also drastically reduced.

Example, Consider a BTree of height 2 shown below over a billions keys.



☑ The root of the B-Tree is always to be kept in main memory, so that a disk-read on the root is never required, only a disk write on the root is required when the root node gets changed. Keeping the root permanently in main memory also reduces number of disk accesses required to find a key within the tree.

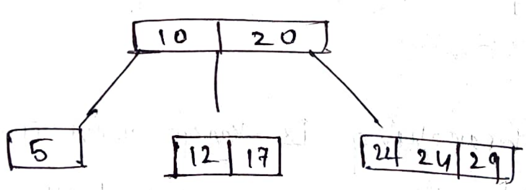
(b)



Inserting 12 :→

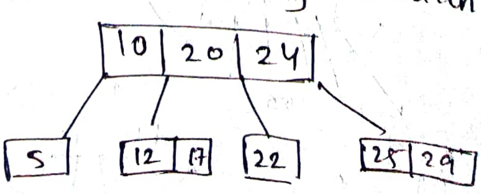
As, 12 > 10 and 12 < 20 so it should be inserted with 17.

~~and also since there~~



Inserting 25 :→

As 25 > 20, it goes to the right of 20. but that node is already full having 2·2−1 = 3 children (t=2) so, we split the node using median = 24.



③ (a) Black height :→ The black height of the node ~~x~~ is the number of black nodes on any path from ~~root~~ but not including to a node x down to leaf.

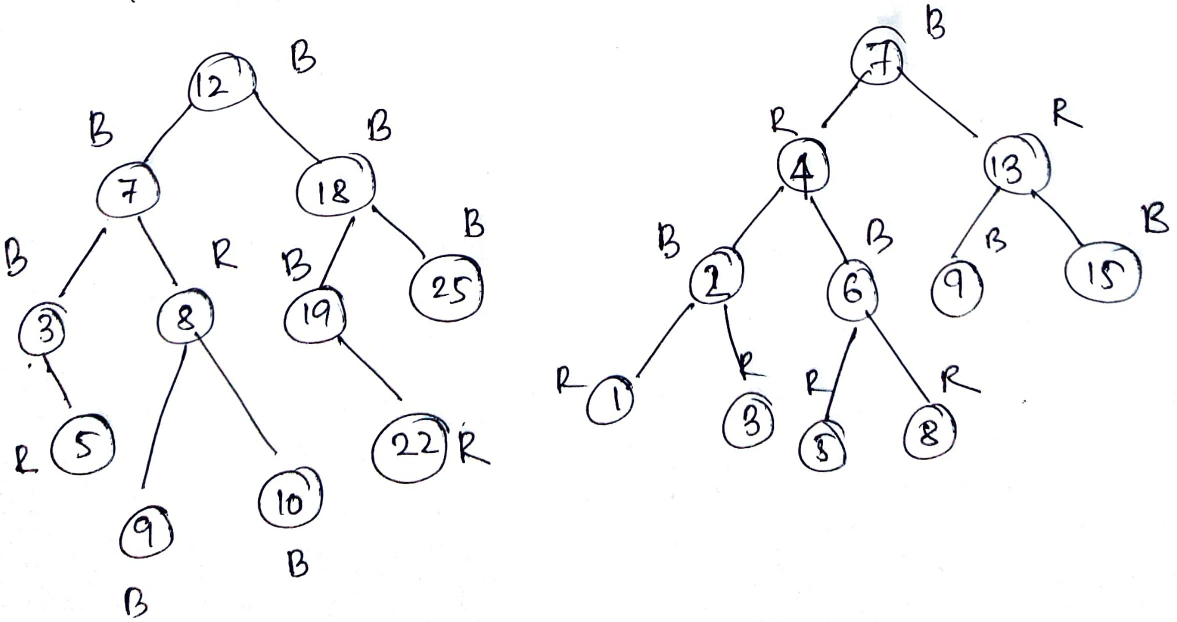
It is denoted by bh(x) for a node x.

~~bh(x) = black~~

Black height of RB Tree :→ The black height of a red black tree is the number of black nodes in any path from the root to the leaf nodes or the black height of any leaf nodes.

☐ Same Height two red black trees :→

let's take two red - black trees H1 and H2 of same height of height 3



(3)

(b) Maximum height of Red-Black Tree :→

Basically a Red-Black Tree having n internal nodes has ~~at most~~ height at most $2\log(n+1)$.

Proof :→

We will first show that the subtree rooted at any node x contains at least $2^{bh(x)}-1$ internal nodes.

here,

$bh(x)$ = The number of black nodes in any path from root but not ~~not~~ including a node x down to leaf.

We ~~will~~ prove this claim by induction on the height of x.

(i) If heigh of $x = 0$,

then x must be leaf and the subtree rooted at x indeed contains at least $2^{bh(x)}-1$

$$= 2^0 - 1$$
$$= 1 - 1$$
$$= 0$$

internal nodes.

(ii) Inductive step :→ Consider a node x that has positive height and it's ~~and~~ is an internal node with two childern.

Each child has a black-hight of either $bh(x)$ or $bh(x)-1$ depending on whether its color is red or black respectively.

Since, the height of a child of x is less than the height of x.

We can use the inductive hypothesis,

By using that,

We can conclude each child has at least

$2^{bh(x)}-1$ internal nodes.

Thus,

We can say that the subtree rooted at x contains at least, ~~$2^{bh(x)}$ + $2^{bh(x)}$ + 1~~

$$2^{bh(x)-1} - 1 + 2^{bh(x)-1} \cancel{-1} \cancel{+1}$$

$$= 2^{bh(x)} - 1$$

internals nodes.

(Proved).

To complete the proof, ~~of~~

let h be the height of the tree.

According to the property of RB Tree, we know at least half the nodes on any simple path from root of leaf, ~~not including~~ excluding the root must be black.

Thus,

$$n \geq 2^{h/2} - 1 \qquad h/2 = \text{the black height of the root.}$$

$$\Rightarrow n+1 \geq 2^{h/2}$$

$$\Rightarrow \frac{h}{2} \leq \log(n+1)$$

$$\Rightarrow h \leq 2\log(n+1) \quad [\text{Proved}]$$

(8)

(a)

Randomized Quick sort Algo :→

In a randomized quicksort algo, we ~~use a pivot~~ choose the pivot randomly. then sort ~~the~~ the array according to that pivot.

In a normal quicksort algorithm we first partition the array in place ~~&~~ such that all elements to the left are smaller that the chosen pivot and all elements in the right are greater that that and we recursively call the other sub problems to solve. ~~~~

Time Complexity :→

n = ~~~~ Array size.

(i) Best case :→ $\Theta(n\log n)$

(ii) Average case :→ $O(n\log n)$

(iii) Worst case :→ $O(n^2)$.

☒ Comparing Heap sort, Quicksort, Mergesort :→

(i) Quicksort is fastest for random data as this algorithm tends to partition the data set into two similar sized pieces. This means that in terms of locality of reference once we have a piece ~~of~~ that fits in memory, locality is exploited until this piece is fully sorted.

(ii)

ii) Heapsort is not really local as the heap accessed all over when the items find their places in the heap. ~~Others flips~~ and others are rearranged in the heap.

(iii) Merge sort recursively breaks down a list into several sublists until each sublist consists of a single elements and we merge them in recursive manner.

☐ <u>Binary heap</u> :→

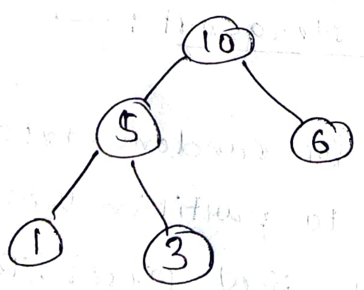
A binary heap is a binary tree with the following properties —

(i) It is a complete binary tree.

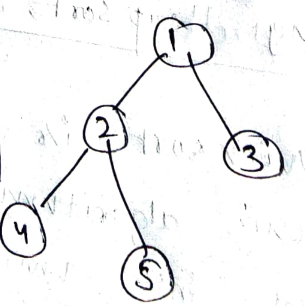
(ii) It can be max/min heap.

<u>Example</u>,

<u>Max heap</u>,



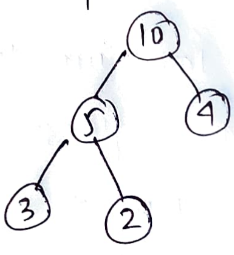
<u>min heap</u>,



☐ **Complete binary tree :→**

A complete binary tree is a binary tree where, all levels are completely filled except possibility the last level and the last level has all the keys as left as possible.

Ex :-



(6)

(a) Y polynomialy reducable to X :-

If an arbitrary instances of problem Y can be solved by using a Polynomial number of standard steps, Plus a polynomial number of calls to a black box that solves the problem X then Y is polymial time reducible to X.

☒ let's Consider,

$$Y \leq_P X$$

X Can be solved in Polynomial time then Y also can be solved in Polynomial time. Then our black box for X is actually not so valuable; we ~~are~~ can replace it with a Polynomial time algorithm X.

Therefore, Y can be solved in Polynomial ~~time~~ number of steps through a number of calls to the black box X. So Y is also an Polynomial time reducible to X.

☒

(b)

Independent set $\leq_P$ set Packing Problem.

~~Proof~~

Independent set :→ For a graph $G = (V, E)$, we say a set of nodes $S \subseteq V$ is independent if no two edges are connected by an edje.

set packing Problem :→ Given a set U of n elements, a collection $S_1, S_2 \dots, S_n$ of subsets of U, and a number k.

does there exist a collection of at least k of the sets with the property that no two of them intersect.