Notes lecture 11

Cuckoo hashing, Perfect hashing, Linked hash table

Trees, Binary trees

*Cuckoo hashing:*

* Structure:
  + 2 tables (same size)
  + for each
  + Each of them has his own hash function
* For each elem 🡪 compute 2 positions (h1(elem) from the first table and h2(elem) from the second one) – it is guaranteed that the elem is on one of these 2 positions
* Insert e:
  + Compute h1(e)
  + If position h1(e) in first table is empty
    - Place e there
  + Else
    - Kick out the elem there (kicked\_e)
    - Place e there
    - If position h2(kicked\_e) in second table is empty
      * Place kicked\_e there
    - Else
      * Kick out the elem there
      * Place kicked\_e there
      * REPEAT
  + If we obtain a cycle 🡺 *resize and rehash*

*Perfect hashing:*

* Idea:
  + Use a hash table of size N (primary hash table).
  + each element of the hash table is another hash table
  + Make the secondary hash table of size , where is the number of elements from this hash table
  + Each secondary hash table will be constructed with a different hash function, and will be reconstructed until it is collision free
* total space needed for the secondary hash tables is at most 2N
* multiple hash functions 🡺 we use universal hashing
  + p - prime number larger than the largest possible key
  + The universal hash function family H can be defined as:
  + Choose a and b randomly for each hash function
* We need to have static keys: once the table is built, no new elements can be added

*Linked hash table:*

* For iteration: preserve the order in which the elements were inserted
* Besides being stored in the hash table, each element is part of a linked listA close-up of a white envelope

  Description automatically generated

*Trees:*

* ordered tree = the order of the children is well defined and relevant
* height of the tree = longest path from the root to a leaf (number of edges)
* left-child right-sibling representation:
  + info
  + address parent node
  + address of the leftmost child
  + address of the right sibling

*Binary trees:*

* have at most two children
* *full* = every internal node has exactly two children
* *complete* = full + all leaves are one the same level
* *almost complete* = complete binary tree except for the last level, where nodes are completed from left to right (binary heap - structure)
* *degenerate* = every internal node has exactly one child (chain of nodes)
* *balanced* = the difference between the height of the left and right subtrees is at most 1 (for every node from the tree)