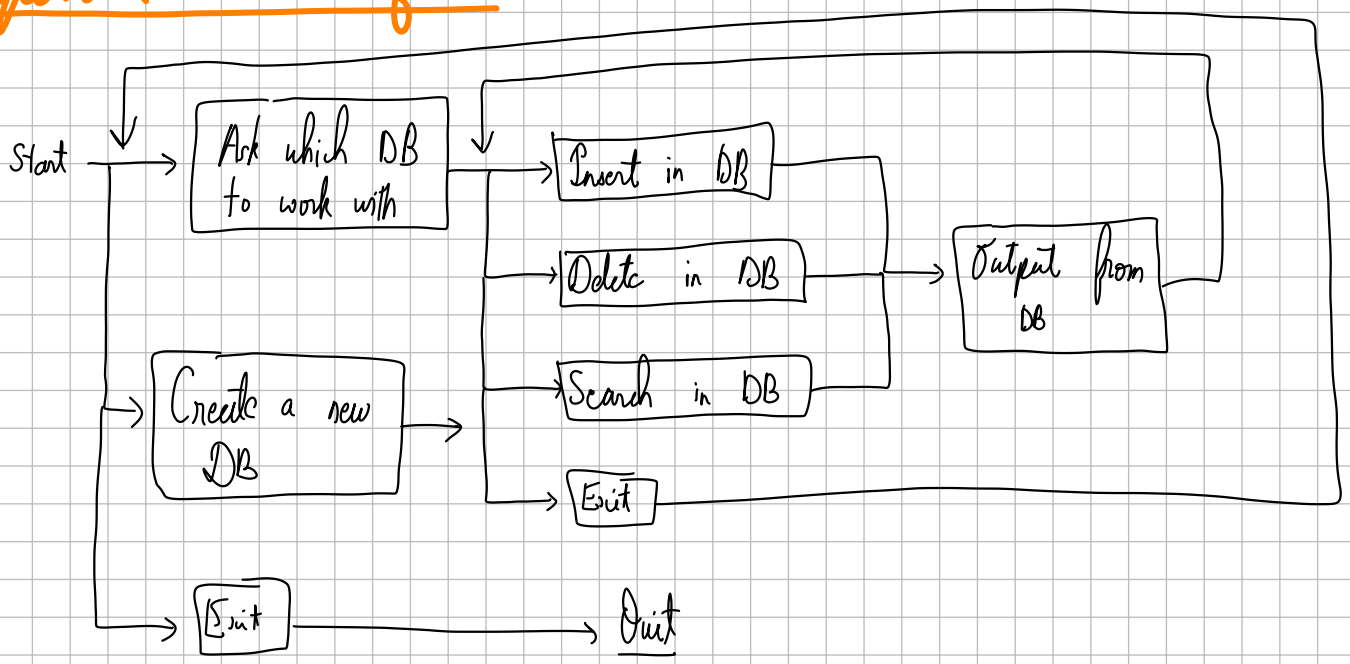
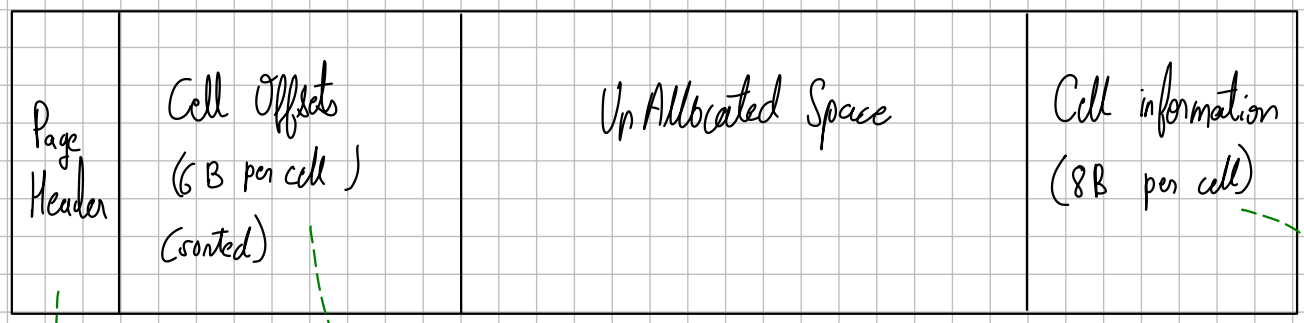
# Required User Interface




# Node Structure → needs to fit in 1 page of the device

grep -ir pagesize /proc/self/smaps

→ 4kB in my device




1 kB

$2^{10}$ Bytes → Max Offset ⇒ 10 bits max required for each offset

Information it needs to contain

- Unallocated Space Table
- 4B Unique Number at starting to identify it as a page
- Page ID (needs to there for ref.)

2B for offset
4B for key

Child Pointer (4B) | Data (4B)

The total no. of nodes in my B_tree will never cross $2^{32}$

If I keep MAX-DEGREE of B-Tree to be X

→ Cell Offset Area ⇒ 6X Bytes
→ Cell Information Area ⇒ 8 X Bytes

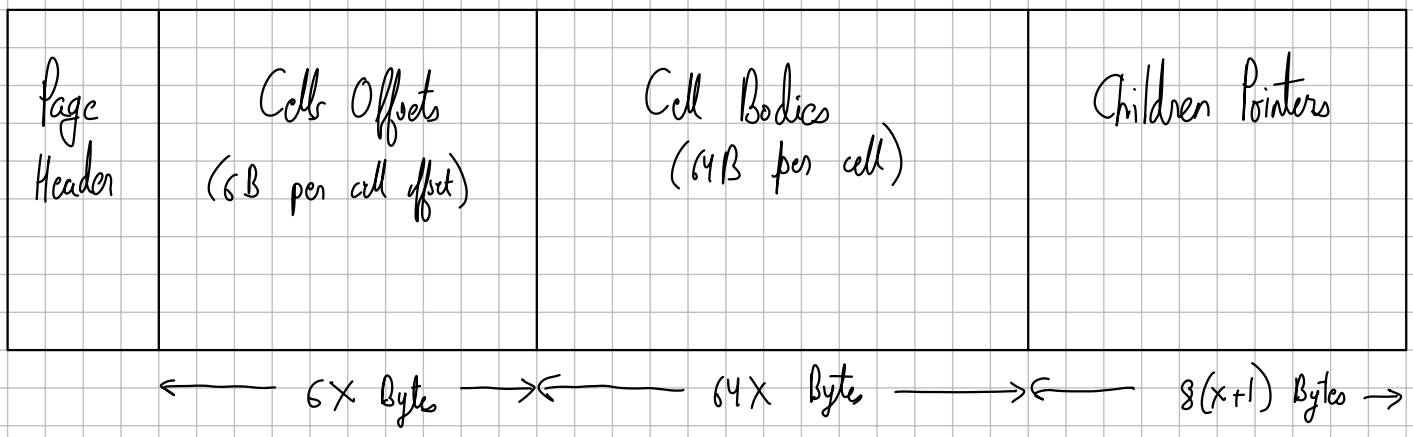
We need to make a tradeoff
↙ less X means less fragmentation problems
but also means less utilisation of page.

| X | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Cell Offset Area (in Bytes) | 6 | 12 | 24 | 48 | 96 | 192 | 384 | 768 | 1536 | 3072 | 6144 |
| Cell Info Area (in Bytes) | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
| | | | | | | | | | | | |
| Total Area (in Bytes) | 14 | 28 | 56 | 112 | 224 | 448 | 896 | 1792 | 3584 | 7168 | 14336 |
| Total Area (in kB) | 0.0137 | 0.0273 | 0.0547 | 0.1094 | 0.2188 | 0.4375 | 0.875 | 1.75 | 3.5 | 7 | 14 |

↳ More X means less disk accesses

↳ What if I can just not encounter the problem of Fragmentation? Afterall, this problem occurs only when we dont know the sizes that the entries might take...

# Revised Node Structure



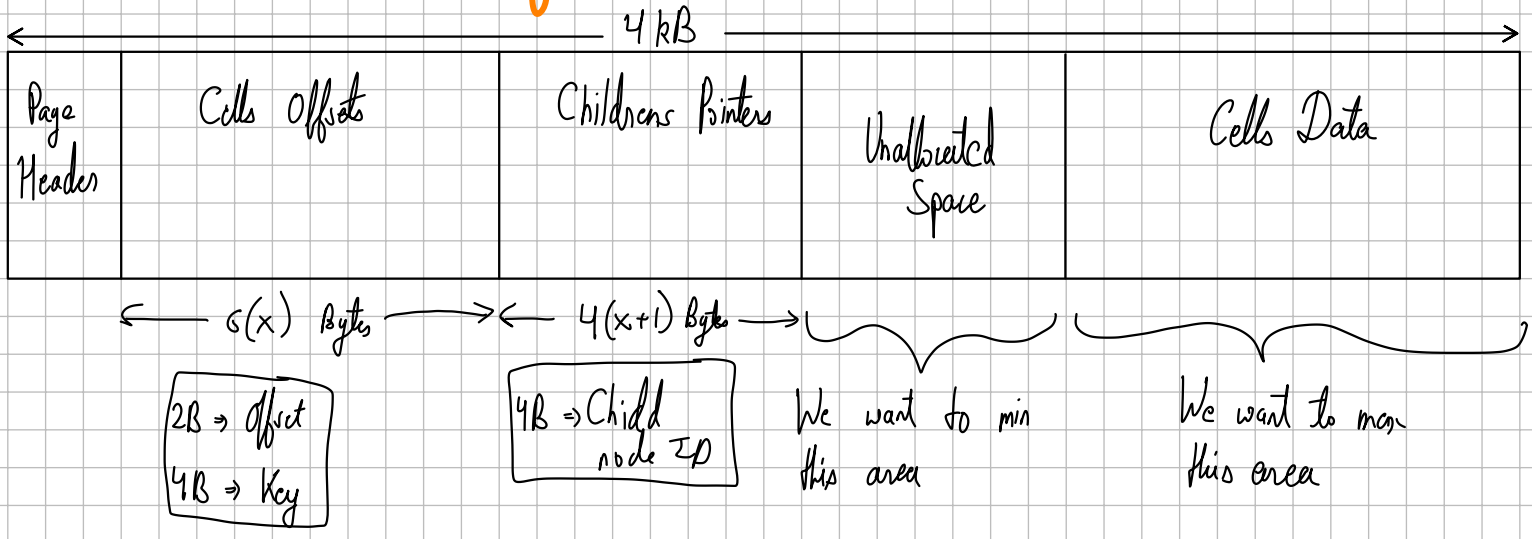

2B → Offset at which cell body is there
4B → key information

64B data for each cell

This was all for Fixed Size Data

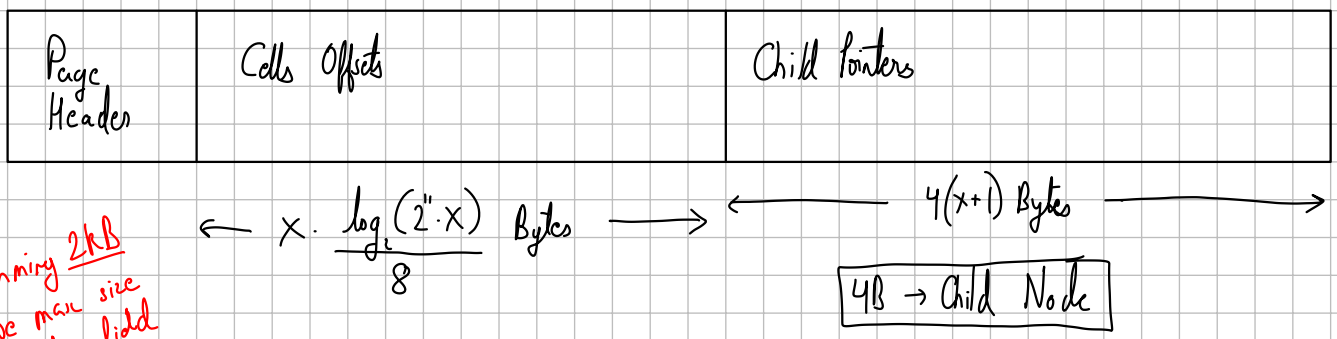
Lets move to Variable Size Data

# Node Structure for Variable Sized Data




| X | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Cell Offset Area (in Bytes) | 6 | 12 | 24 | 48 | 96 | 192 | 384 | 768 | 1536 | 3072 | 6144 |
| Cell Info Area (in Bytes) | 8 | 12 | 20 | 36 | 68 | 132 | 260 | 516 | 1028 | 2052 | 4100 |
| | | | | | | | | | | | |
| Total Area (in Bytes) | 14 | 24 | 44 | 84 | 164 | 324 | 644 | 1284 | 2564 | 5124 | 10244 |
| Total Area (in kB) | 0.0137 | 0.0234 | 0.043 | 0.082 | 0.1602 | 0.3164 | 0.6289 | 1.2539 | 2.5039 | 5.0039 | 10.004 |

↳ If we increase the fanout then the disk seeks would also decrease, thus we put data in different page entirely. The additional seeks to fetch data might get compensated.




Assuming 2kB to be max size of data field

Q How do I store the pointer to the data in the cell offset?

↳ If I store offset in the data page... Then what about when data pages has overload pages?

↳ Ok, so maybe set a limit for how big a data field can be... Maybe 2kB ... In test this is about 315-410 words ... About a page worth of text.

↳ Worst case in all 'x' cells have 2kB of data. How much max offset will I need? ⇒ $X \cdot 2k = X \cdot 2 \cdot 2^{10} = 2^{11} \cdot X$ ⇒ To store this big offset I will need atleast $\log_2(2^{11} \cdot X)$ bits ⇒ $\log_2(2^{11} \cdot X) / 8$ Bytes, cell

| X | 256 | 291 | 326 | 361 | 396 | 431 | 466 | 501 | 536 | 571 | 606 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Each Cell Offset Size | 2.375 | 2.3981094179 | 2.4185910193 | 2.4369818784 | 2.4536695775 | 2.4689430074 | 2.4830232681 | 2.496083349 | 2.508261149 | 2.519668367 | 2.530396748 |
| Cell Offset Area | 608 | 697.8498406 | 788.46067228 | 879.75045809 | 971.65315269 | 1064.1144362 | 1157.0888429 | 1250.537758 | 1344.427976 | 1438.730638 | 1533.420429 |
| Child Pointers Area | 1028 | 1168 | 1308 | 1448 | 1588 | 1728 | 1868 | 2008 | 2148 | 2288 | 2428 |
| | | | | | | | | | | | |
| Total Area (Bytes) | 1636 | 1865.8498406 | 2096.4606723 | 2327.7504581 | 2559.6531527 | 2792.1144362 | 3025.0888429 | 3258.537758 | 3492.427976 | 3726.730638 | 3961.420429 |
| Total Area (kB) | 1.59765625 | 1.822118985 | 2.0473248753 | 2.2731938067 | 2.4996612819 | 2.7266742541 | 2.9541883232 | 3.182165779 | 3.410574195 | 3.639385388 | 3.868574638 |
| | | | | | | | | | | | |
| Worst Case Data Size (kB) | 512 | 582 | 652 | 722 | 792 | 862 | 932 | 1002 | 1072 | 1142 | 1212 |

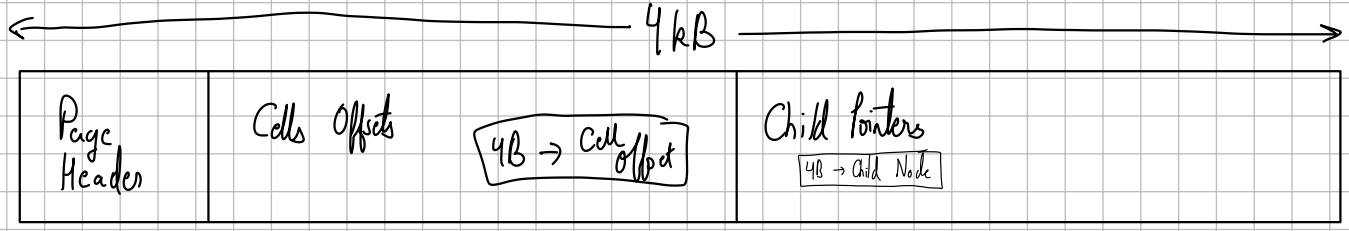
⇐ According to above decided

| X | 256 | 291 | 326 | 361 | 396 | 431 | 466 | 501 | 536 | 571 | 606 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Each Cell Offset Size | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Cell Offset Area | 768 | 873 | 978 | 1083 | 1188 | 1293 | 1398 | 1503 | 1608 | 1713 | 1818 |
| Child Pointers Area | 1028 | 1168 | 1308 | 1448 | 1588 | 1728 | 1868 | 2008 | 2148 | 2288 | 2428 |
| | | | | | | | | | | | |
| Total Area (Bytes) | 1796 | 2041 | 2286 | 2531 | 2776 | 3021 | 3266 | 3511 | 3756 | 4001 | 4246 |
| Total Area (kB) | 1.75390625 | 1.9931640625 | 2.232421875 | 2.4716796875 | 2.7109375 | 2.9501953125 | 3.189453125 | 3.428710938 | 3.66796875 | 3.907226563 | 4.146484375 |
| | | | | | | | | | | | |
| Worst Case Data Size (kB) | 512 | 582 | 652 | 722 | 792 | 862 | 932 | 1002 | 1072 | 1142 | 1212 |

⇐ Rounded off

↳ Ok, so all this seems since I was already thinking of putting 4 bytes in

↳ So 4Bytes per cell offset. This will allow for the data field of a cell to be a max of Doesn't matter. We will make a way for larger data to fit... Somehow...



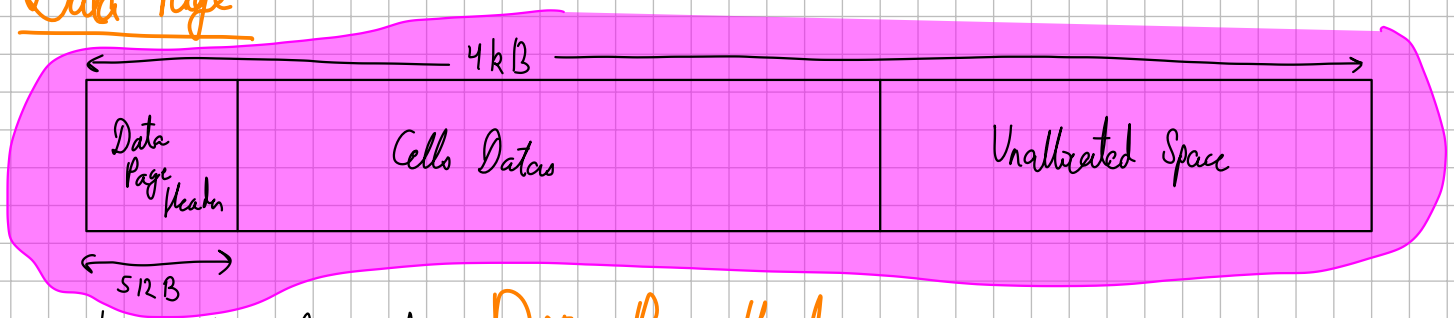


~~X = Max_Degree = 466~~ → Made wrong assumption of Cells offset size

↳ This makes size of Page Header to be 364 Bytes

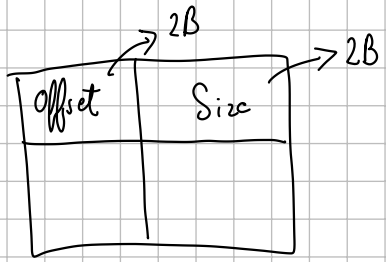
↳ Also, our Data field max size limit goes upto 8MB!!

## Data Page



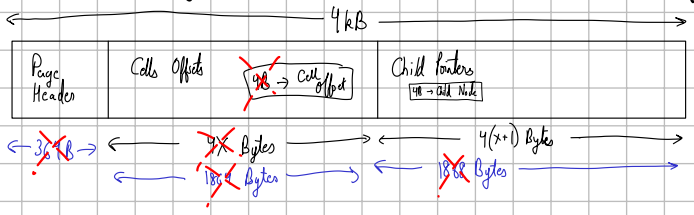

↳ Needs to hold info: **DATA Page Header**

- ↳ Identification_num (4B)
- ↳ Page_Type (1B) ⇒ 45
- ↳ Data currently held (2B) → Data size currently held only in this data page.
- ↳ Next data page id (4B)
- ↳ parent_node_page_id (4B)
- ↳ Space_Table_Size (2B) → Size of this table →




- ↳ Unallocated Space Table →
  - ↳ Atmost 123 entries

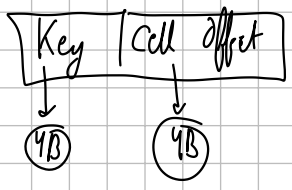
↳ Data page is decided. Now I need to finalize the node page.

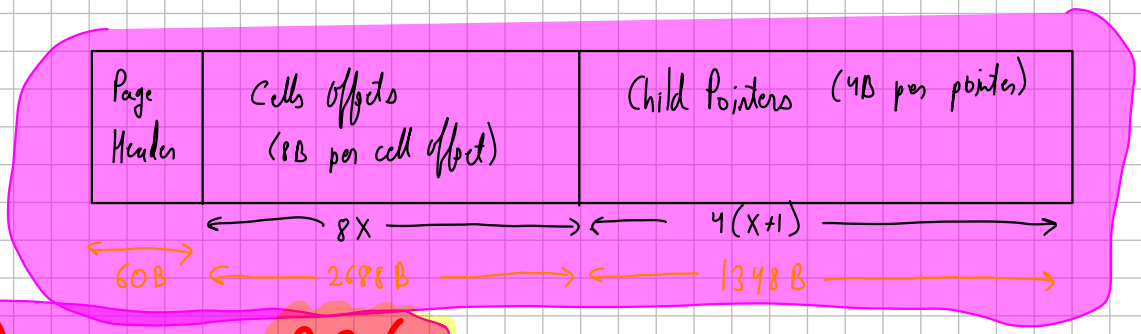



⇒ Page Header needs to hold info such as:

- ↳ Identification Num (4B)
- ↳ Page_Type (1B) ⇒ 33
- ↳ Current num cells in the node (Block_size) (2B)
- ↳ Padding (to make size 60B)

| X | 256 | 291 | 326 | 361 | 396 | 431 | 466 | 501 | 536 | 571 | 606 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Each Cell Offset Size | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| Cell Offset Area | 2048 | 2328 | 2608 | 2888 | 3168 | 3448 | 3728 | 4008 | 4288 | 4568 | 4848 |
| Child Pointers Area | 1028 | 1168 | 1308 | 1448 | 1588 | 1728 | 1868 | 2008 | 2148 | 2288 | 2428 |
| Total Area (Bytes) | 3076 | 3496 | 3916 | 4336 | 4756 | 5176 | 5596 | 6016 | 6436 | 6856 | 7276 |
| Total Area (kB) | 3.00390625 | 3.4140625 | 3.82421875 | 4.234375 | 4.64453125 | 5.0546875 | 5.46484375 | 5.875 | 6.28515625 | 6.6953125 | 7.10546875 |
| Worst Case Data Size (kB) | 512 | 582 | 652 | 722 | 792 | 862 | 932 | 1002 | 1072 | 1142 | 1212 |
| | 1020 | 600 | 180 | -240 | -660 | -1080 | -1500 | -1920 | -2340 | -2760 | -3180 |



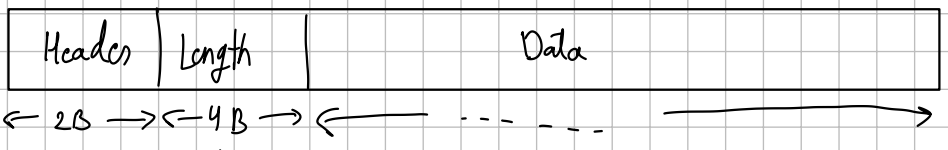




X = Max_Degree = 336

| X | 326 | 331 | 336 | 341 | 346 | 351 | 356 | 361 | 366 | 371 | 376 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Each Cell Offset Size | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| Cell Offset Area | 2608 | 2648 | 2688 | 2728 | 2768 | 2808 | 2848 | 2888 | 2928 | 2968 | 3008 |
| Child Pointers Area | 1308 | 1328 | 1348 | 1368 | 1388 | 1408 | 1428 | 1448 | 1468 | 1488 | 1508 |
| Total Area (Bytes) | 3916 | 3976 | 4036 | 4096 | 4156 | 4216 | 4276 | 4336 | 4396 | 4456 | 4516 |
| Total Area (kB) | 3.82421875 | 3.8828125 | 3.94140625 | 4 | 4.05859375 | 4.1171875 | 4.17578125 | 4.234375 | 4.29296875 | 4.3515625 | 4.41015625 |
| Worst Case Data Size (kB) | 652 | 662 | 672 | 682 | 692 | 702 | 712 | 722 | 732 | 742 | 752 |
| | 180 | 120 | 60 | 0 | -60 | -120 | -180 | -240 | -300 | -360 | -420 |

## Data inside DataPage

↳ So, as to be able to tell the start & end of a data slice in the datapage, we need data header and size of data.




↳ Will allow upto 4GB of Data

# File Structure

↳ Now that individual nodes structure is decided. I need to layout how these nodes will arrange themself in the file. Currently I'm thinking having just all the pages one after another....

↳ Now I have the node pages (all of them with their respective data pages)

↳ Puzzled about how to get:

- ↳ All the data pages need to be next to the node page?
- ↳ How will I know where a page is there in the file?
- ↳ How will I know which page is the root node of the B-Tree?
- ↳ How will I create new nodes (pages)?
- ↳ How will I delete nodes (pages)?
- ↳ How will I deal with fragmentation in the file (due to deletion)?

How File might end up looking like:

| File Header | Some Pages | Free Space | Some Pages | Free Space | Some Pages | Free Space |
|---|---|---|---|---|---|---|

↳ Now the above questions become very important to answer...

↳ What if, I save the nodes (and their data pages) in the level-order of B-Tree?

- ↳ This might make it a bit easier to locate the page we need... But how will I deal with insertion of nodes and deletion?
- ↳ If I store the level-order in reverse-order (i.e. leafs to root) then insertion will be easier since in B-Tree a new node is always added at the root level ✗
  - ↳ not true
  - ↳ Splits also cause new node creation at the internal nodes
- ↳ I don't think level order will work because creating new nodes at internal nodes level can potentially ask for shifting all pages.... Massively reducing the efficiency...

↳ What if each page's id is actually the offset in the file?

↳ This would simplify locating the root node and any node for that matter. Thus, the order in which nodes are stored doesn't matter (except maybe for optimisations?)

↳ Puzzled about how to get:
- ↳ All the data pages next to be next to the node page? ✓
- ↳ How will I know where a page is there in the file? ✓
- ↳ How will I know which page is the root node of the B-Tree? ✓
- ↳ How will I create new nodes (pages)? ✓ → Just make new page at end of file
- ↳ How will I delete nodes (pages)? ✓ → Just remove the page and no complexity will arise.
- ↳ How will I deal with fragmentation in the file (due to deletion)? ✓ → Time-to-time defragmentation should be fine...

↳ Will be even better if I make 4kB slabs in the file and instead of saving offset bytes as Page Id. Save pages index (as in nth page from start) as Page Id.

## File Header

↳ Will always be the first page (page index 0) of the file

↳ Will hold info such as
- ↳ Random Identification Number (4B)
- ↳ Page Type (1B) ⇒ Will tell what type of Page it is → Value here will 21
- ↳ Total pages (4B)
- ↳ Size of Total Data (8B)
- ↳ Root node page id (4B)
- ↳ Space_Table_size (2B) ⇒ To tell the current size of this table
- ↳ Available Spaces Table ⇒ (384 Bytes)
  - ↳ Max 200 rows

| Page Id (Offset) (4B) | Num of Pages (1B) |
|---|---|
| — | — |
| — | — |
| — | |

64

- ↳ Padding

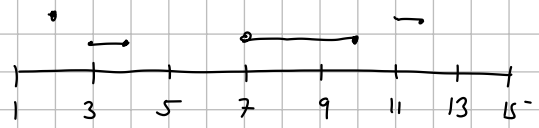
| Page Type | Id (in Page_Type field) |
|---|---|
| File Header | 21 |
| Node Page | 33 |
| Data Page | 45 |

Todo

- ~~Add function to easily Add, Delete pages in a DB file~~ ✓
- ~~Add defragmenter for the DB file~~ ✓
- ~~Add function to add a key and data to a node~~ ✓
- ~~Add function to add a data page to an already made node/data page~~ ✓

## Merge Intervals Problem

I/p : [3, 2) , [7, 4) , [2, 1), [11, 2) ⇒




O/p : [2, 3) , [7, 6]

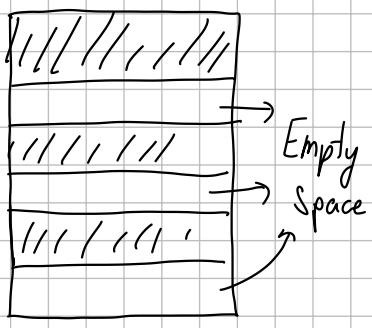
[2, 3, 4)

Approach-1

- Sort i/p based on 0th elem ⇒ O(n log n)
- Then keep combining till valid ⇒ O(n)

T: O(n log n)
S: O(n)

# Defragmentor for Datapage

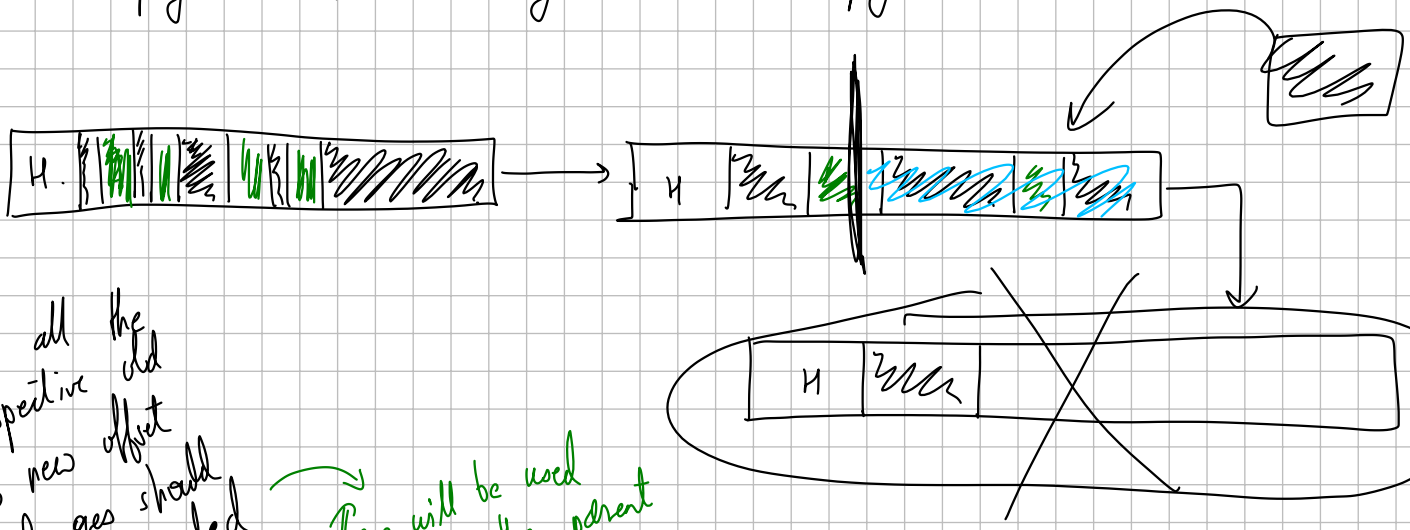



2 cases
- A data exists which extends to the next datpage... ↑ Case-1
- No such data exists ↑ Case-2

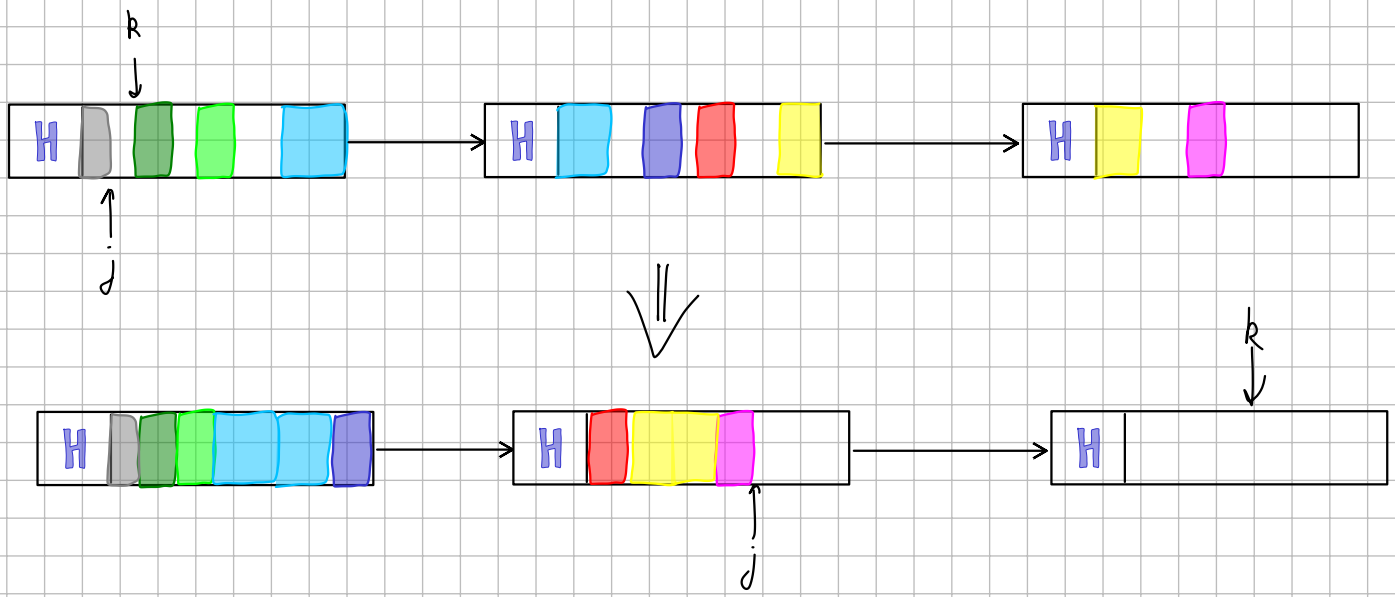
Case-1 can handle Case-2

## CASE-1

→ In any arbitary datapage there can be atmost 2 such datas, one from left continued from previous datapage, and, other starting in this datapage and then continuing to the next datapage.




And all the respective old to new offset changes should be recorded → These will be used to update the parent nodepage



# Putting in the Datapage

↳ Main problems

→ Unlike defragmentation, the data here needs to be put in the spaces first where there is space available.

Eg:

↳ Data should be put here first

↳ 

→ So, we should naturally use ~~First Fit~~ Best-Fit always instead of Best-fit.

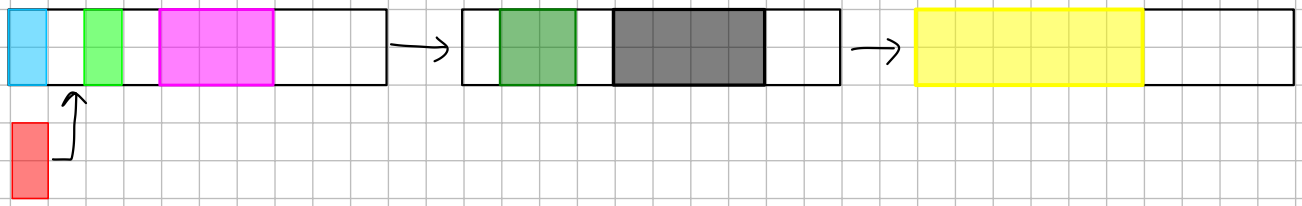
→ The problem comes when the data we want to put can't fit in the datapage.

↳ If datapage can fit in datapage when defragmented then defragmentation is done

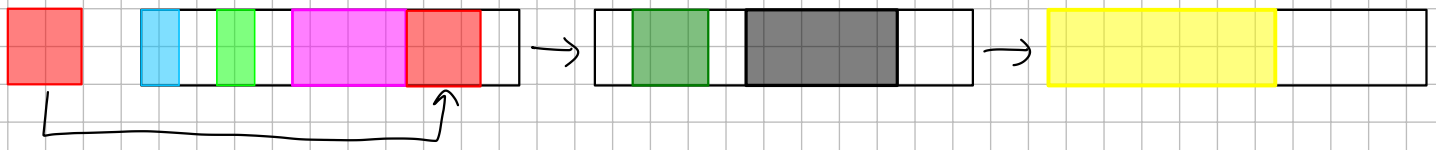
↳ But, if data cannot fit inside the datapage even if defragmentation is done, then what should we do?

↳ Do, we defragment again?

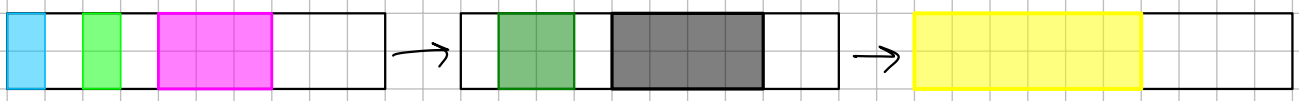
## CASE - 1

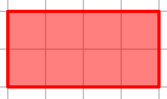


## CASE - 2

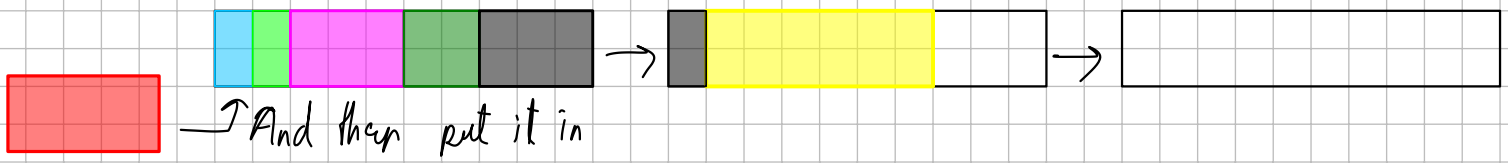


## CASE - 3

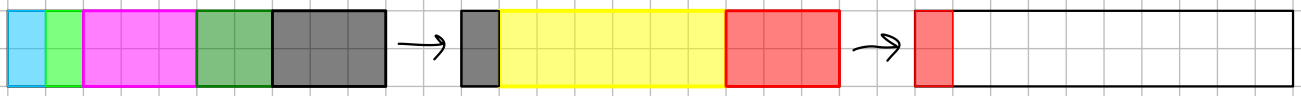




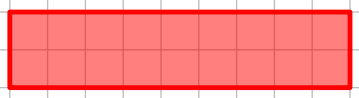
Can fit here but only after defragmentation.
So, we defragment, and



↑ And then put it in

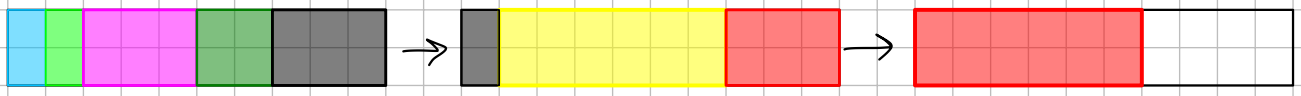


## Case - 4



↑ Very big data, needs multiple datapages to fit...

Again defragmentation is needed...



## Idea

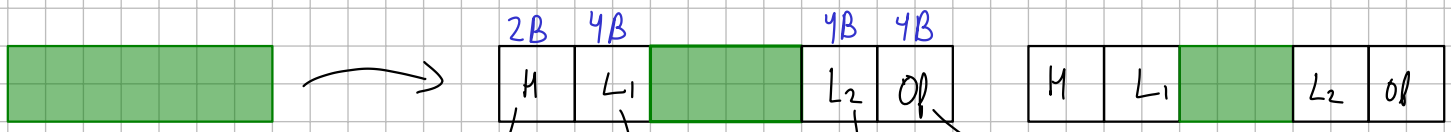
↳ Currently, a big problem is coming from trying to save the data in on continuous block... What if also allow the defragmentation of data?

- ↳ This will reduce the need for defragmentation drastically.
- ↳ Adds overhead in memory and computation for reads and writes.
- ↳ Makes it more complicated
- ↳ Smaller data might also need multiple datapage accesses to read/write
- ↳ Deletion will also lead to lots fragmentation...

# Considering Fragmented Data

↳ I think only experimentation will tell which technique is better. But, I dont want to implement both and then experiment !!

↳ So, lets see putting data in here... First since all data can be fragmented we will need a way to see if a sequence of bytes are indeed a part of data or not and if yes, where is the next part of the data....




**Header** ⇓ To identify this as start of a legitimate piece of data

**Length** of the data in this strip

**Length** of the remaining data (which lies in other data strips)

**Offset** ⇒ To point to where next data strip is stored.

⇒ Too Wasteful! This is taking up 14B for EVERY data strip... Did some calculation and found out that, $\frac{3584\,B}{14\,B} = \boxed{256 < X = 366}$

main-data available in a Datapage

So, even if store OB of data for all keys even then 1 Datapage wont be enough!

This is not what I want !!

↳ The data overhead is too big.

↳ Possible the data read might cause the seeking of all the datapages

↳ It is also possible in non-fragmented data, but more so here

↳ It is more complicated to implement.

Reasons why I am ditching the idea of having fragmented data.