Chapter 17: Amortized Analysis
Chapter 18: B-Trees

17.1-3) Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2, and 1 otherwise. User aggregate analysis to determine the amortized cost per function.

Solution:

operations	1	2	3	4	5	6	7	8	9	...	n
cost	1	2	1	4	1	1	1	8	1	...	n or 1

$$\begin{aligned} Total\ cost &= \sum_{k=0}^{\lfloor lg\ n \rfloor} 2^k + \sum_{k=0}^{\lceil n - lg\ n \rceil} 1 \\ &= \frac{1 - 2^{\lfloor lg\ n \rfloor + 1}}{1 - 2} + n - \lfloor lg\ n \rfloor \\ &= \frac{1 - 2^{lgn} \cdot 2^1}{-1} + n - \lfloor lg\ n \rfloor \\ &= \frac{-2 \cdot n + 1}{-1} + n - \lfloor lg\ n \rfloor \\ &= 2n + n - \lfloor lg\ n \rfloor - 1 \\ &= 3n - \lfloor lg\ n \rfloor - 1 \\ &\leq 3n \end{aligned}$$

The total cost to compute n operations is $O(n)$. The average cost of each operation in the worst case is then $\frac{O(n)}{n} = O(1)$. That is, the amortized cost per function is $O(1)$.

17.2-1) Suppose we perform a sequence of stack operations on a stack whose size never exceeds k. After every k operations, we make a copy of the entire stack for backup purposes. Show that the cost of n stack operations, including copying the stack, is $O(n)$ by assigning suitable amortized costs to the various stack operations.

Solution:
Assign the amortized cost to the operations as follow:

operation	amortized cost
PUSH	3
POP	0
MULTIPOP	0
COPY	0

PUSH gets assigned a cost of 3 to pay for the item pushed, 1 credit for the future POP and 1 credit for the future COPY. The POP cannot be used unless there are items on the stack. POP does not need to pay for itself since each item on the stack is reserving 1 credit for it. MULTIPOP contains POPs, so its cost is the cost of the number of POPs used and since POP is already taken care of, MULTIPOP does not incurr a cost. The COPY does not incurr a cost since PUSH has credit for this as well. Thus, in the worst case, the amortized cdost for n operations (n pushes) is $O(3n) = O(n)$.

We know that the credit $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$ (where $\hat{c}_i$ is the amortized cost of the ith operation with actual cost c_i) will always be satisfied because POPs and MULTIPOPs cannot occur if items have not been pushed and COPY does not incurr any cost if the stack is empty.

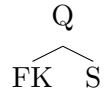
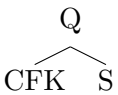
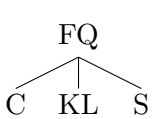
17.2-2) Redo Exercise 17.1-3 using an accounting method of analysis.

Solution:
From Excersize 17.1-3, we have that the total cost is bounded above by $3n$. This implies that the total cost of n activities is at most $3n$. Let us assign each operation a cost of 3. Then the total amortized cost is $O(3 \cdot n) = O(n)$, which gives us the cost per operation $\frac{O(n)}{n} = O(1)$ as obtained using aggregate analysis.

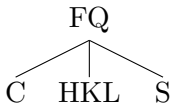
We can guarantee that the credit will always be greater than or equal to zero because for n operation, the total amortized cost is $\sum_{i=0}^{n} \hat{c}_i = 3n$ and the actual cost is $\sum_{i=0}^{n} \hat{c}_i \leq 3n$. Thus the credit $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$ is always satisfied.

18.2-1 Show the results of inserting the keys
F, S Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E
in order into an empty B-tree with minimum degree 2. Draw only the configurations of the tree just before some node must split, and also draw the final configuration.

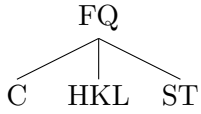
Solution:
The B-tree has minimum degree 2. This means that each node can have a maximum of 2(2)=4 children and 2(2)-1=3 keys.

empty B-Tree	∅
F	F
S	FS
Q	FQS
K	
C	
L	

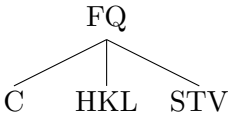
H


FQ
C
HKL
S

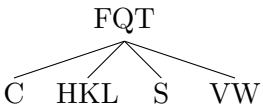
T


FQ
C
HKL
ST

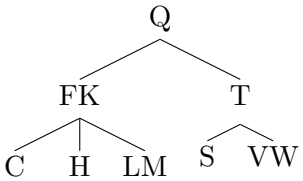
V


FQ
C
HKL
STV

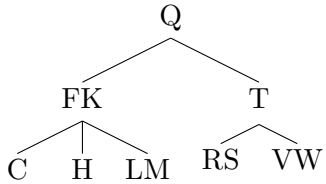
W


FQT
C
HKL
S
VW

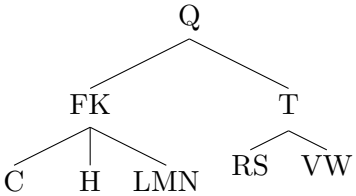
M


Q
FK
T
C
H
LM
S
VW

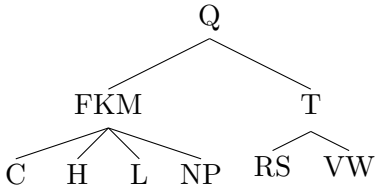
R


Q
FK
T
C
H
LM
RS
VW

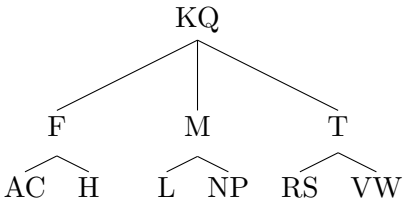
N


Q
FK
T
C
H
LMN
RS
VW

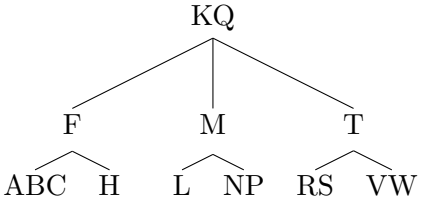
P


Q
FKM
T
C
H
L
NP
RS
VW

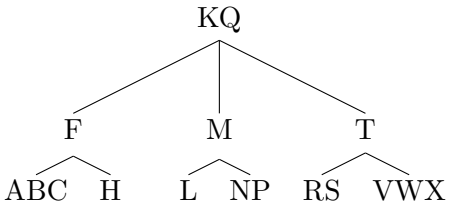
A


KQ
F
M
T
AC
H
L
NP
RS
VW

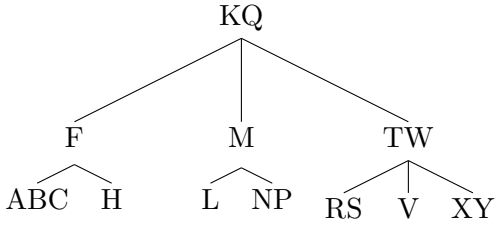
B


KQ
F
M
T
ABC
H
L
NP
RS
VW

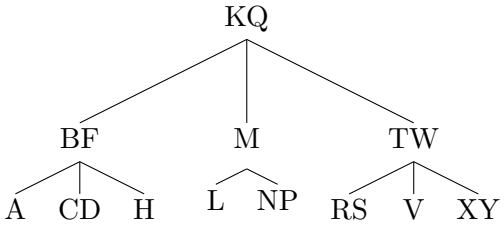
X



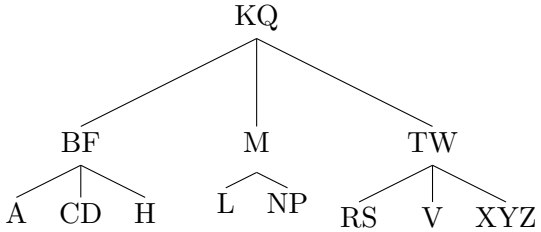
Y



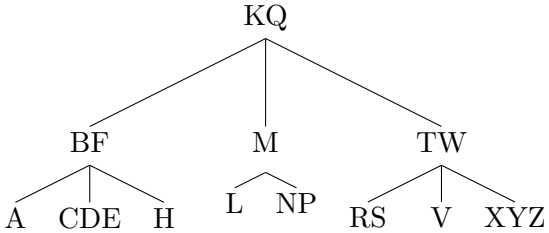
D



Z



E



18.2-2 Explain under what circumstances, if any, redundant DISK-READ or DISK-WRITE operations occur during the course of executing a call to B-TREE-INSERT. (A redundant DISK-READ is a DISK-READ for a page that is already in memory. A redundant DISK-WRITE writes a disk a page of information that is identical to what already stored there.)

Solution:
B-TREE-INSERT calls B-TREE-INSERT-NONFULL and may call B-TREE-SPLIT-CHILD. B-TREE-INSERT-NONFULL may call B-TREE-INSERT-NONFULL and B-TREE-SPLIT-CHILD.

DISK-WRITE is done in only two places:

1. B-TREE-SPLIT-CHILD: to update the parent and the two children affected
2. B-TREE-INSERT-NONFULL: when the key is inserted in a leaf node, it is used to update the leaf node

So, there is no redundancy here.

DISK-READ is done in only one place:

1. B-TREE-INSERT-NONFULL: to transverse down the tree when looking for the key

   In the logic, if the current node x is not a leaf, we determine the index of the child to which the key belongs. Once the index is found, the child is read. If the child node is full, it is split, then it is determined which of the two children from the split the key belongs to, and then it is searched within that subtree. (Note that the subtree might be a real subtree or just a leaf). In the recursive call, if the "subtree" is a leaf, then no DISK-READ is done. If it's not a leaf, then a DISK-READ to the next child is done to repeat the process. In this way, it is ensured that there is no redundant DISK-READ.

Thus, none of the DISK-READs or DISK-WRITEs are redundant during the course of executing a call to B-TREE-INSERT.

References

[1] Cormen, Thomas. H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, Third Edition*. MIT Press, Cambridge, MA, 2009.

[2] http://www.cs.unm.edu/ saia/561-f09/lec/lec8.pdf