COMP 8547
Advanced Computing Concepts

Dr. Olena Syrotkina

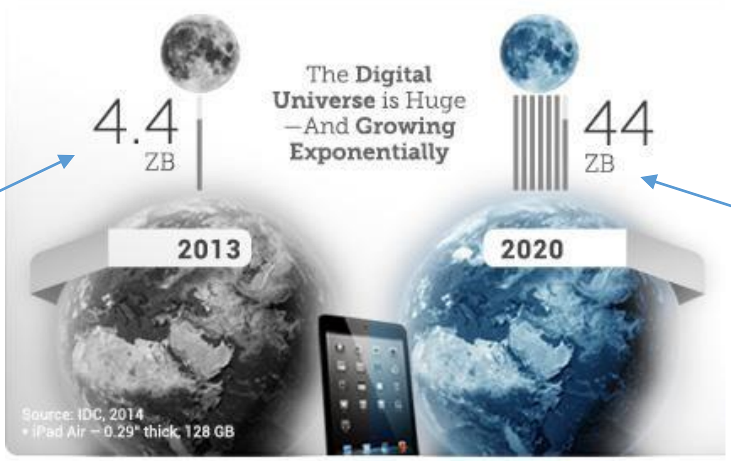
Fall 2022: 07 – Memory Management

Chapter 7 – Memory Management

Contents

Big data – brief overview

- Big data poses new challenges
- Some facts:
  - More than 24 million driver's licenses in California
  - The US has issued more than 450 million social security numbers
  - Facebook has more than 1.28 billion active users
  - Approximately 2.5 billion gmail accounts
  - 80 million skype users
  - More than 2 petabytes of data in the Sequence Read Archive at NCBI
  - Size of digital data by 2020: 44 zetabytes (44 trillion GB) [6]
- With hardware limitations, the entire ADT may not be stored in main memory
- Some data has to be stored in secondary memory (e.g., hard disk)
- Disk accesses require I/O operations
  - Much more time consuming than RAM accesses
- By comparison
  - A disk access requires about 10ms
  - Current processors can perform billions of instructions per second
  - The Intel Core i7-4770K can perform approx 23407 MIPS
- All data may not be placed in main memory
- Solution:
  - Some data stored in RAM
  - While some other in hard disk


4.4 ZB
The Digital Universe is Huge —And Growing Exponentially
44 ZB
2013
2020
Source: IDC, 2014
• iPad Air – 0.29" thick 128 GB

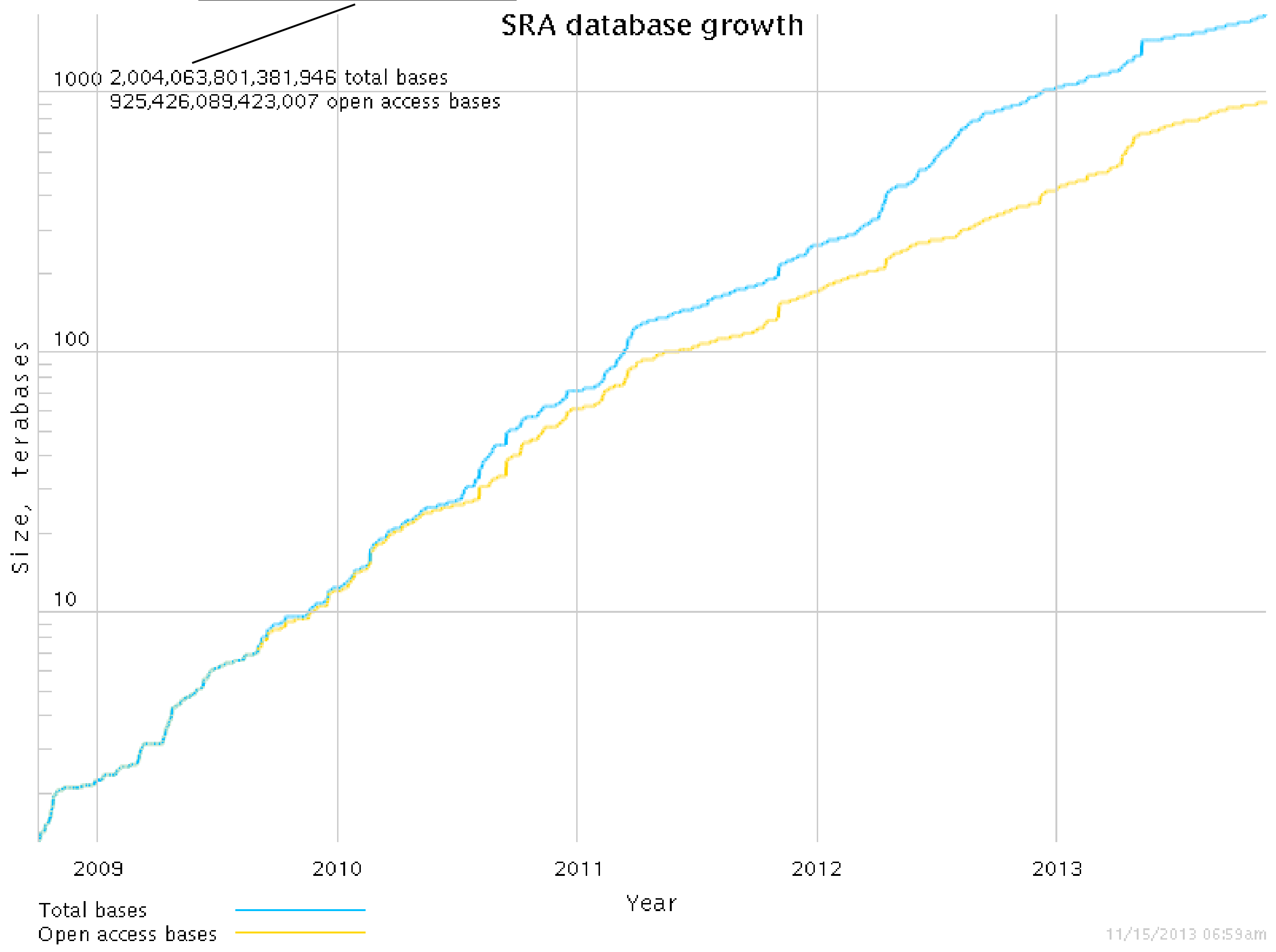
By 2013: Almost one stack of tablets from the earth to the moon[1]

By 2020: Approx. 6 stacks of tablets from the earth to the moon

Big data – next generation sequencing & bioinformatics

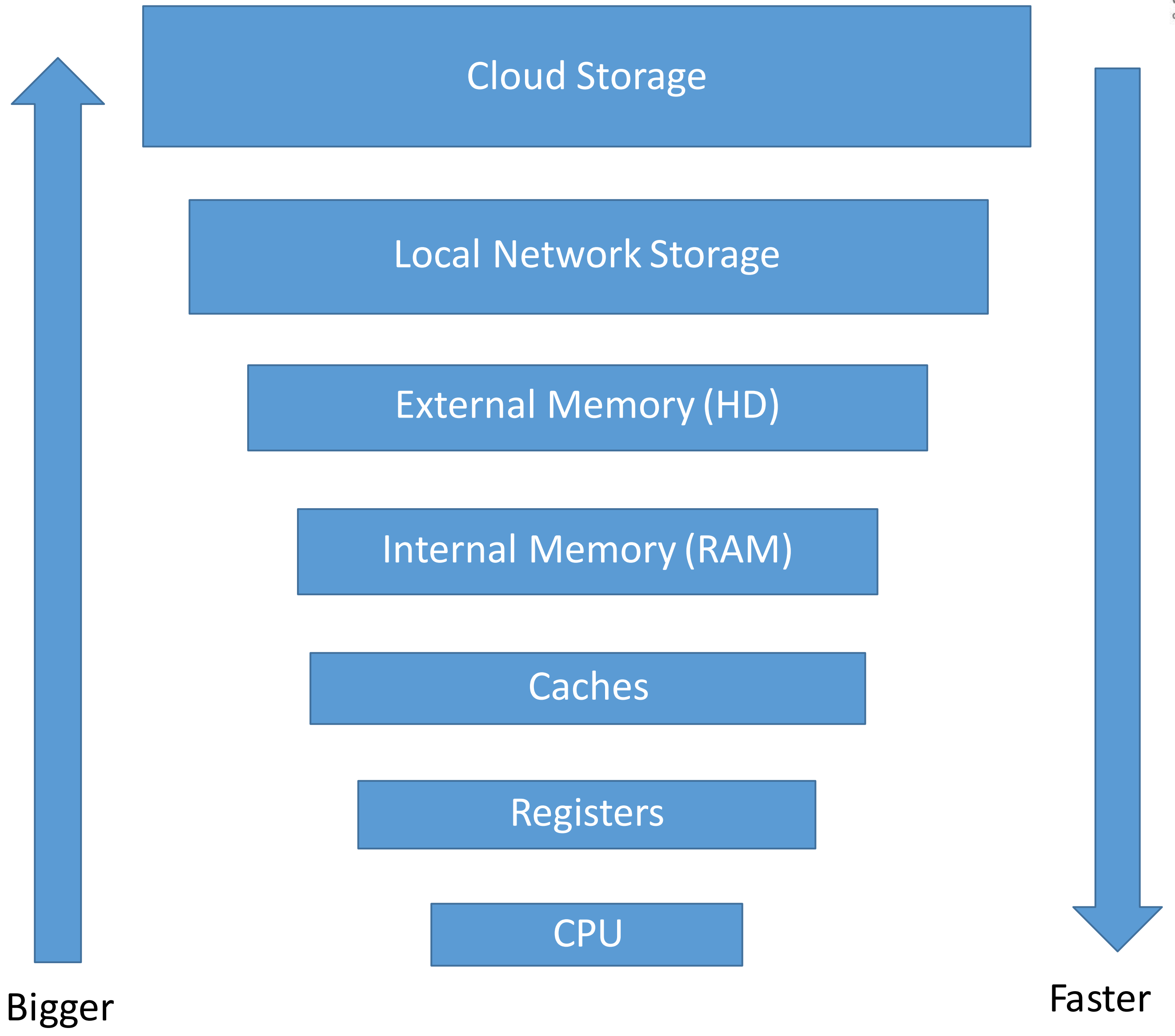
- Large datasets created by emergent applications: Internet, bioinformatics
- Example: Sequence Read Archive (SRA) of the NCBI [7]
- AI/Machine Learning techniques used to find novel biomarkers/RNA splicing in cancer
- Large data structures needed to solve the relevant problems
- For example, 30 samples of RNA-seq data involve 1.7 billion reads (strings of length 72)
- Genome/Transcriptome assembly and alignment requires:
  - Large ADTs
  - Large graphs (De Bruijn graphs) ≈ 7 billion nodes [8]
  - Large files
  - Fast algorithms

2 petabytes of data (2^{15} bases)



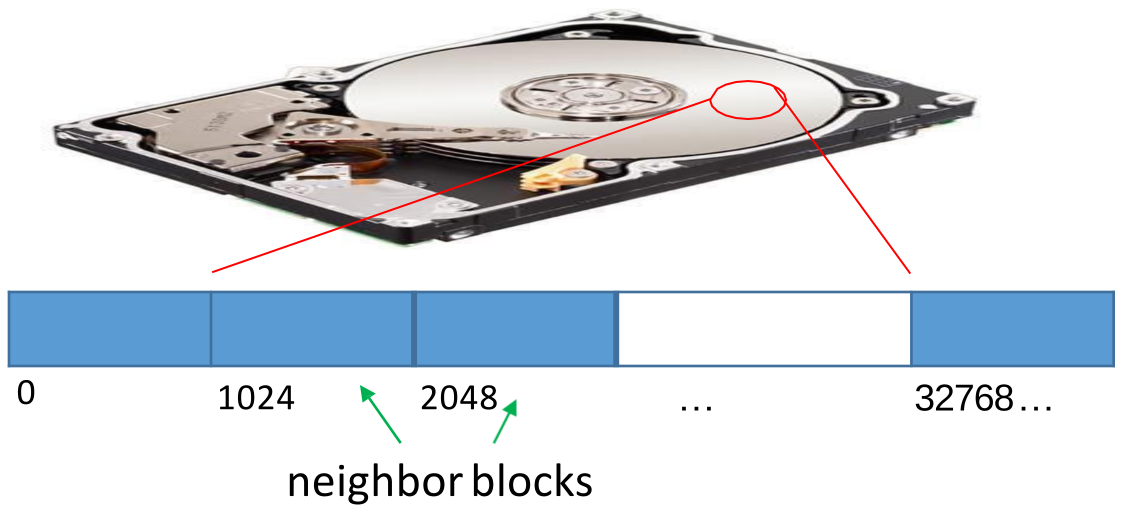
Caching – memory systems

- CPU uses internal registers for basic calculations
- Caches allow to store data temporarily
- Internal memory is larger: aka RAM or main memory
- External memory: Slower access
  - includes hard disks, DVDs, tapes
- Network storage has sower access
  - includes NAS (network attached storage), RAIDs, etc.
- Cloud storage
  - Storage server, control server
  - Offered by Google, Amazon, Dropbox and others



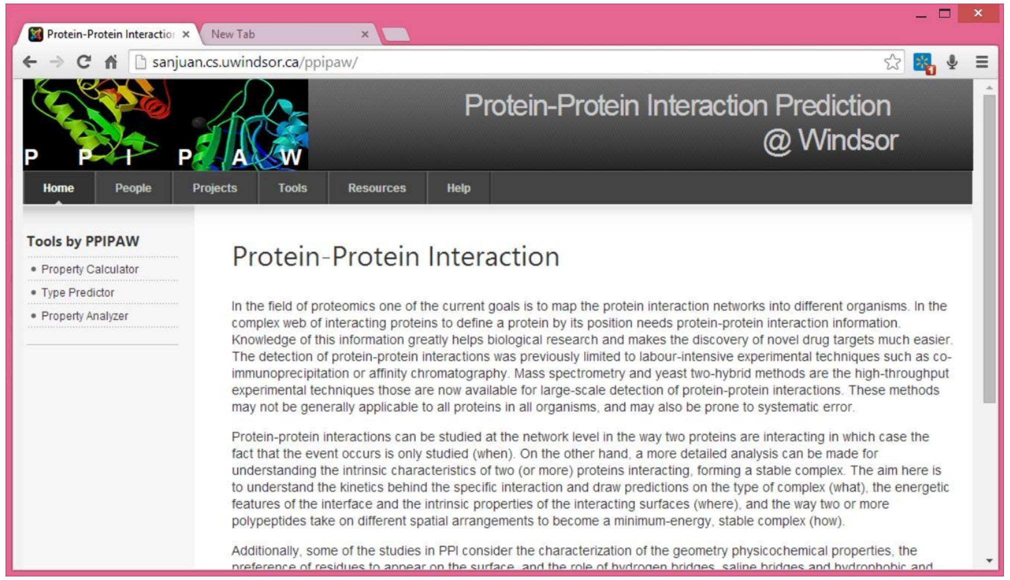
Caching and blocking

- There's a trade off between blocks in memory hierarchy
- For example, main memory is 10 to 100 times faster than secondary memory
- Fewer accesses to slower memory is desired
- Algorithms for caching solve these problems
- There aren't specific mechanisms since these algorithms are platform-dependent
  - Unlike the RAM model discussed in algorithm analysis!
  - But some general approaches exist



Caching in Web browsers

- Browsers store Web pages in a cache memory in the client's storage
- Pages are quickly retrieved from client when needed
- Main purpose
  - Reduce network traffic
  - Speed up browsing
- Fully associative cache
  - Cache and server act as "internal" and "external" memories
  - In the cache, memory has *m* slots
  - A Web page can be placed in any slot
- General caching mechanism
  - A Web page *p* is requested
  - Browser checks if *p* has changed
  - If not changed, *p* uploaded from cache
  - If changed, *p* requested from Internet and updated in cache
  - When cache becomes full, a page *p*' is removed by a page replacement scheme



Protein-Protein Interactio × New Tab ×

sanjuan.cs.uwindsor.ca/ppipaw/

P P I P A W

Protein-Protein Interaction Prediction @ Windsor

Home | People | Projects | Tools | Resources | Help

Tools by PPIPAW

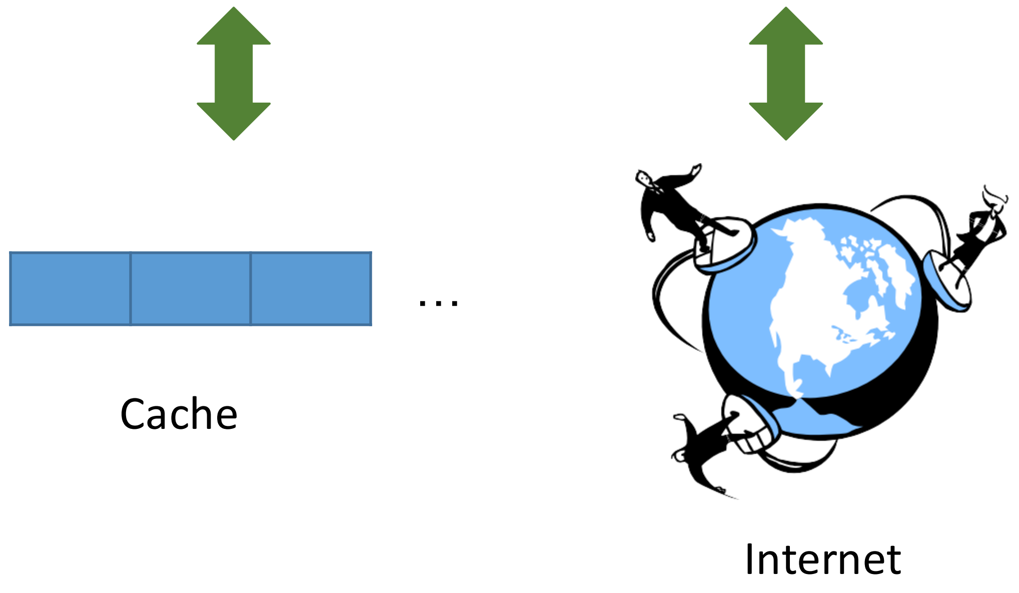
- Property Calculator
- Type Predictor
- Property Analyzer

Protein-Protein Interaction

In the field of proteomics one of the current goals is to map the protein interaction networks into different organisms. In the complex web of interacting proteins to define a protein by its position needs protein-protein interaction information. Knowledge of this information greatly helps biological research and makes the discovery of novel drug targets much easier. The detection of protein-protein interactions was previously limited to labour-intensive experimental techniques such as co-immunoprecipitation or affinity chromatography. Mass spectrometry and yeast two-hybrid methods are the high-throughput experimental techniques those are now available for large-scale detection of protein-protein interactions. These methods may not be generally applicable to all proteins in all organisms, and may also be prone to systematic error.

Protein-protein interactions can be studied at the network level in the way two proteins are interacting in which case the fact that the event occurs is only studied (when). On the other hand, a more detailed analysis can be made for understanding the intrinsic characteristics of two (or more) proteins interacting, forming a stable complex. The aim here is to understand the kinetics behind the specific interaction and draw predictions on the type of complex (what), the energetic features of the interface and the intrinsic properties of the interacting surfaces (where), and the way two or more polypeptides take on different spatial arrangements to become a minimum-energy, stable complex (how).

Additionally, some of the studies in PPI consider the characterization of the geometry physicochemical properties, the preference of residues to appear on the surface, and the role of hydrogen bridges, saline bridges and hydrophobic and

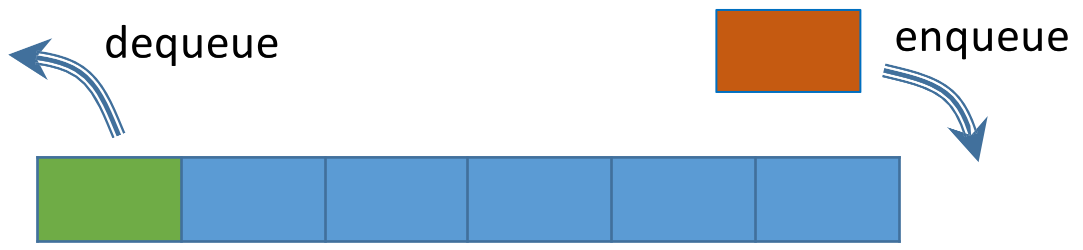


Page replacement schemes

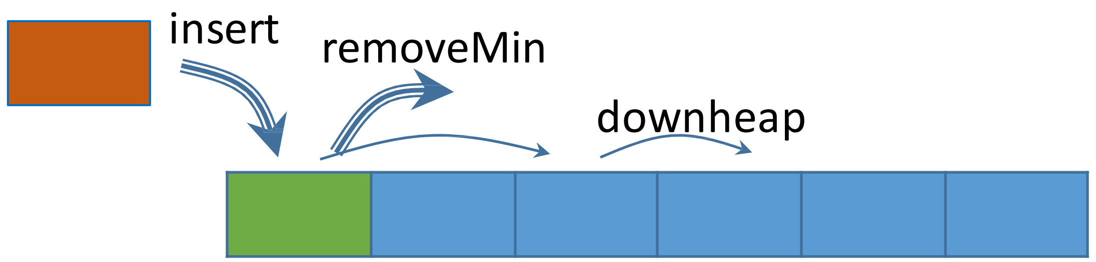
Three main schemes

- First-in, first-out (FIFO)
  - It is a simple queue mechanism
  - Insert/remove takes O(1)
  - Dequeue chooses the page to be replaced
- Least recently used (LRU)
  - Sets priorities based on number of accesses to web pages
  - Implemented via an adaptable priority queue (priorities change over time)
  - Insert/remove/replace take O(log n)
  - RemoveMin selects web page to be replaced
- Random
  - Randomly choose a web page from the cache and replace it with the new page

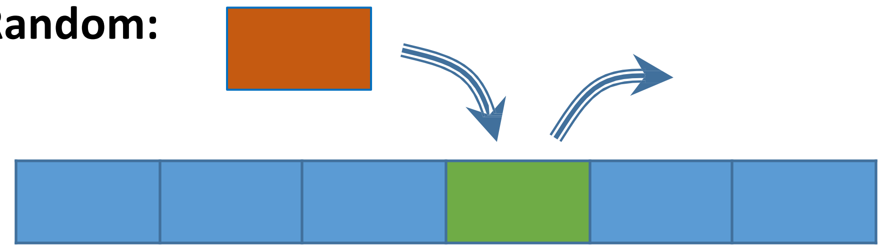
FIFO:



LRU:



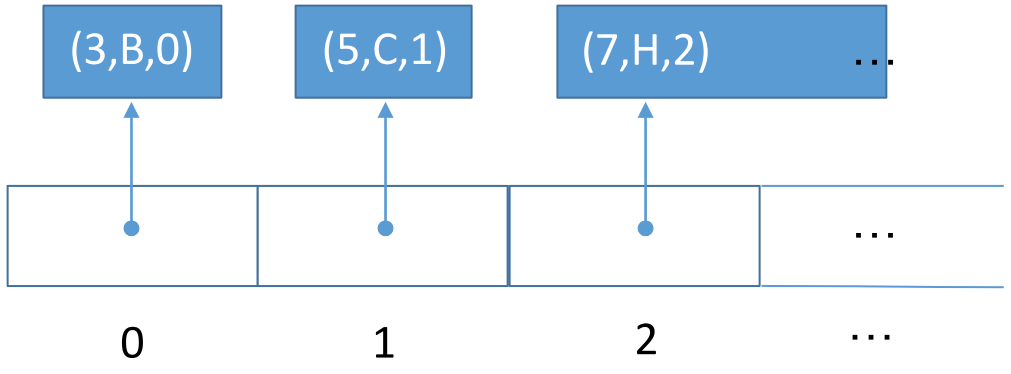
Random:



Adaptable priority queues – overview

- Extends the priority queue ADT with additional functionality
- Present in applications in which keys may change over time
- Example:
  - A Web page cache that uses number of accesses as keys
  - These keys change over time as user browses different pages
- Additional methods:
  - Find entry e
  - Remove entry e
  - Replace key k of entry e
  - Replace value v of entry e
- More details: Sec 9.5 of [2]

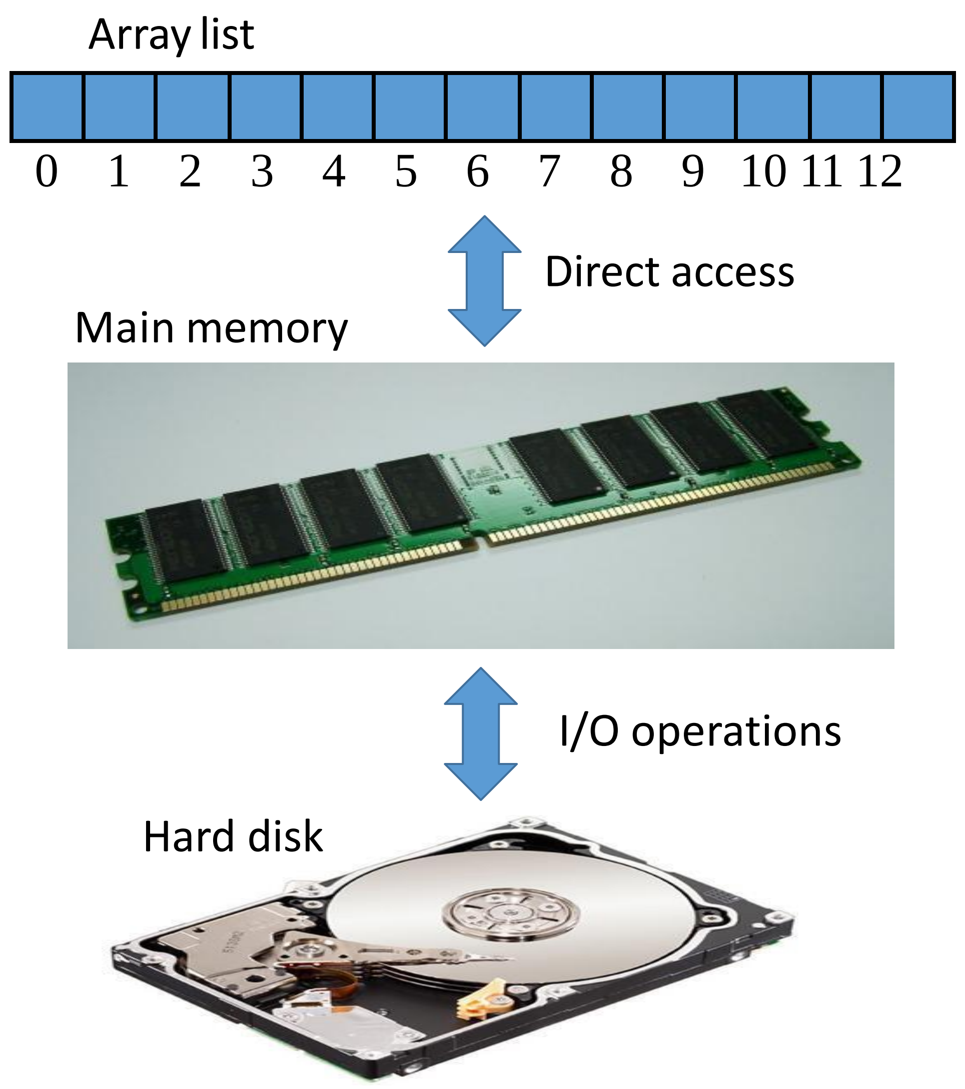
- Finding an entry
  - Can be done by keeping an array of references to entries
  - The reference can be obtained when the key is inserted
  - A look-up table is maintained:



- Remove, removeMin and replace:
  - Run in O(log n) using the look-up table
  - Must update the look-up table every time there is a change
  - Apply Downheap to update heap

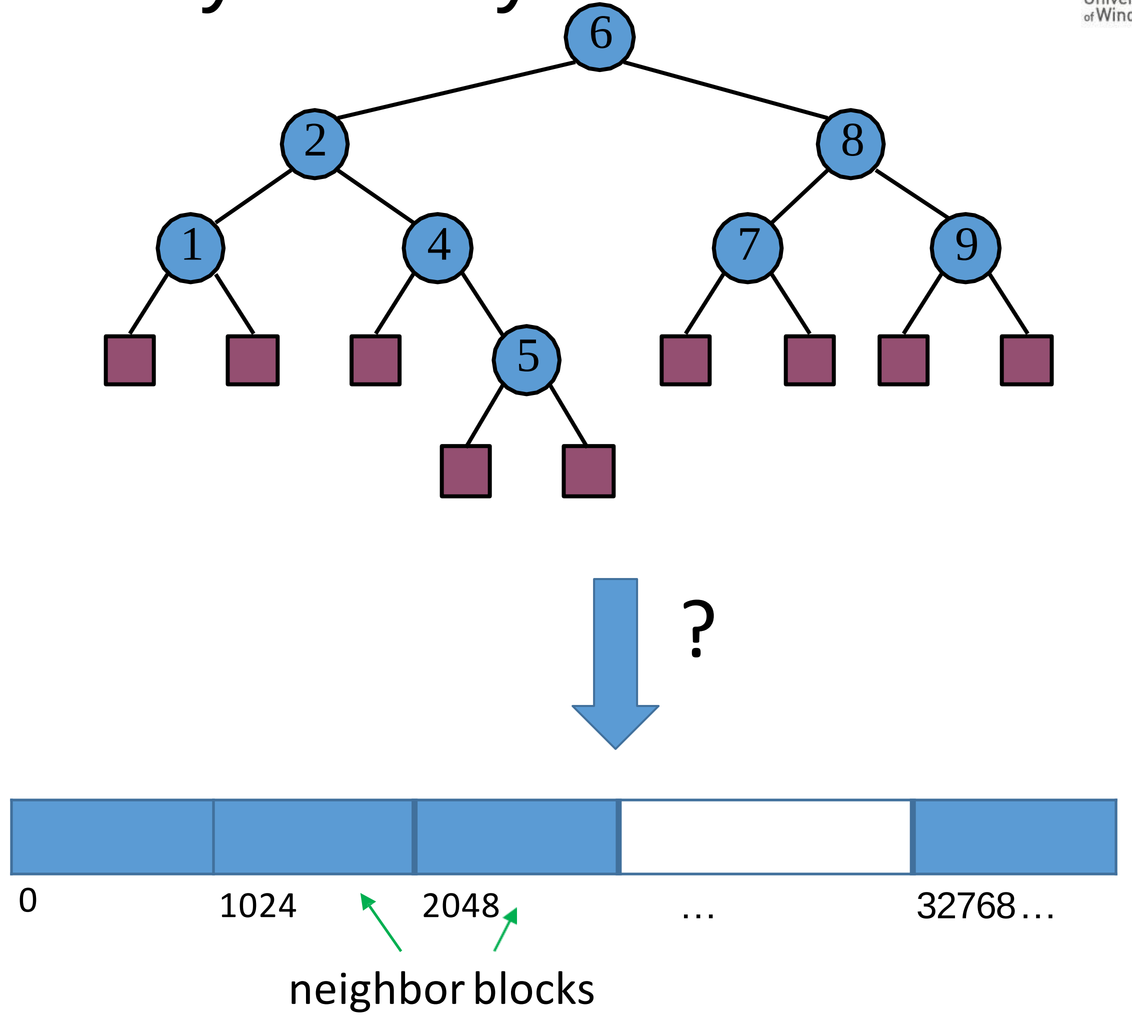
External search

- Consider the search problem in a dictionary
- Big search tree doesn’t fit in main memory
- We also use external memory
- Main factors:
  - Disk blocks: Hard disks are organized in blocks
  - Disk transfer: Some costs associated with transferring from HD to main memory
  - Costs referred to as I/O complexity
- Inefficient external memory representation
  - Store dictionary of n entries in hard disk, on an array list
  - Use blocks of size B
  - Search tasks O(n/B)
- Efficient representation:
  - B-trees
  - Extendible hashing



How do we store a large BST in secondary memory?

- BST doesn't fit in main memory
- We can then use external memory
- Naïve approach
  - Divide the tree into pieces (groups of nodes)
  - Store these pieces into blocks
  - Which nodes do we place in the same block?
  - What happens when we perform a search?
  - It's an inefficient representation
- Efficient representation:
  - B-trees

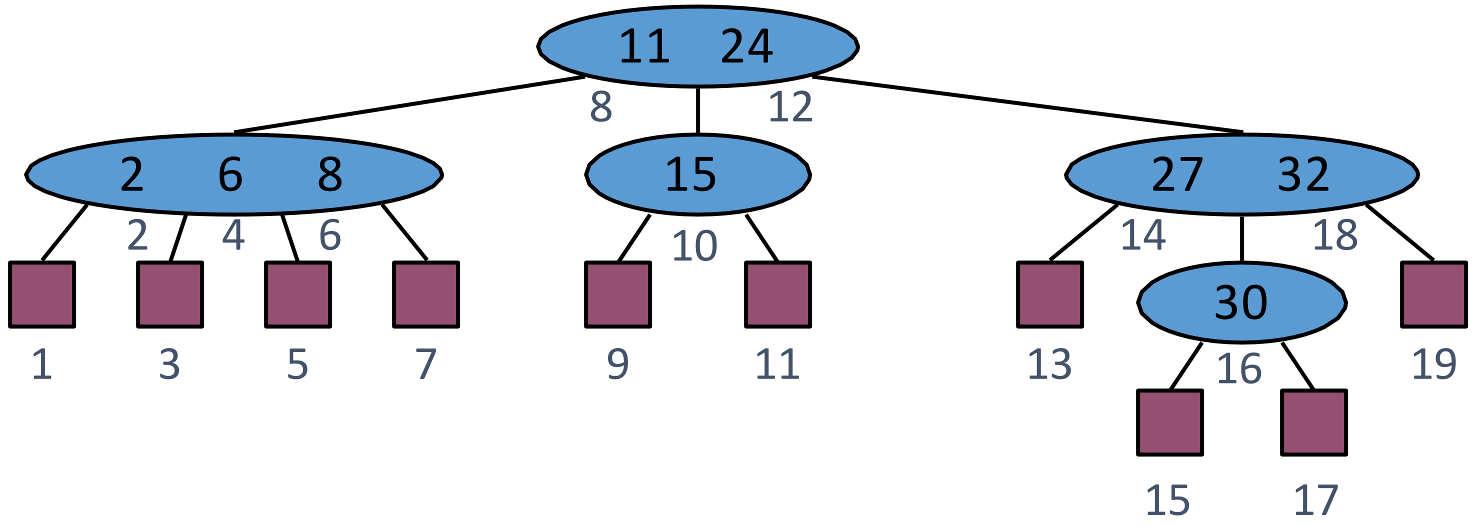


Multi-way search trees – review

- A multi-way search tree is an ordered tree such that
  - Each internal node has at least two children and stores $d-1$ key-element items (k_i, o_i), where d is the number of children
  - For a node with children $v_1\ v_2 \ldots v_d$ storing keys $k_1\ k_2 \ldots k_{d-1}$
    - keys in the subtree of v_1 are less than k_1
    - keys in the subtree of v_i are between k_{i-1} and k_i $(i = 2, \ldots, d-1)$
    - keys in the subtree of v_d are greater than k_{d-1}
- The height of a multi-way search tree is O(log n)

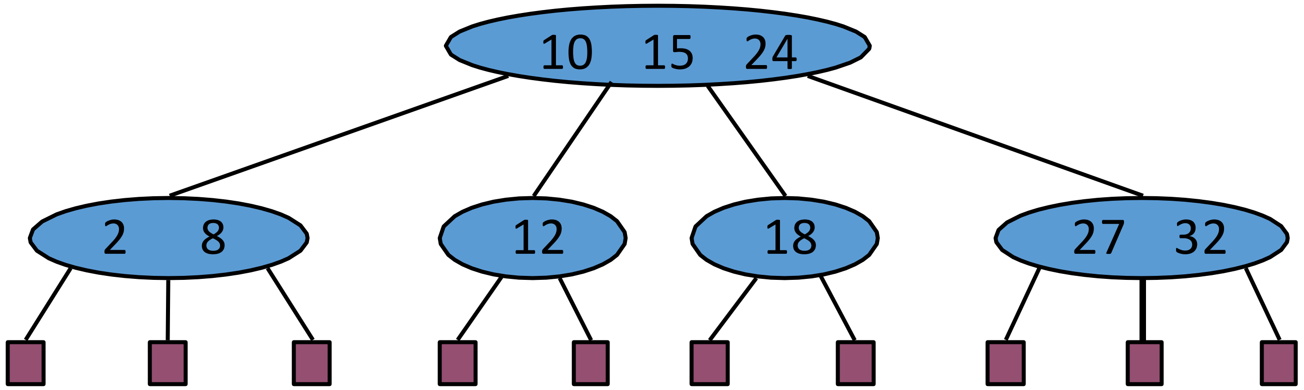
Inorder traversal:

- (k_i, o_i) of node v is visited between the recursive traversals of the subtrees of v rooted at children v_i and v_{i+1}
- The keys of the multi-way search tree are visited in increasing order



(*a*,*b*) Trees

- An (*a*,*b*) tree is a multi-way search tree with the following properties:
  - Node-Size Property: every internal node has at least *a* children and at most *b* children
  - Depth Property: all the external nodes have the same depth
- Depending on the number of children, an internal node of an (*a*,*b*) tree is called a 2-node, 3-node, 4-node, and so on
- Example: A (2,4) tree



(a,b) Trees properties and operations

- Height: An (*a*,*b*) tree storing ***n*** items has height $\Omega(\log n/\log b)$
- Let f(b) be the time for performing a search operation in a list (e.g., an array list)
- Search in an (*a*,*b*) tree with ***n*** items takes $O([f(b)/\log a]/\log n)$ time
- Let $g(b)$ be the time for performing split and fusion operations (defined for a multi-way search tree)
- Insertions and removals in/from an (*a*,*b*) tree with ***n*** items takes $O([g(b)/\log a]/\log n)$ time

B-trees

- A B-tree is the best-known method used to store a dictionary using external memory
- It uses both main (RAM) and external (hard disk)

Definition

- A B-tree of order d is an (a,b) tree in which $a = \lceil d/2 \rceil$ and $b = d$
- Choice of d depends on the size of block B
  - d-1 keys in internal nodes must fit in a block of size B
  - Ideally, a block of size B should contain d - 1 keys
- Then, if B (and hence d) is constant, $f(b)$ and $g(b)$ are both O(1)
- In general, complexity is also given in terms of both n and B

- Insertion of k:
  - Do a search of k
  - Once a leaf node v is found, insert k in v
  - If v has more than d children,
    split the parent into two nodes with
    $\lfloor (d+1)/2 \rfloor$ and $\lceil (d+1)/2 \rceil$ children
  - If parent "overflows" continue with grandparent, and so on
  - If root overflows, split it into two nodes
- Removals
  - Similar to insertion, except that nodes may "underflow" (i.e., has less than a-1 keys)

Complexity:

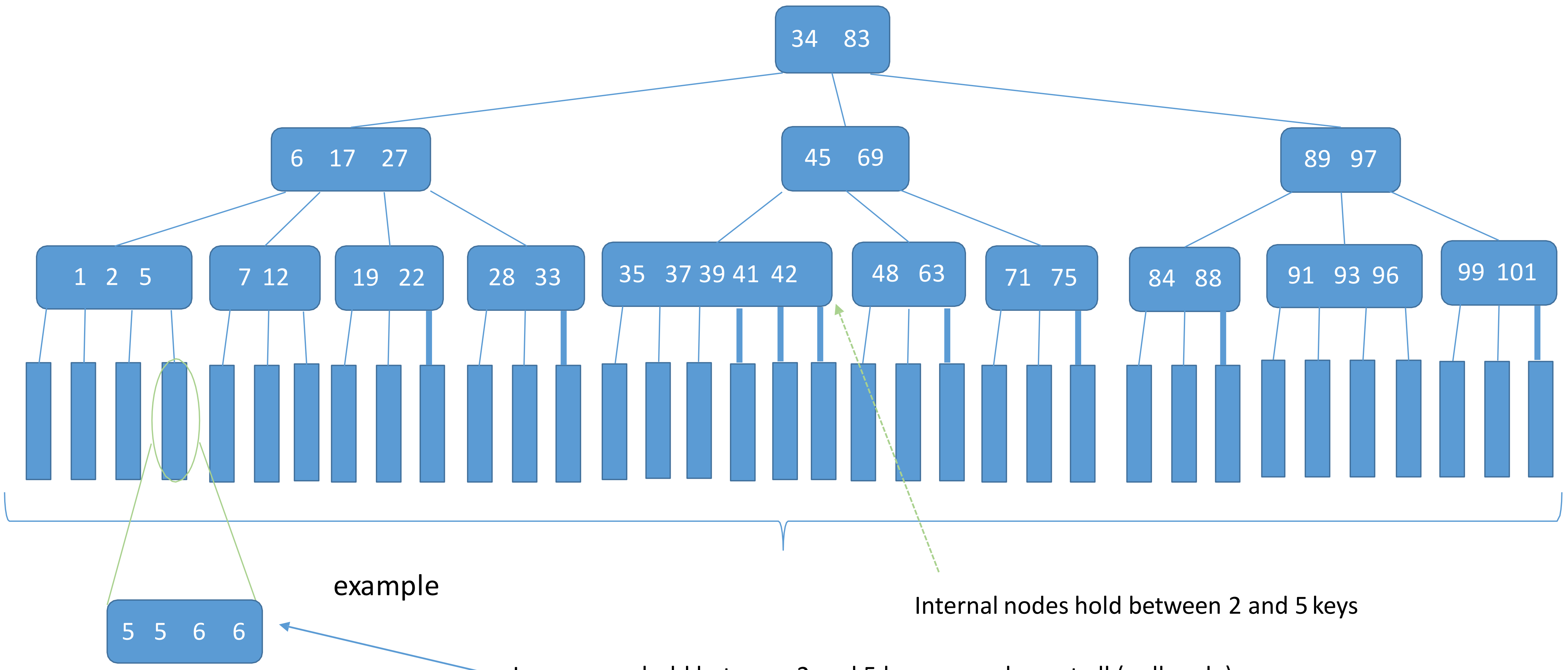
- Searches, insertions and removals
  - $O(\log_B n)$ I/O complexity
  - Uses $O(n/B)$ blocks

Example

B-tree of order 6

Max # of children= 6

Min# of children = 3



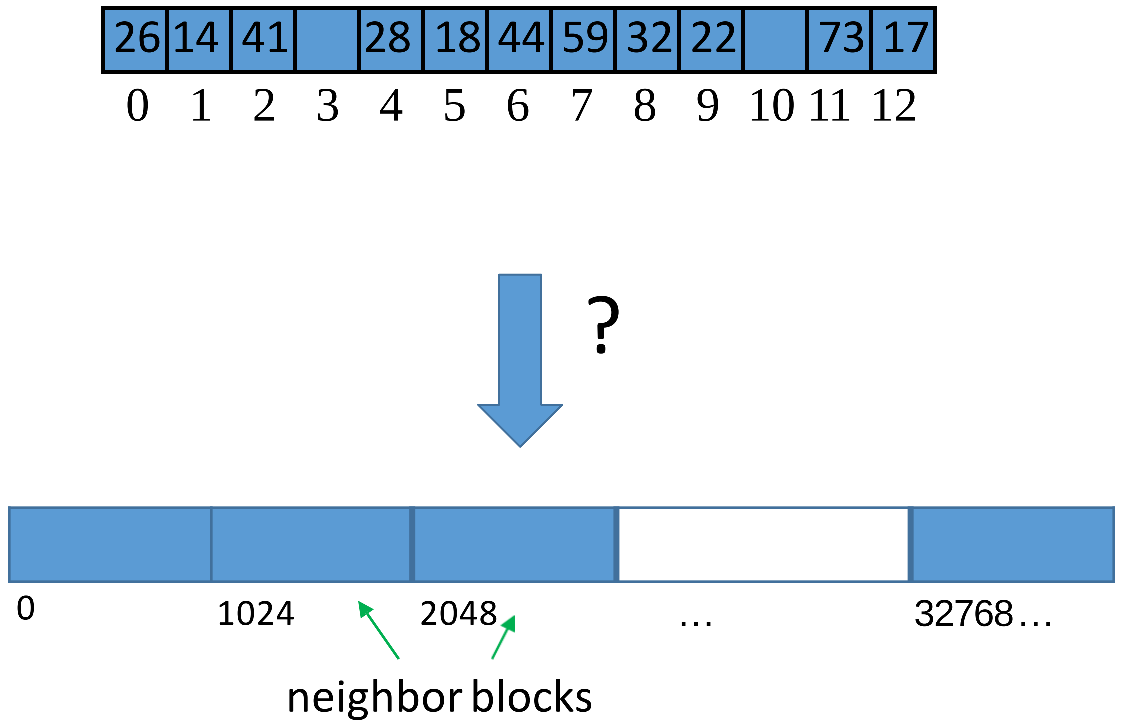
example

Internal nodes hold between 2 and 5 keys

Leaves may hold between 2 and 5 keys, or no keys at all (null node)

How do we store a large Hash Table in secondary memory?

- Hash table doesn't fit in main memory
- We can then use external memory
- Naïve approach
  - Divide the Hash table into pieces (groups of neighbor keys)
  - Store these pieces into blocks
  - Which keys do we place in the same block?
  - What happens when we perform a search/insertion?
  - What if we use quadratic probing?
  - It's an inefficient representation
- Efficient representation:
  - Extendible Hash Tables

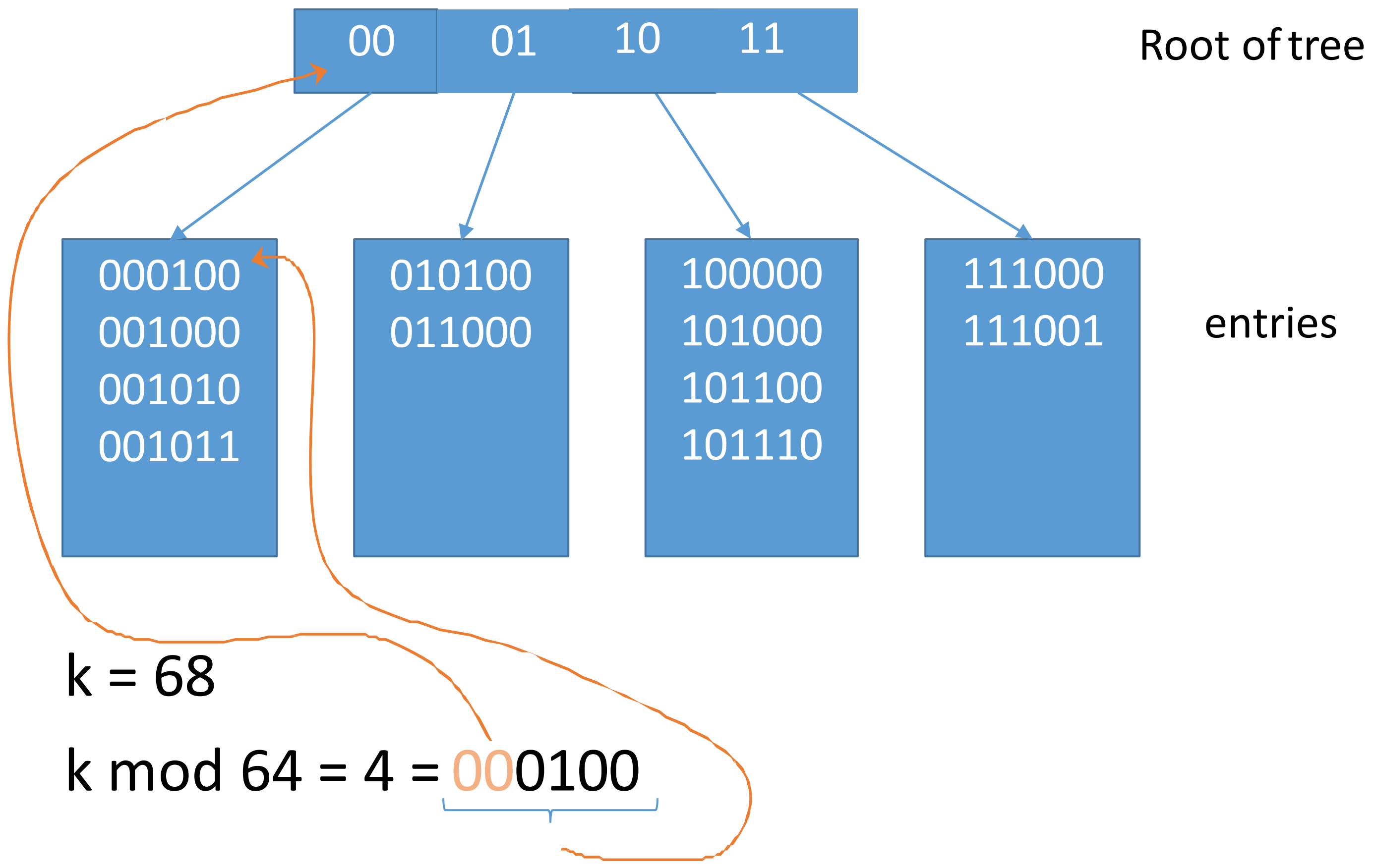


Extendible hashing

- As in B-trees, we allow the algorithm to use blocks of the hard disk
- B = size of block
- M = max number of blocks
- N = size of hash table
- Determine hash function:
  - h(k) = k mod N
- Hash table composed of
  - Root of the "tree": Contains M entries of $\log_2 M$ bits each
  - Blocks: Each block contains B entries of $\log_2 N$ bits each

Example:

M = 4, B = 4, N = 64, h(k) = k mod 64

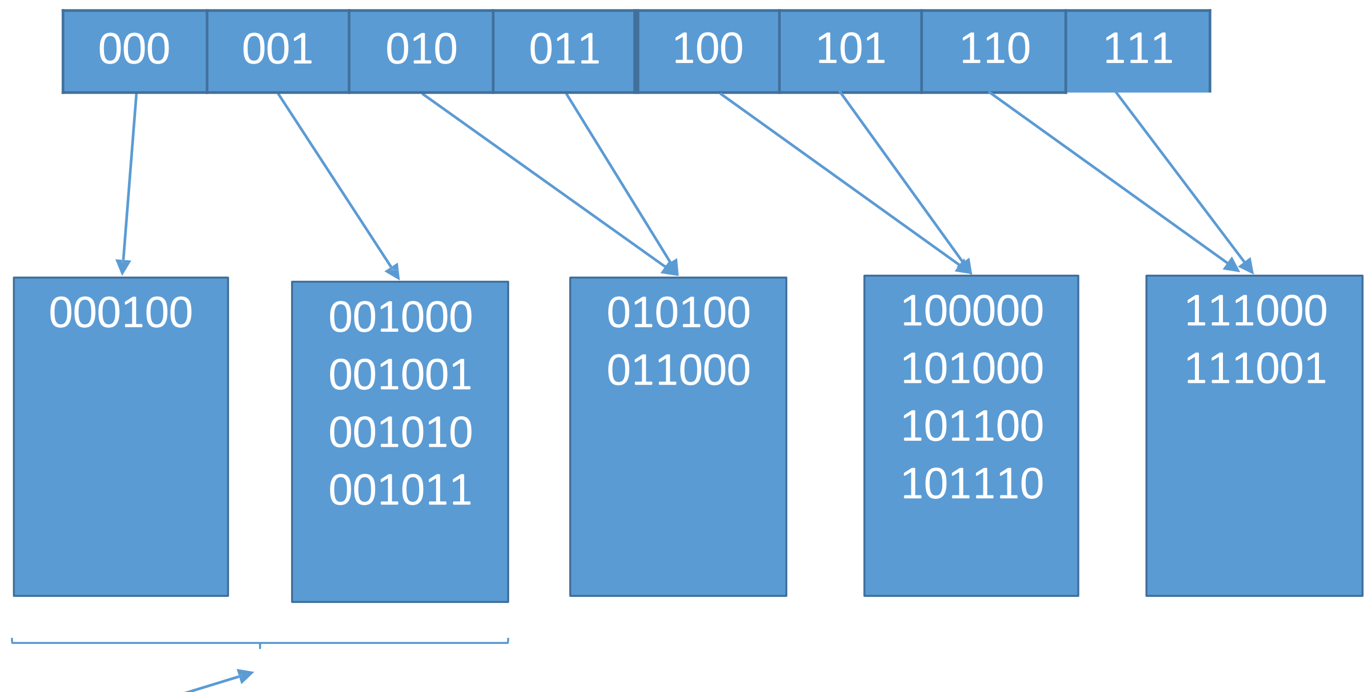


k = 68

k mod 64 = 4 = $\underbrace{000100}$

Extendible hashing - insertion

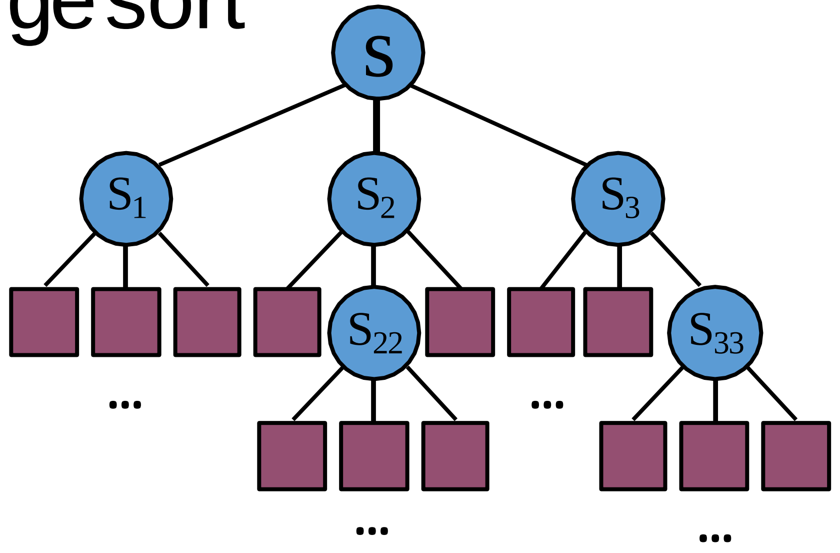
- Insertion may require splitting a leaf, if full
- Root may have to be expanded
- M changes dynamically
- Example: Inserting k = 137
- k mod 64 = 9 = 001001
- First block will become full
- Root of the tree expanded to M = 8 entries
- First block split into two blocks
- Further details: Sec. 9 of [3]



Performance:

- Some leaves may be split many times, while others may not
- Chances are that the “tree” will be balanced if keys are distributed “uniformly”
- Expected number of leaves = (n/M) $\log_2 e$

External memory sorting – Multiway merge sort



- Sorting an array list that doesn't fit in main memory
- Can be done by Multiway Merge sort
- It's a generalization of Mergesort
- Uses the principles of divide an conquer
  - Divide: divide the input list $\boldsymbol{S}$ into d disjoint sub-lists $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_d$
  - Recur: sort sub-lists $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_d$ recursively
  - Conquer: combine the sorted versions of $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_d$ into a sorted $\boldsymbol{S}$
- The key of the algorithm is Multiway merge

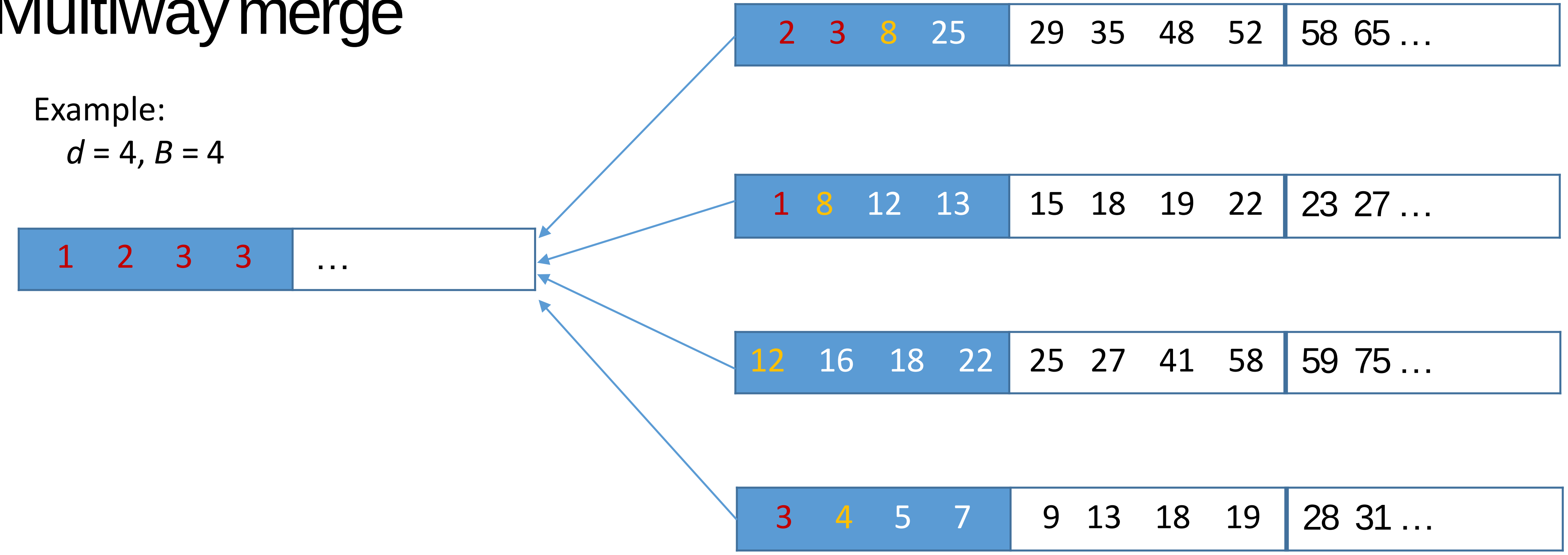
- Let B be the size of the block and M be the size of the internal memory
- Assume merge performs O(n/B) disk transfers
- Then, Multiway Merge sort will run in

$$O((n/B)\log(n/B)\ log\ d)$$

- Choosing d to be $\Theta(n/B)$ will lead to a low number of block transfers
- Best choice is $d = (M/\text{B}) - 1$

Multiway merge

Example:
$d = 4$, $B = 4$



- Blue blocks are allocated in main memory
- Red numbers have already been merged
- White blocks reside in hard disk
- White numbers are in memory and are the next ones to be merged

- Finding the minimum of the *d* lists is time consuming: O(*d*)
- Significant improvements can be made using a priority queue and *d*+1 lists (See Sec. 7.12 of [3])

Java classes for practice

- B-trees
  - Class Btree implements a B-tree
  - Uses a parameter, M, which is even
  - Max number of children is M-1
  - B-tree nodes are not saved in the disk but in main memory arrays
  - It's good for practicing the main concepts of B-trees
  - Entries in the tree are composed of a key and a value
  - A key is a comparable object
  - A value is an object
- Link:
  - http://algs4.cs.princeton.edu/code/

- Multiway merge
  - Implements the Multiway merge algorithm using a priority queue
  - Reads the sorted inputs from files (one file for each list)
  - Outputs the merged (sorted) list in the standard output
  - Keys in the priority queue are of any type as long as they are comparable
  - Input files have to be already sorted
- Link:
  - http://algs4.cs.princeton.edu/code/

Review and Further Reading

- Memory management
  - Ch. 15 of [2]
- Adaptable priority queues
  - Sec 9.5 of [2]
- B-trees
  - Sec. 14.1.2 of [1], Sec. 15.3 of [2], Sec. 4.7 of [3], Ch. 6 of [4]
- Multiway Mergesort
  - Sec. 14.1.3 of [1], Sec. 15.4 of [2], Sec. 7.12 of [3]
- Java Virtual Machine
  - Sec. 15.1 of [2]
  - Developer's documentation: http://docs.oracle.com/javase/specs/jvms/se8/html/

References

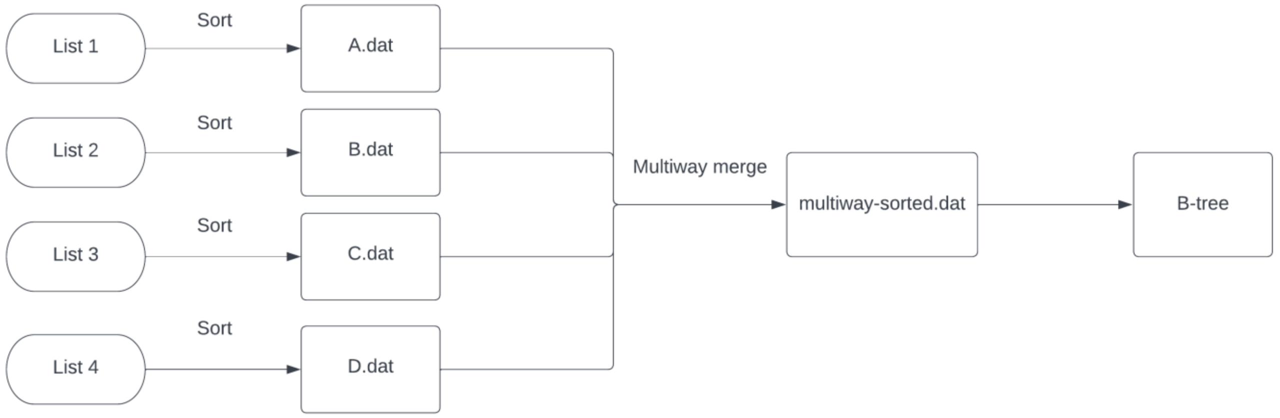
1. Algorithm design by M. Goodrich and R. Tamassia, Wiley, 2001.
2. Data Structures and Algorithms in Java, 6th Edition, by M. Goodrich and R. Tamassia, Wiley, 2014.
3. Data Structures and Algorithm Analysis in Java, 3rd Edition, by M. Weiss, Addison-Wesley, 2012.
4. Algorithms, 4th Edition, by R. Sedgewick, Addison-Wesley, 2011.
5. Java documentation:
   - http://docs.oracle.com/javase/8/docs/api/overview-summary.html
6. Data Growth, Business Opportunities, and the IT Imperatives:
   - http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm
7. Sequence read archive (SRA)
   - http://www.ncbi.nlm.nih.gov/sra
8. Parallelized short read assembly of large genomes using de Bruijn graphs, by Y. Liu et al. BMC Bioinformatics 2011, 12:354

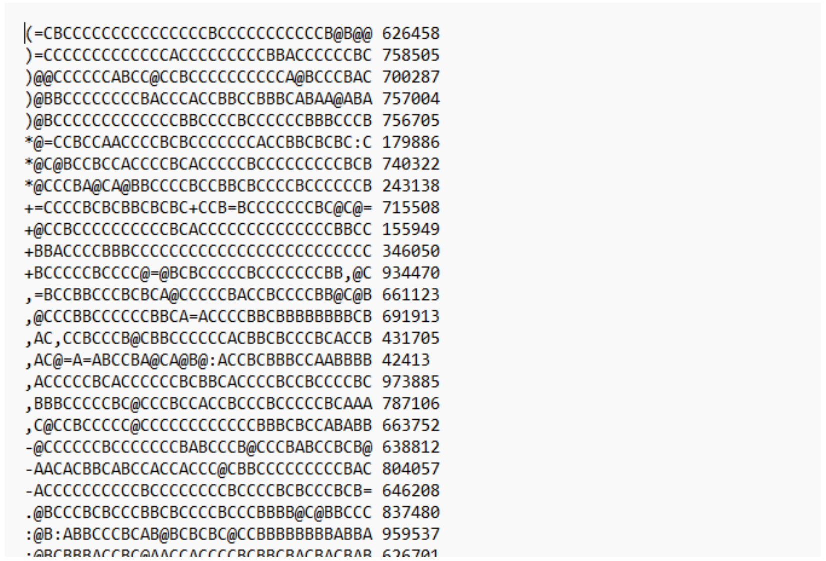
Lab – Practice

1. Use the Multiway.java and Btree.java implementations provided in the source code.
   - Create 4 lists of 250,000 random strings each (length of 10), sort them separately using Quicksort and save them in 4 different files (called A.dat, B.dat, C.dat, and D.dat).
   - Use Multiway.java to merge the files and save them into a separate file (called multiway-sorted.dat).
   - Using that file as input, create a B-tree with the whole list. For each entry, use the string as the key and "0" as the value.
   - Print the tree in in-order traversal.
2. Consider the one million RNA-seq reads given in the files called Chip-seq-reads-1M.txt
   - Create a B-tree and insert all the reads from Chip-seq-reads-1M.dat as they appear in the file. List the B-tree in in-order traversal and save the output all keys in a file (called B-tree.dat).

Lab – Practice (Cont'd)

Flowchart for task 1





```
(=CBCCCCCCCCCCCCCCCBCCCCCCCCCCCB@B@@ 626458
)=CCCCCCCCCCCCCACCCCCCCCCBBACCCCCCBC 758505
)@@CCCCCCABCC@CCBCCCCCCCCCCCA@BCCCBAC 700287
)@BBCCCCCCCCBACCCACCBBCCBBBCABAA@ABA 757004
)@BCCCCCCCCCCCCCBBCCCCBCCCCCCBBBCCCB 756705
*@=CCBCCAACCCCBCBCCCCCCCACCBBCBCBC:C 179886
*@C@BCCBCCACCCCBCACCCCCBCCCCCCCCCBCB 740322
*@CCCBA@CA@BBCCCCBCCBBCBCCCCBCCCCCCB 243138
+=CCCCBCBCBBCBCBC+CCB=BCCCCCCCBC@C@= 715508
+@CCBCCCCCCCCCCBCACCCCCCCCCCCCCCBBCC 155949
+BBACCCCBBBCCCCCCCCCCCCCCCCCCCCCCCCC 346050
+BCCCCCBCCCC@=@BCBCCCCCBCCCCCCCBB,@C 934470
,=BCCBBCCCBCBCA@CCCCCBACCBCCCCBB@C@B 661123
,@CCCBBCCCCCCBBCA=ACCCCBBCBBBBBBBBCB 691913
,AC,CCBCCCB@CBBCCCCCCACBBCBCCCBCACCB 431705
,AC@=A=ABCCBA@CA@B@:ACCBCBBBCCAABBBB 42413
,ACCCCCBCACCCCCCBCBBCACCCCBCCBCCCCBC 973885
,BBBCCCCCBC@CCCBCCACCBCCCBCCCCCBCAAA 787106
,C@CCBCCCCC@CCCCCCCCCCCCBBBCBCCABABB 663752
-@CCCCCCBCCCCCCCBABCCCB@CCCBABCCBCB@ 638812
-AACACBBCABCCACCACCC@CBBCCCCCCCCCBAC 804057
-ACCCCCCCCCCBCCCCCCCCBCCCCBCBCCCBCB= 646208
.@BCCCBCBCCCBBCBCCCCBCCCBBBB@C@BBCCC 837480
:@B:ABBCCCBCAB@BCBCBC@CCBBBBBBBBABBA 959537
:@BCBBBACCBC@AACCACCCCBCBBCBACBACBAB 626701
```

B-tree.dat

Exercises

1. Discuss the main problems encountered as a result of Big Data and the challenges for the near future.
2. What is fragmentation? Design an algorithm that implements the Worst-fit approach using a priority queue.
3. Write an algorithm for the LRU page replacement approach.
4. For the Web page replacement mechanism, can you use another ADT to speed up the LRU approach? Discuss it in detail and give the reasons for why you use such an ADT.
5. Define a multiway search tree.
6. Suppose a multiway search tree T in which each internal node has at least 4 and at most 7 children. What are the values of *a* and *b* for a valid (*a,b*) tree? What about the order *d* for a B-tree?
7. Start with an empty B-tree of degree 4. Insert the following keys in the order they appear: 73, 45, 12, 19, 85, 34, 43, 11. Show the resulting tree.
8. Consider the (a,b) search tree. Write algorithms for insertion and removals.
9. Do the same as #8 but for a B-tree. Also, write an algorithm for searching a key.
10. Suppose you are storing 32-bit keys in a B-tree, and that each block in the hard disk contains 1024 bits. What is the best value for the degree d of the B-tree?
11. Show that the height of an (a,b) tree that stores n items is $\Omega(\log n/\log b)$.
12. *Implement the insertion algorithm for extendible hashing. Discuss its complexity. Do the same for removing keys.
13. For an extendible hashing data structure, how will you choose the max number of blocks and the size of each block?
14. Consider Multiway merge as discussed in class. Write an algorithm that uses a priority queue to perform each step of merge in O(log d).
15. *Design an efficient external-memory algorithm that determines whether an array of *n* integers contains a value occurring more than *n*/2 times.
16. *Implement the B-tree ADT that uses a block of size 1024 and stores integer keys. Test the number of disk transfers for a sequence of 1000 insertions.