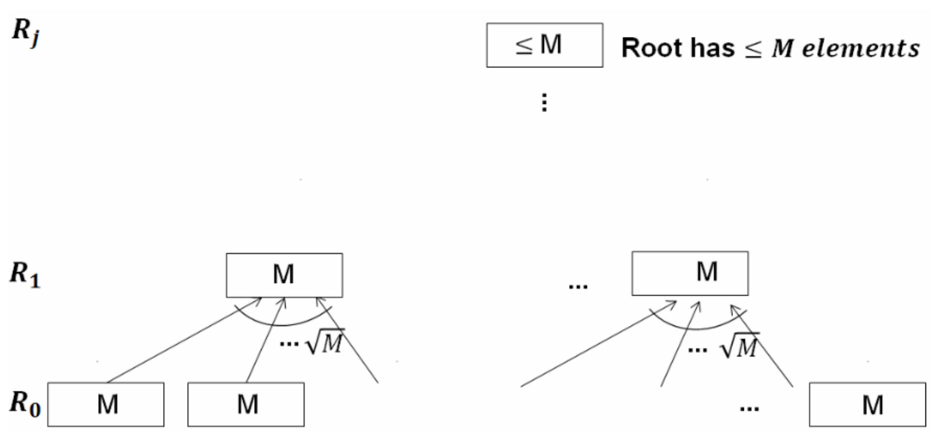
CSE 5095: Research Topics in Big Data Analytics 2/4/2014

Notes by: Ioannis Papavasileiou

Deterministic out-of-core Selection

input: A sequence $X = k_1, k_2, \ldots, k_N$ and an integer $i, 1 \leq i \leq N$
output: the i-th smallest element of X

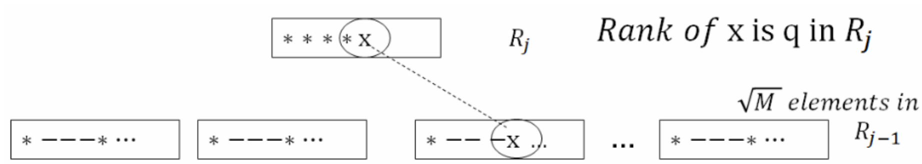
We will employ deterministic sampling. Recall that the BFPRT algorithm employs a simple form of deterministic sampling. In any out-of-core algorithm, we typically are interested in counting the number of I/O operations and we are not concerned with the computation time.



Think of a tree where the degree is $\sqrt{M}$ and each leaf has M elements. First we sort each leaf and every leaf sends its keys with ranks $\sqrt{M}, 2\sqrt{M}, ..., M$ to its parent. We continue this process in every node until we reach a node that has $\leq M$ elements. This node is the Root. The leaves are at level 0 and let the level of the root be j.

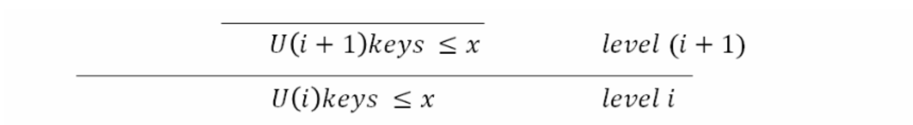
Without loss of generality lets assume that $|R_j| = M$.
In the following lines we are going to describe one level of sampling.
Pick $l_1 \& l_2$ from R_j such that
$Rank(l_1, R_j) = \frac{i|R_j|}{N} - \delta$ and $Rank(l_2, R_j) = \frac{i|R_j|}{N} + \delta$
δ is going to be defined in the analysis.
Let x be an element of R_j whose rank in R_j is q. In this case, what is $Rank(x, R_0)$?

We know that $|R_1| = \frac{N}{\sqrt{M}}$, $|R_2| = \frac{N}{(\sqrt{M})^2} = \frac{N}{M}$, ..., $|R_j| = \frac{N}{(\sqrt{M})^j} = M \Rightarrow N = M^{\frac{j+2}{2}} \Rightarrow j = 2c - 2$, where $c = \frac{\log N}{\log M}$.



$Rank(x, R_{j-1}) \geq q\sqrt{M}$
Also, $Rank(x, R_{j-1}) \leq q\sqrt{M} + (\sqrt{M} - 1)^2$.
This is because there is an uncertainty of $\sqrt{M} - 1$ contributed by every node (except one) in level $(j-1)$ to the rank of x.

Let $U(i)$ be the rank of x in R_i.



$U(i) \geq U(i+1)\sqrt{M}$,
$U(i) \leq U(i+1)\sqrt{M} + M^{\frac{j-i}{2}}\sqrt{M}$
Note that the number of nodes in level i is $(\sqrt{M})^{j-i} = M^{\frac{j-i}{2}}$ and each such node contributes an uncertainty of $\sqrt{M} - 1$ to the rank of x.

By repeated substitutions we have:

$U(i) \leq qM^{\frac{j-i}{2}} + (j-i)M^{\frac{j-i+1}{2}}$
and so:

$$U(0) \leq qM^{\frac{j}{2}} + jM^{\frac{j+1}{2}} = q\frac{N}{M} + (2c-2)\frac{N}{\sqrt{M}} \tag{1}$$

i.e.,

$$Rank(x, R_0) \in [q\frac{N}{M}, q\frac{N}{M} + (2c-2)\frac{N}{\sqrt{M}}] \tag{2}$$

We can pick l_1 such that $Rank(l_1, R_j) = i\frac{|R_j|}{N} - (2c - 2 + \epsilon)\sqrt{M}$ and l_2 such that $Rank(l_2, R_j) = i\frac{|R_j|}{N} + (2c - 2 - \epsilon)\sqrt{M}$.

Using equations 1 and 2, $Rank(l_1, R_0) \in [i - (2c - 2 + \epsilon)\frac{N}{\sqrt{M}}, i - \epsilon\frac{N}{\sqrt{M}}]$ and $Rank(l_2, R_0) \in [i + (2c - 2 + \epsilon)\frac{N}{\sqrt{M}}, i + (4c - 4 + \epsilon)\frac{N}{\sqrt{M}}]$.

As a result, we see that the number of input keys whose values are in the range $[l_1, l_2]$ is $\leq (6c - 6 + 2\epsilon)\frac{N}{\sqrt{M}}$.

This is how one level of sampling works. The actual selection algorithm uses the above sampling process as a building block.

Algorithm

To begin with all the input keys are alive; $n \leftarrow N$;
/* n is the number of alive keys */
repeat
  do one level of sampling;
  compute l_1 and l_2 as described above;
  eliminate all the alive keys that are not in the range $[l_1, l_2]$;
  Adjust n and i accordingly;
  if $n \leq M$ **then**
    quit the loop
  end if
until forever
perform an appropriate selection on the alive keys and output the correct key

ANALYSIS:

The number of alive keys reduces by a factor of $\Omega(\sqrt{M}/c) = \Omega\left(\frac{\sqrt{M}\log M}{\log n}\right)$ in each iteration of the Repeat loop. If N is a polynomial in M, then the number of iterations is $O(1)$.

In any iteration of the repeat loop the number of I/O operations is:

$\frac{n}{B} + \frac{n}{B\sqrt{M}} + \frac{n}{BM} + ... =$

$= \frac{n}{B}(1 + \frac{1}{\sqrt{M}} + \frac{1}{M} + ...) \leq \frac{n}{B}(\frac{1}{1-\frac{1}{\sqrt{M}}}) = O(\frac{n}{B})$

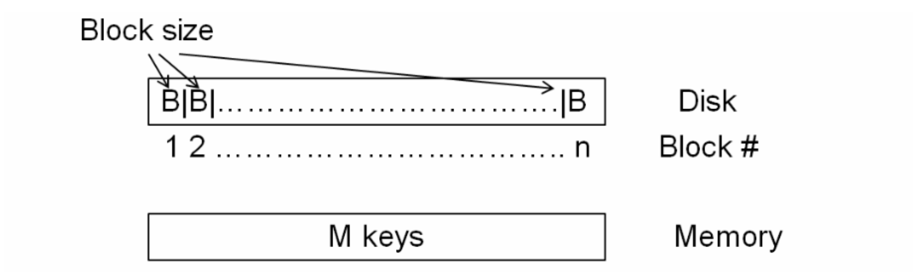
Lemma: We can do out-of-core selection deterministically in $O(\frac{N}{B})$ I/O operations.

A lower bound on I/O operations required to sort N keys

Lemma: Sorting N keys needs $\Omega\left(\frac{N}{B}\frac{\log\frac{N}{B}}{\log\frac{M}{B}}\right)$ I/O operations.

Proof: Assume:

1. No new keys are generated
2. The disk is thought of as consisting of $n = \frac{N}{B}$ blocks and the I/O's are done only with respect to these blocks



To begin with, there are $N!$ permutations such that the correct answer could be any one of these. The number of permutations that can be generated in one I/O is $\binom{M}{B}B!$.

After one I/O the number of permutations remaining for the adversary is reduced to $\frac{N!}{\binom{M}{B}B!}$.

So we can see that the number of permutations remaining after t I/O operations is $\frac{N!}{\left(\binom{M}{B}B!\right)^t}$. However, when $t > n$, the $B!$ term vanishes since there are only n I/O operations in which we can bring an unseen block from the disk to the core memory.

$\Rightarrow$ number of permuations remaining after t operations is $\leq \frac{N!}{\binom{M}{B}^t(B!)^{\frac{N}{B}}}$

we want this to be ≤ 1. $\Rightarrow N! < \binom{M}{B}^t(B!)^{\frac{N}{B}}$.

We'll use the following (crude) approximations: $\log(x!) \approx x\log x$ and $\log\binom{a}{b} \approx b\log\frac{a}{b}$.

So, $N \log N \leq t \log \binom{M}{B} + \frac{N}{B} \log(B!) = tB \log \frac{M}{B} + \frac{N}{B} B \log B \Rightarrow$
$N \log \frac{N}{B} \leq tB \log \frac{M}{B} \Rightarrow$
$t \geq \frac{N}{B} \frac{\log \frac{N}{B}}{\log \frac{M}{B}}$ ■