CS 225 Spring 2019 :: TA Lecture Notes
3/13 B Tree Analysis + Hash

By Wenjie

- The height of a BTree determines maximum number of **disk/network/space seeks** possible when searching for data.
- The height of a BTree is **$\log_m$ (n)** where **m** is the order of the BTree.
- Therefore **seeks <= $\log_m$(n)**

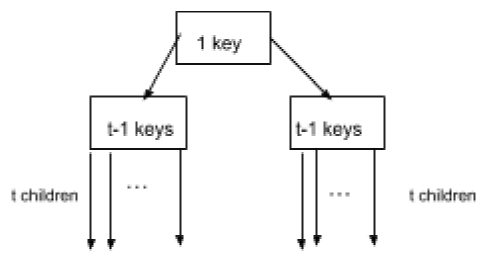
To prove the above property, we need to find a relationship between the number of keys **n** and the height of the BTree **h**. In other words: how is the height **h** bounded by the keys **n?**

Minimum number of keys in a BTree of height **h** and order **m**:

Level by level analysis:

Let t= ceil(m/2)

Level	Nodes	Keys	Children
Root	1	1	2
1	2	2*(t-1)	2*t
2	2*t	2*t*(t-1)	2*t^2
3	2*t^2	2*t^2*(t-1)	2*t^3
...	...	....	....
h	2*t^(h-1)	2*t^(h-1)*(t-1)	0 (leaves)



Min total nodes = $1+2+2*t+2*t^2 ... +2*t^{(h-1)} = 1+2*\sum_{i=0}^{h-1} t^i = 1+2*\frac{t^{h-1+1}-1}{t-1} = 1+2*\frac{t^h-1}{t-1}$

Min total keys = $1+2*\frac{t^h-1}{t-1}*(t-1) = 2 * celling\,(m/2)^k - 1 = 2*t^h-1$

Root has 1 key

All other nodes have >= t-1 keys

Thus: $n \geq 2*t^h-1$ *(for any BTree of height h and order m)*

Solving for h: $\frac{n+1}{2} \geq t^h \Rightarrow log_t(\frac{n+1}{2}) \geq h$

Since $t = ceil(m/2)$ we can say: $log_t(\frac{n+1}{2}) \sim log_m(n)$

CS 225 Spring 2019 :: TA Lecture Notes
3/13 B Tree Analysis + Hash

By Wenjie

Thus we have: $h \leq log_{m}(n) \Rightarrow seeks \leq log_{m}(n)$

Given m=101 , a BTree of height h=4 has:

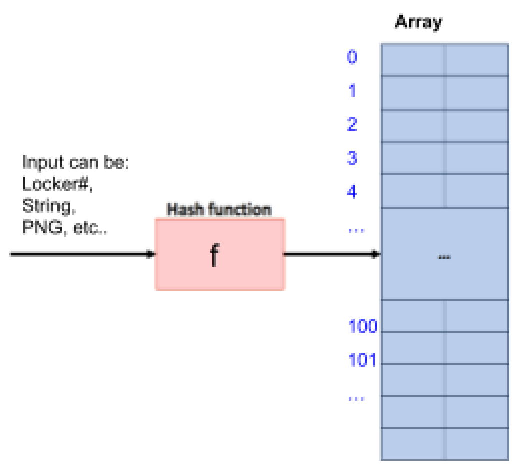
Minimum keys: $2 * t^{h} - 1 = 2 * ceil(101/2)^{4} - 1 = 2 * (51)^{4} \approx 12.5\ million$

Maximum keys: //Practice problem

Hashing Introduction

As the high school locker number-to-student name, it is a one-to-one mapping:

Locker #	Name
103	Craig
92	Wade
330	...
46	
124	



Keyspace = all possible locker numbers

There are **3 components** of Hash Tables:

1. The hash function: f(h) -> Integers
   - Choose a good hash function is tricky
   - Do not use self created hash function
2. The compression: Integer -> array
3. What happened in chaos

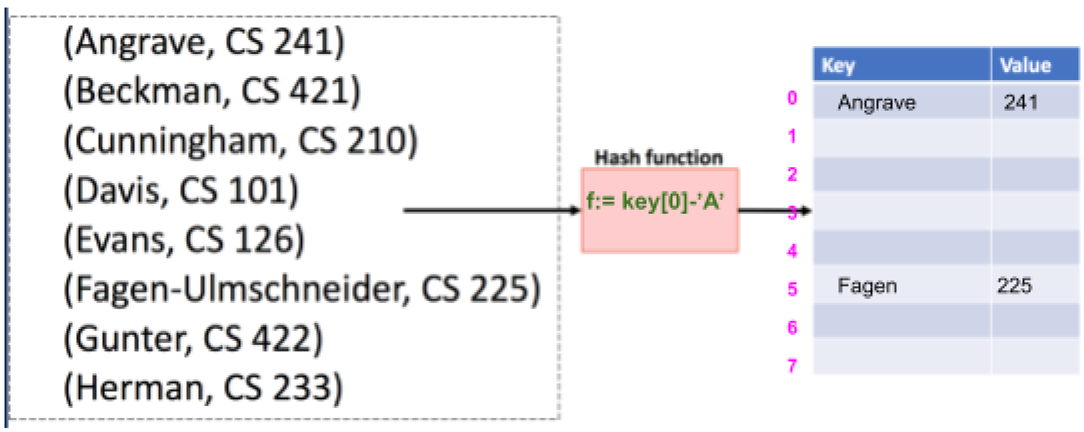
CS 225 Spring 2019 :: TA Lecture Notes
3/13 B Tree Analysis + Hash

By Wenjie

Ideally, a Perfect Hash Function is a **one-to-one** function, where for each **data / input**, the hash function produces a **unique output**.

Suppose we want to map the following (faculty, course) pairs into a hash table.
Since no two faculty first names start with the same letter; we can define a perfect hash function

f:= (faculty, course) → first letter of first name ASCII index



A **good** hash function should be:

- 1. Computation time take O(1)
- 2. Deterministic
- 3. Satisfy the SUHA = simple uniform hashing assumption where p(f(k1)) = p(f(k2)) = 1/n