CS240 Assignment 4

Nissan Pow

July 11, 2006

Question 1

(a)

Search	MTF	Transpose	MTM
3	[0,1,2,3,4,5,6]	[0,1,2,3,4,5,6]	[0,1,2,3,4,5,6]
1	[3,0,1,2,4,5,6]	[0,1,3,2,4,5,6]	[0,1,2,3,4,5,6]
6	[1,3,0,2,4,5,6]	[1,0,3,2,4,5,6]	[0,2,3,1,4,5,6]
5	[6,1,3,0,2,4,5]	[1,0,3,2,4,6,5]	[0,2,3,6,1,4,5]
5	[5,6,1,3,0,2,4]	[1,0,3,2,4,5,6]	[0,2,3,5,6,1,4]
2	[5,6,1,3,0,2,4]	[1,0,3,2,5,4,6]	[0,2,3,5,6,1,4]
4	[2,5,6,1,3,0,4]	[1,0,2,3,5,4,6]	[0,3,5,2,6,1,4]
	[4,2,5,6,1,3,0]	[1,0,2,3,4,5,6]	[0,3,5,4,2,6,1]

(b)

Use the following array to answer questions a and b.

Search	MTF	Transpose	MTM
Start	`[0,1,2,3,4,5,6]`		
3	c=4 , m=5	c=4 , m=3	c=4 , m=0
1	c=3 , m=4	c=2 , m=3	c=2 , m=4
6	c=7 , m=8	c=7 , m=3	c=7 , m=5
5	c=7 , m=8	c=7 , m=3	c=7 , m=5
5	c=1 , m=0	c=6 , m=3	c=4 , m=0
2	c=6 , m=7	c=4 , m=3	c=2 , m=4
4	c=7 , m=8	c=6 , m=3	c=7 , m=5
Total comparisons	35	36	33
Total movements	40	21	23

(c)

Since MTF, Transpose, and MTM are all within a constant factor of C_{OPT} (MTF $< 2C_{OPT}$ from slides), the asymptotic order of the average performance of each of those heuristics is $O(1)$.

Question 2

Use a data structure that's a hybrid between a BTree and a Heap, which I shall call BHeap. A BHeap of order d is a Heap such that:

- every node has $\leq d$ children
- every non-root node has $\geq \lceil \frac{d}{2} \rceil$ children
- all external nodes have the same depth
- every node itself is structured as a Heap

Choose d such that one d-node fills exactly one disk page. Thus the height of the BHeap will be $O(\log_B n)$. Basically to SiftUp/SiftDown, we retrieve a "node" (which is actually a heap), perform the usual SiftUp/SiftDown on this node, and continue retrieving nodes and sifting up/down if necessary. This will require $O(\log_B n)$ disk accesses. Hence, ExtractMin and Insert each use at most $O(\log_B n)$ disk accesses.

Question 3

(a)

	Insert 31	Insert 26	Insert 16	Insert 23	Insert 11	Insert 30	Insert 20
0:						30	20
1:	31	31	31	31	11	11	11
2:					31	31	30
3:				23	23	23	23
4:							31
5:							
6:		26	16	16	16	16	16
7:			26	26	26	26	26
8:							
9:							

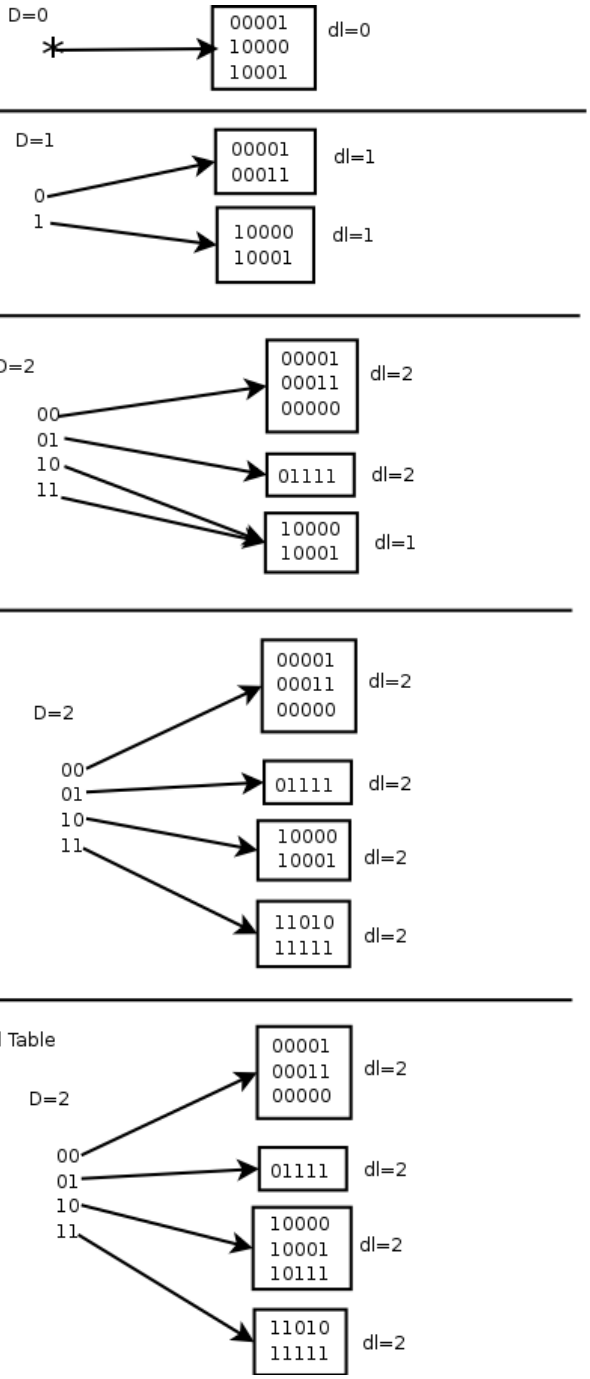
(b)

When inserting any key k, if we encounter a key k' with $k' > k$, we swap to put k in this position and then proceed to insert k'. This ensures that for all keys with the same value of $f(x)$, the keys encountered before it in its probe sequence are all smaller than it. Now consider any arbitrary probe sequence with last value x; by the argument above, x is the largest value in its probe sequence. Now suppose another key, y, with $y > x$ and $f(x) \neq f(y)$, is supposed to be inserted in the middle of this probe sequence. Since we are doing linear probing, there cannot be any empty spaces in x's probe sequence. Therefore y has to be probed as well. And since $y > x$, y will be inserted *after* the end of x's probe sequence, thus maintaining the assertion above. Hence for any key, the keys encountered before it in its probe sequence are all smaller than it.

(c)

When searching for an element,x, that's not present in the hash, if we encounter an element that's greater than x *before* finding x, we can immediately terminate the search. This holds because of part (b) above. In the best case (or even average case), it improves the runtime of searches for elements not in the hash since we are able to terminate the search without probing the entire probe sequence.

Question 4


D=0
00001
10000
10001
dl=0
D=1
00001
00011
dl=1
0
1
10000
10001
dl=1
D=2
00001
00011
00000
dl=2
00
01
10
11
01111
dl=2
10000
10001
dl=1
D=2
00001
00011
00000
dl=2
00
01
10
11
01111
dl=2
10000
10001
dl=2
11010
11111
dl=2
Final Table
D=2
00001
00011
00000
dl=2
00
01
10
11
01111
dl=2
10000
10001
10111
dl=2
11010
11111
dl=2