Birkbeck, University of London

Fundamentals of Computing Coursework 2

Baran Buluttekin

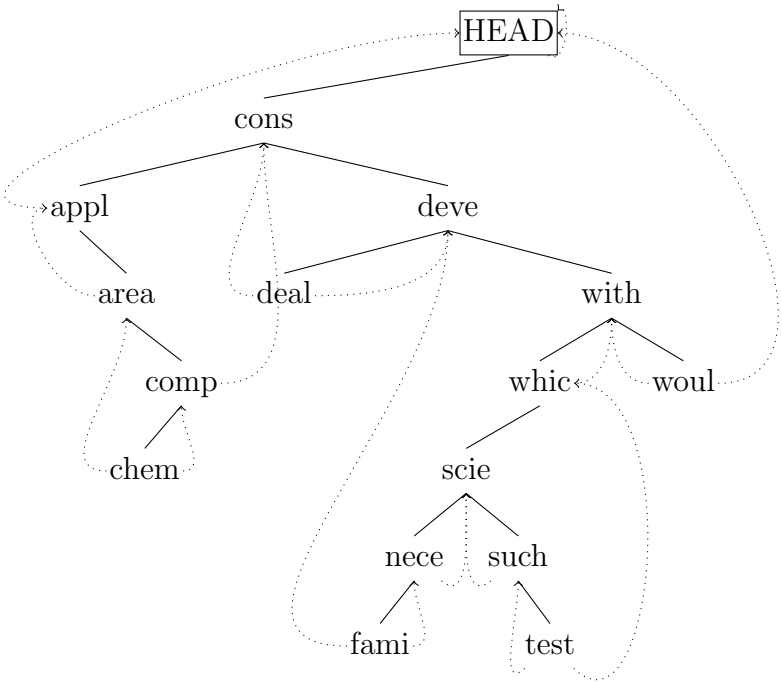
13153116

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.

Answers

- Baran Buluttekin
- Summation of values = 171
- Line number = 72
- X = 1 its 5th word
- words = consisted developing application area with which computer scientist would necessarily familiar such chemical testing dealing
- four letter version = cons, deve, appl, area, with, whic, comp, scie, woul, nece, fami, such, chem, test, deal

1. Threaded binary tree:



2. *Post-order* traversed: chem, comp, area, appl, deal, fami, nece, test, such, scie, whic, woul, with, deve, cons

3. Traversed *pre-order* with algorithm from p.16: cons, appl, area, comp, chem, deve, deal, with, whic, scie, nece, fami, such, test, woul

   For the stack below left most item represent the first item in and right most item is the last item get in to the stack. Words crossed out represent item that was in the stack but popped out.

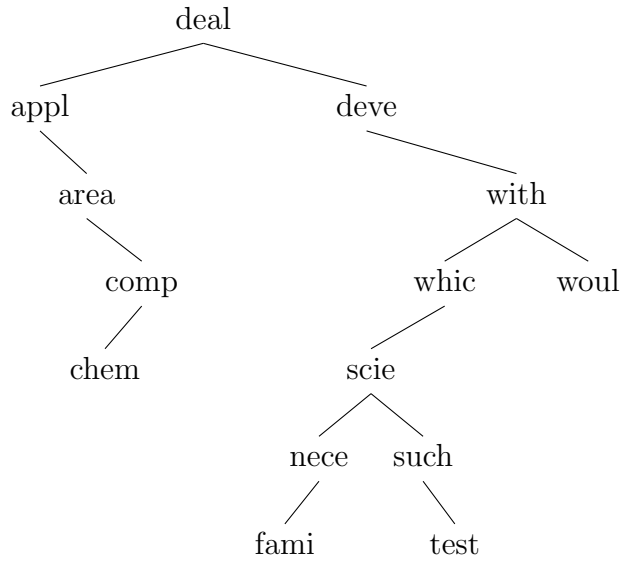
My 3^{th} node is appl,
Nodes visited: cons, appl
Stack: cons

My 6^{th} node is whic,
Nodes visited: cons, appl, area, comp, chem, deve, deal, with, whic
Stack: ~~cons~~, ~~appl~~, ~~area~~, ~~comp~~, ~~chem~~, ~~deve~~, ~~deal~~, with

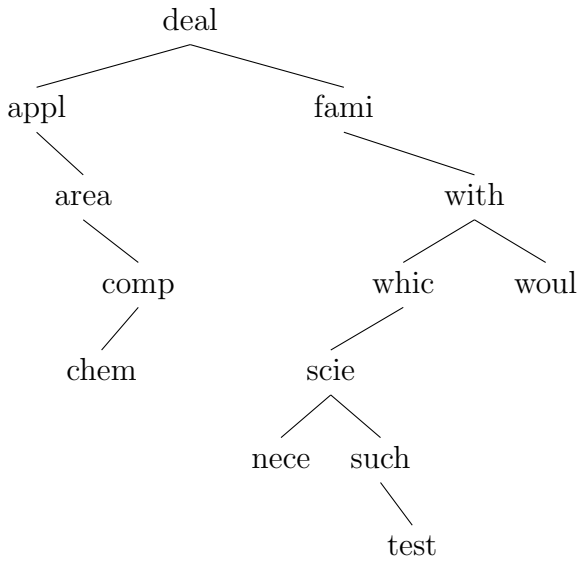
My 9^{th} node is woul,
Nodes visited: cons, appl, area, comp, chem, deve, deal, with, whic, scie, nece, fami, such, test, woul
Stack: ~~comp, appl, area, comp, chem, deve, deal, with, whic, scie, nece, fami, such, test~~

My 12^{th} node is such,
Nodes visited: cons, appl, area, comp, chem, deve, deal, with, whic, scie, nece, fami, such
Stack: ~~cons~~, ~~appl~~, ~~area~~, ~~comp~~. ~~chem~~, ~~deve~~, ~~deal~~, with, whic, ~~scie~~, ~~nece~~, ~~fami~~

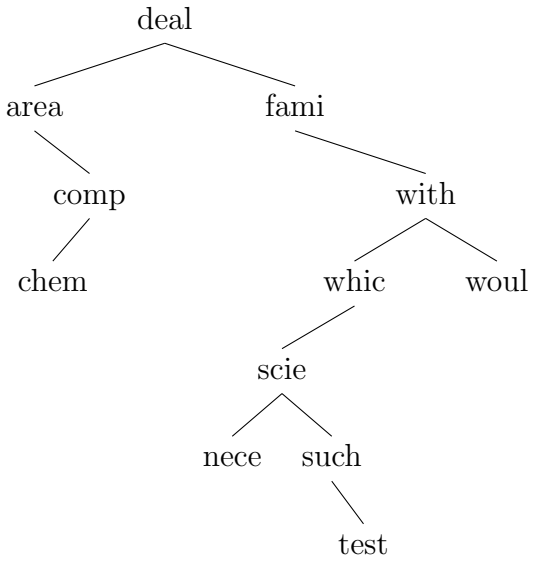
4. 1^{st} item is cons, when removed:



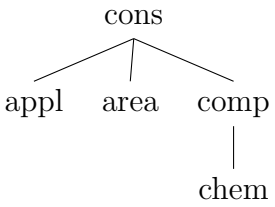
2^{nd} item is deve, when removed:



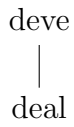
3^{rd} item is appl, when removed:



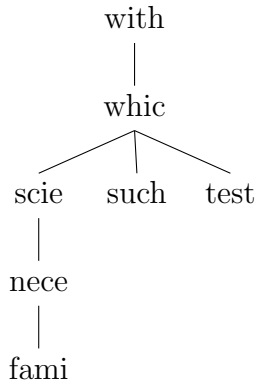
5. We will obtain 4 fallowing trees.
   $1^{st} tree$:



$2^{nd} tree$:



$3^{rd} tree$:



$4^{th} tree$:

Node: woul

6. Algorithm:

```
void FindLinkedNode(DataValue k, Treenode ROOT)
{
    P ← ROOT;
    // Find the node N
    while (P↑INFO ≠ k)
    {
        P ← P↑LLINK if (P↑INFO > k) else P↑RLINK;
    }
    if (P = nil)
        repot_not_found();
    else
    { // Find the head node by moving along nodes
      head node will have a pinter Q
        Q ← P↑SUCC;
        while (Q↑INFO ≠ 9999)
        {
            Q ← Q↑SUCC;
        }
        // Once found delete the head note to
           replace to new position
        if (Q↑SUCC ≠ P)
        {
            Back ← Q↑PRED;
            Front ← Q↑SUCC;
            Back↑SUCC ← Front;
            Front↑PRED ← Back;
            // Insert head node after node N
            Prev ← P↑PRED;
            Prev↑SUCC ← Q;
            Q↑PRED ← Prev;
            Q↑SUCC ← P;
            P↑PRED ← Q;
        }
    }
}
```

7. B-tree examples

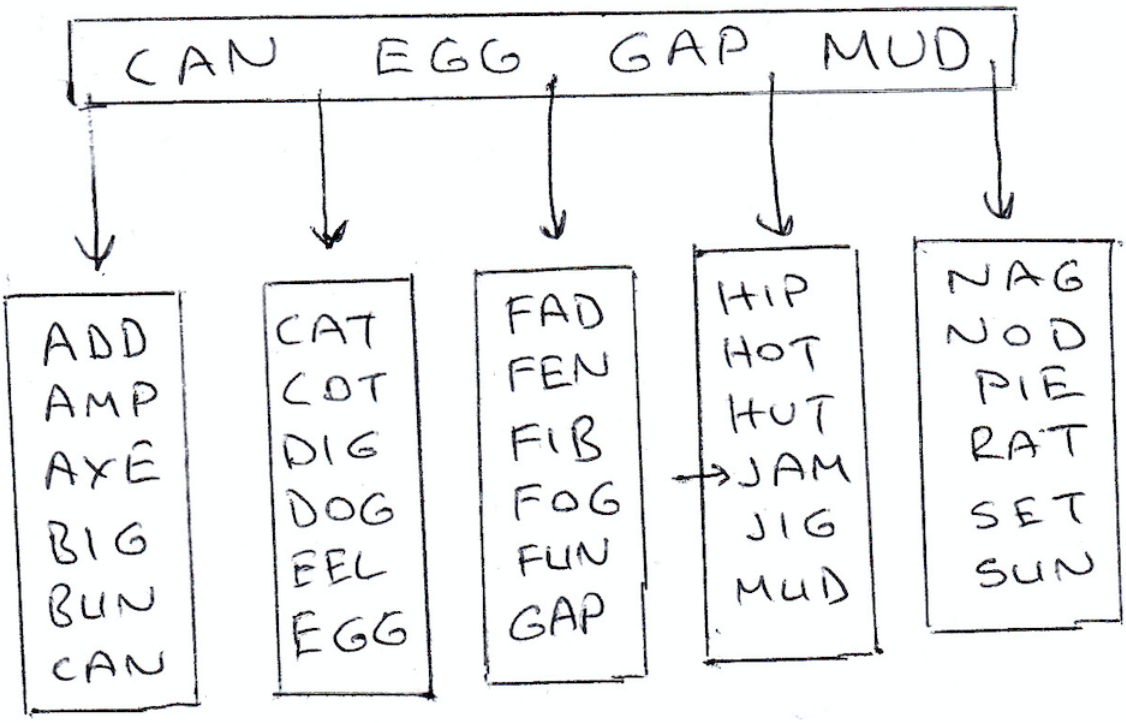
Strategy for data-pages:

(a) Odd number items will always be on the left-hand page.

(b) When merging a page, I will always merge with the left-hand page. Unless the page doesn't have a left page or merging with left-hand page cause overflowing data-page.

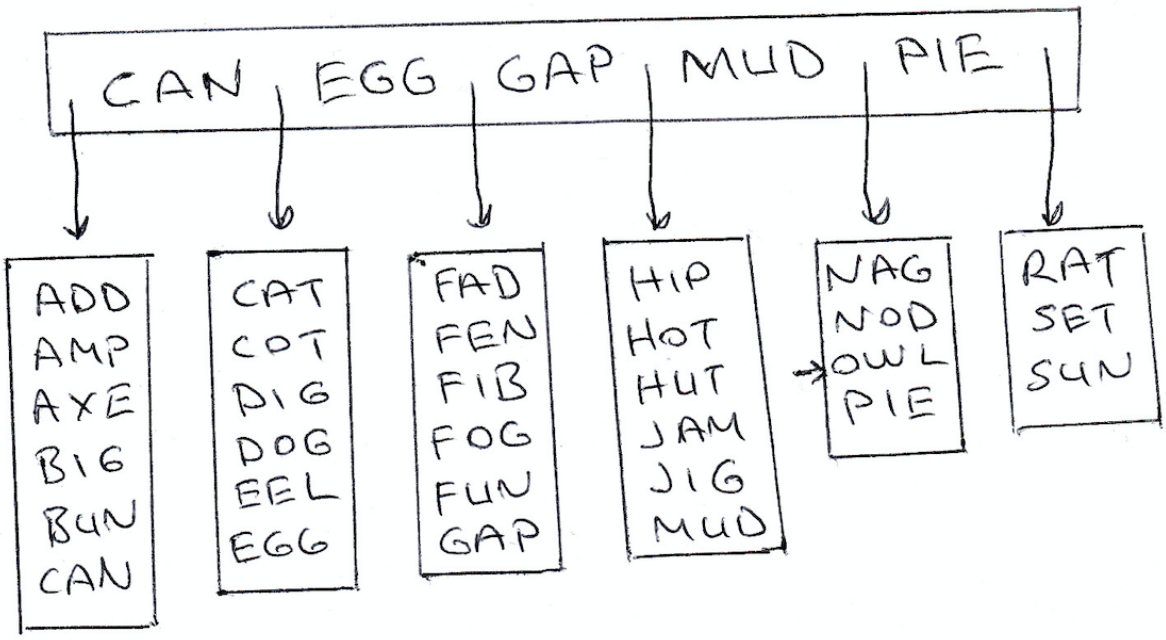
m = 6 r = 6
Inserting JAM:

This item will be inserted to data page of [HIP, $\cdots$, MUD]. This data page have one more record space available, record can be added without any split.



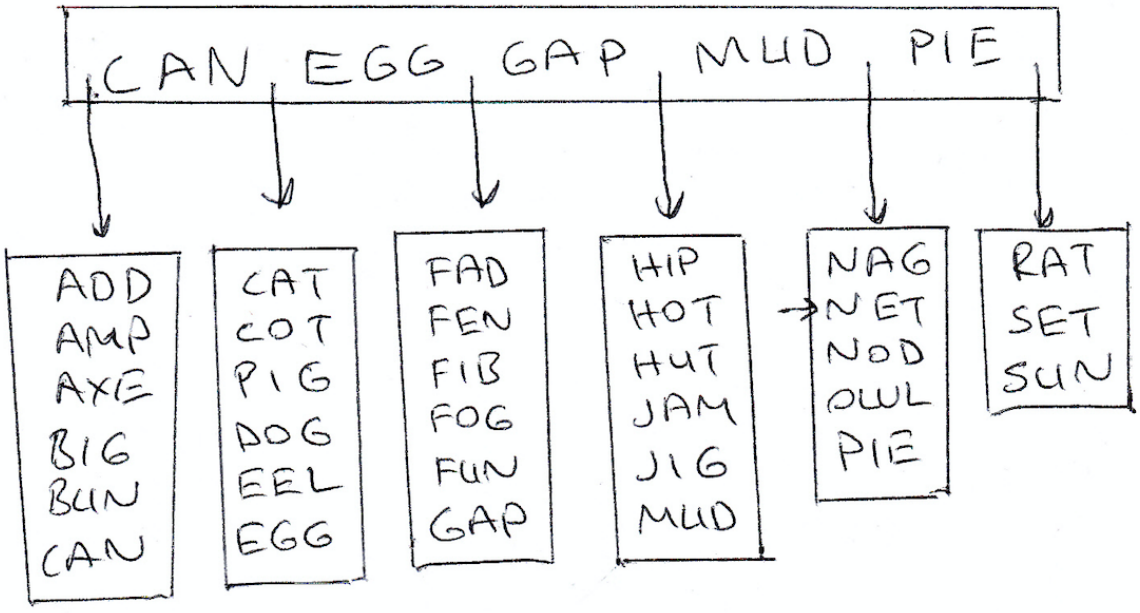
Inserting OWL:

This record should be inserted to data-page of [NAG, $\cdots$, SUN]. This page have already 6 item, adding this record will split the page. Final result will have max number of pointer page.



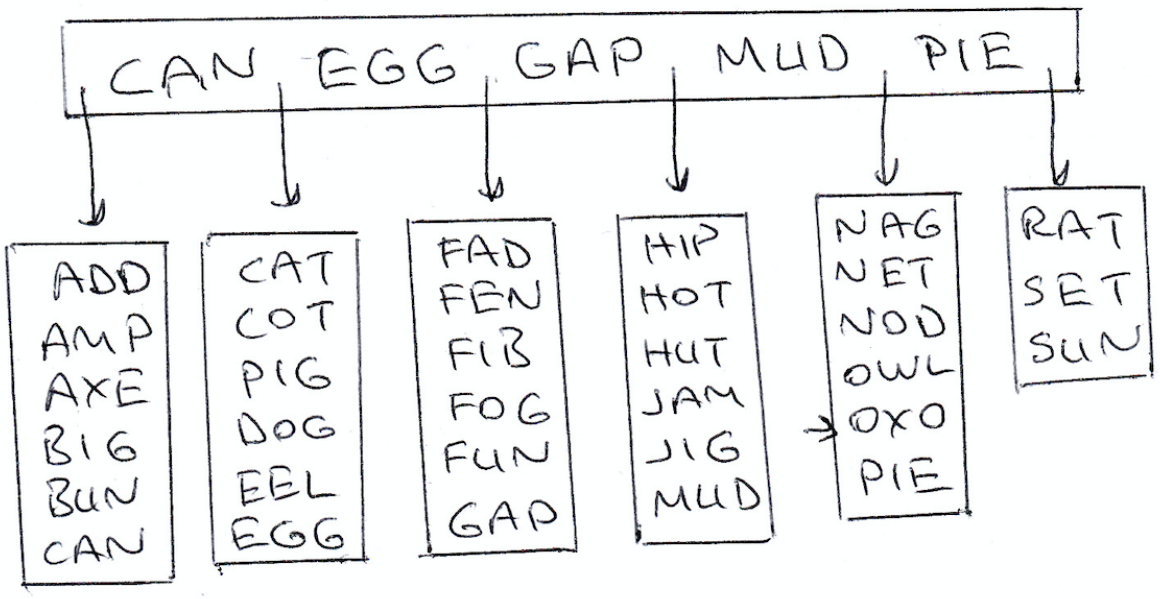
Inserting NET:

Does not cause split just inserted.



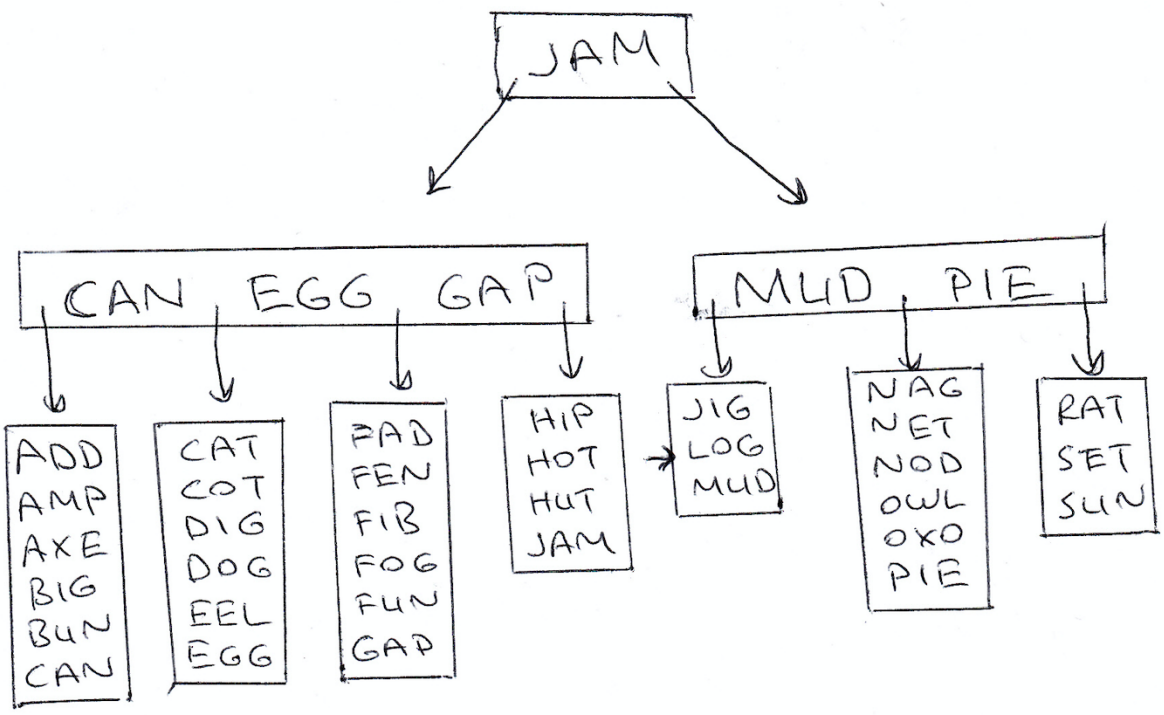
Inserting OXO:

Does not cause any split but the data-page it goes into have maximum number of data.



Inserting LOG:

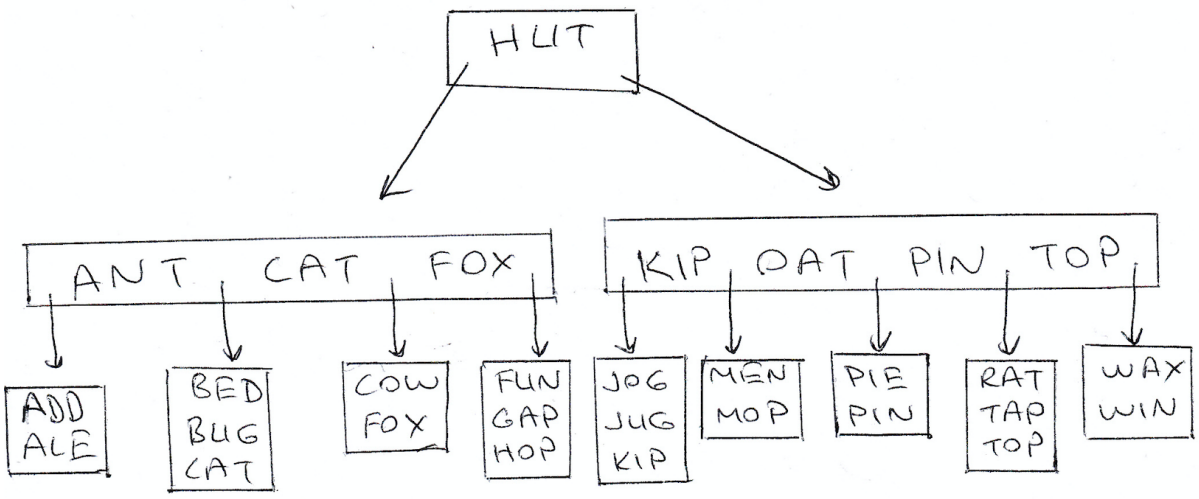
This will go to data-page [HIP, $\cdots$, MUD]. Result is a split data-page. Because the maximum number of pointer pages reached in this level the key we obtain will go to layer up.



8. Deletion

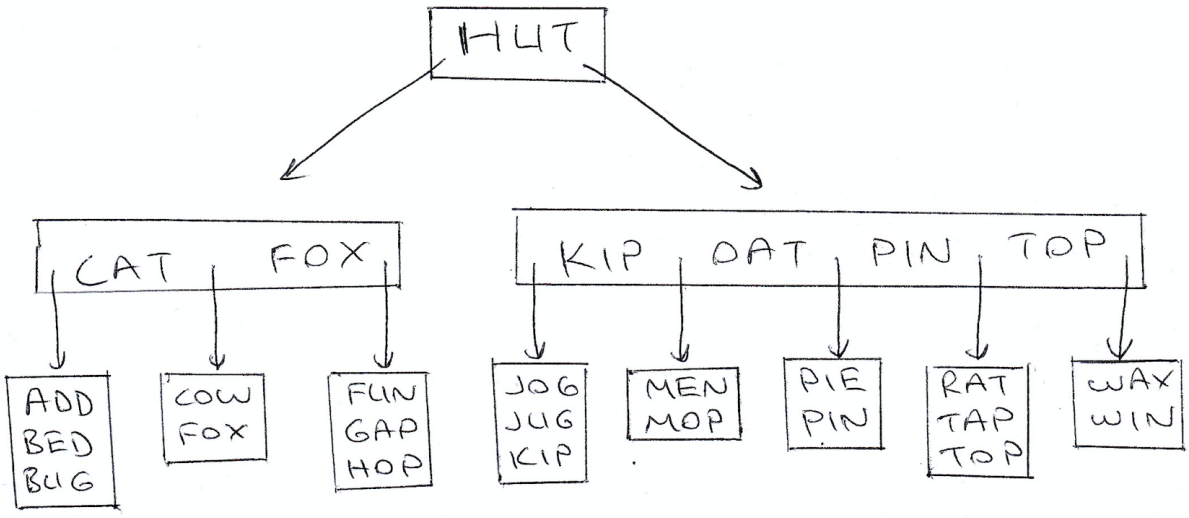
   m = 5 and r = 4
   Deleting RIM:

   When RIM is deleted data-page it was occupying will have less than min number record and needs to be merged with another page. Because there is no left page in this pointer page I merged it with right data-page. and key SAG not in use can be removed. Removing that key will leave that pointer page less than minimum number of pointer pages and we ca merge it with left pointer page to avoid it. Top level key PIN no longer required and we can bring it to lover level to pointer-page.



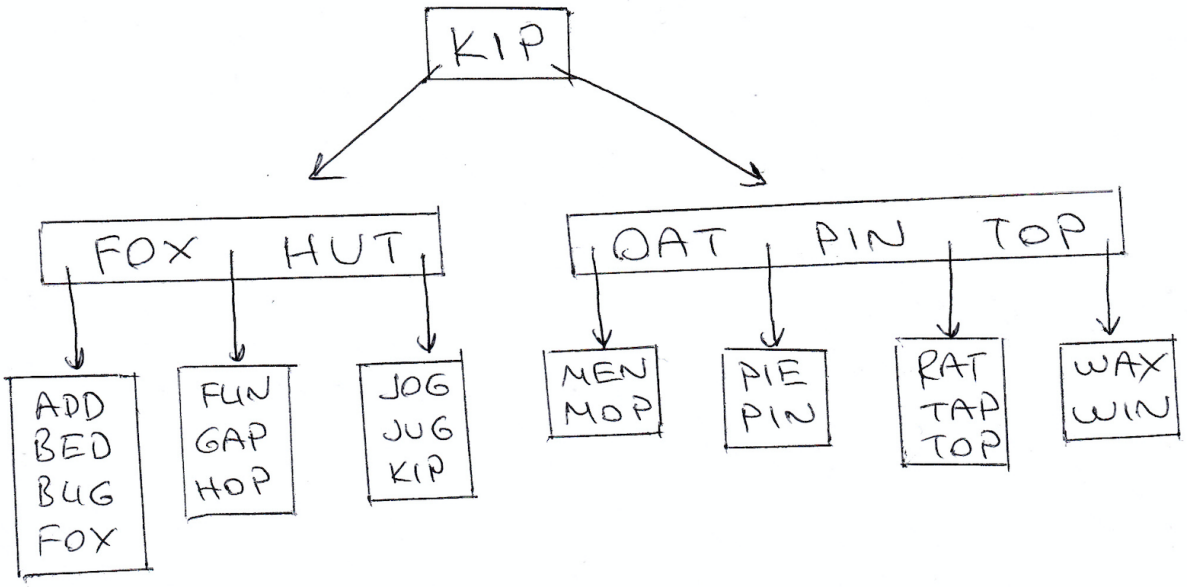
Deleting ALE:

When we delete ALE data-page it was in will have to merge. In this case there is no left page thats why I merged it with right page. It's pointer page no longer in use can deleted.



Deleting COW:

Deleting COW also require merging. When merged CAT key is not needed and can be deleted. After the key deleted this pointer page does not have enough data-page and have to be merged with next pointer-page. Upper level HUT is used in merging but the end result have more than maximum number of pointer-pages. I split the pointer pages by taking KIP key to upper level.



9. Delete RAT:

   m = 5 and r = 4
   When the RAT is deleted data-page have to merge with left data-page and key PIN can be deleted. Result then have insufficient number of pointer pages. Have to be merged with left pointer page by bringing the top level key FOX down.