Reminder

• search tree of degree $\Theta(B) \Rightarrow$ height $\lg_B N$
  
  ▶ **leaves**: pointers to $b$ strings [$b = \Theta(B)$]
  
  ▶ **internal**: separators $L(v_1), R(v_1), ..., L(v_b), R(v_b)$

• search $P$: at every node with children $v_1, ..., v_b$
  
  ▶ load 1 block containing $L(v_1), ..., R(v_b)$: one IO
  
  ▶ load $\lg B$ strings & compare with $P$ (bin. search)
    
    - $O(|P|/B)$ IOs per comparison

• **total**: $O(\lg_B N \times \lg B \times |P|/B) = O(|P|/B \lg N)$
$D = \{\text{alan}, \text{turing}, \text{ate}, \text{an}, \text{acid}, \text{apple}\}$, $B = 8$

$D = \{\text{alan}, \text{turing}, \text{ate}, \text{an}, \text{acid}, \text{apple}\}$, $B = 8$
First Improvement

• **add Patricia Tries** (PT) to B-tree nodes

• **PTs** for string set \( S = \{S_1, ..., S_k\} \):
  
  ▶ compact trie over \( S \) (cf. suffix tree)
  
  ▶ edges: store 1st (branching) character & length

  ▶ size: \( O(k) \) [**NOT** \( O(|\Sigma|S_i||) \)!!!]

• **blind search**: skip characters not stored
  
  ▶ \( \sim \) false matches
Correct Insertion Point

- say blind search ends at leaf $\lambda$
  - compute $L = \text{LCP}(P, \lambda)$
  - $u$: 1st node on root-to-$\lambda$ path with $d \geq L$ chars

1. $d = L$, $c_i < P_{L+1} < c_{i+1}$
2. $d > L$

\[ \text{(1) } d = L, \ c_i < P_{L+1} < c_{i+1} \quad \text{(2) } d > L \]
Blind Search: IOs

- at every node with children $v_1,\ldots,v_b$:
  - load PTs: one IO with $S=L(v_1),\ldots,R(v_b)$
  - search PTs for $\lambda$: no IOs
  - load one string and compare with $P$: $O(|P|/B)$ IOs
  - identification of insertion point: no IOs

- total: $O(|P|/B \ lg_B N)$ IOs
Second Improvement

• search for $P$:
  \[ \ldots \rightarrow \pi \rightarrow \sigma \rightarrow \ldots \]

• in $\text{PT}_\pi$:
  \[ \text{compute } L = \text{LCP}(P, \lambda) \]

• all strings in $\sigma$ begin with $L$

  \[ \Rightarrow \text{ in } \text{PT}_\sigma: \]
  \[ \text{compute } L' = \text{LCP}(P, \lambda') \text{ starting at } P[L+1] \]
Final Complexity

- pass matched LCPs down the B-tree
- telescoping sum \( \sum_{i \leq h} \frac{L_i - L_{i-1}}{B} \) IOs
  - height of B-tree \( h = \lg_B N \)
  - \( L_i = \) LCP-value on level \( i \) of String B-tree
- with \( L_0 = 0 \) and \( L_h \leq |P| \):
  - \( O(|P|/B + \lg_B N) \) IOs
- inserting \( P \) to \( D \) possible in \( O(|P| \cdot h) \) IOs
Outlook on Cache Oblivious Data Structures
The Model

- Like EM:
  - $M$: size of internal memory $\triangleq$ cache
  - external memory $\triangleq$ RAM
  - $B$: block transfer size

- Now: $M$ & $B$ unknown
  - analysis over all values of $M,B$

- cache oblivious algorithm:
  - achieves EM lower bound for all values of $M,B$
Thoughts on CO-Model

• Example: Scanning $N \gg M$ items
  - optimal $O(N/B)$ in EM
  - no need to know $B \Rightarrow$ cache oblivious

• assumes optimal cache replacement
  - otherwise always next block evicted $\sim M=1$
  - LRU is 2-competitive

• tall cache assumption: $M = \Omega(B^2)$
Funnelsort

- *k*-funnel: black box for **merging** COly
  - merge *k* sorted lists of total size *k*³
  - $O(k^3/B \lg_{M/B}(k^3/B)+k)$ IO's
  - space $k^2$

⇒ **Funnelsort** array $A[1,N]$:

1. split $A$ into $N^{1/3}$ segments (size $N^{2/3}$)
2. sort each segment recursively
3. merge parts with $N^{1/3}$-funnels

- IO: $T(N) = N^{1/3}T(N^{2/3})+O(N/B \lg_{M/B}N/B+N^{1/3})$
  $= O(N/B \lg_{M/B}N/B)$ [see blackboard]
**k-Funnels**

- **binary tree**
  - *k leaves*: input streams
  - *internal* nodes: mergers
  - output stream at root (merged input streams)

- **buffers** between merge nodes
  - \( h = \log k \) levels with buffers
  - size of buffers:
    - on level \( h/2 \): \( k^{3/2} \)
    - 1 upper and \( k^{1/2} \) lower \( k^{1/2} \)-funnels: recursively
Example: 16-Funnel

**Input buffers**

**Output buffer**

---

**THEOREM 38.1**

Let $T$ be a complete binary tree with $N$ leaves laid out using the van Emde Boas layout. The number of memory transfers needed to perform a search (traverse a root-to-leaf path) and a range query in $T$ is $O(\log B N)$ and $O(\log B N + K B)$, respectively.

The navigation from node to node in the van Emde Boas layout is straightforward if the tree is implemented using pointers. However, navigation using arithmetic on array indexes is also possible. This avoids the use of pointers and hence saves space.

The constant in the $O(\log B N)$ bound for search in Theorem 38.1 can be four. Further investigations of which constants are possible for cache-oblivious comparison based searching appear in [9].

### 38.2.2 $k$-Merger

In the I/O-model the two basic optimal sorting algorithms are multi-way versions of Merge-sort and distribution sorting (Quicksort) [2]. Similarly, Frigo et al. [20] showed how both merge based and distribution based optimal cache-oblivious sorting algorithms can be developed. The merging based algorithm, Funnelsort, is based on a called $k$-merger. This structure has been used as a basic building block in several cache-oblivious algorithms. Here we describe a simplified version of the $k$-merger due to Brodal and Fagerberg [15].

**Binary mergers and merge trees**

A binary merger merges two sorted input streams into a sorted output stream: In one merge step an element is moved from the head of one of the input streams to the tail of the output stream; the heads of the input streams, as well as the tail of the output stream, reside in buffers of limited capacity.

Binary mergers can be combined to form binary merge trees by letting the output buffer of one merger be the input buffer of another—in other words, a binary merge tree is a binary tree with mergers at the nodes and buffers at the edges, and it is used to merge a set of sorted input streams (at the leaves) into one sorted output stream (at the root). Refer to Figure 38.6 for an example.

An invocation of a binary merger in a binary merge tree is a recursive procedure that performs merge steps until the output buffer is full (or both input streams are exhausted); if

---

Lazy Filling

**procedure** \( \text{FILL}(v) \):

**while** (\( v \)'s output buffer not full)

**if** (left input buffer empty)

\( \text{FILL} \) (left child of \( v \))

**if** (right input buffer empty)

\( \text{FILL} \) (right child of \( v \))

perform one merge step
Size of $k$-Funnel

- recall: size of buffers:
  - on level $h/2$: $k^{3/2}$
  - upper and lower $k^{1/2}$-funnels: recursively

$$S(k) = k^{1/2} k^{3/2} + (k^{3/2} + 1) S(k^{1/2})$$

$$= \Theta(k^2)$$
IOs of $k$-Funnels (Idea)

- consider 1st recursive level where $j$-mergers have size $\leq M/3$ (coarsest level of detail)
- even though recursion continues, on level $j$ all work in cache $\Rightarrow j^3/B+j$ IO's for $j^3$ elt.s
  
  - only when input buffer empty: evict, fill $j^3$ elements in input buffer, reload $\Rightarrow$ no extra IOs
  
  - on path: only $O(\lg j k)$ such $j$-funnels, $j=\Omega(M^{1/4})$

$\Rightarrow O(k^3/B \lg M(k)+k) \rightarrow O(k^3/B \lg_{M/B}(k^3/B)+k)$