HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

C BASIC

HASH TABLE

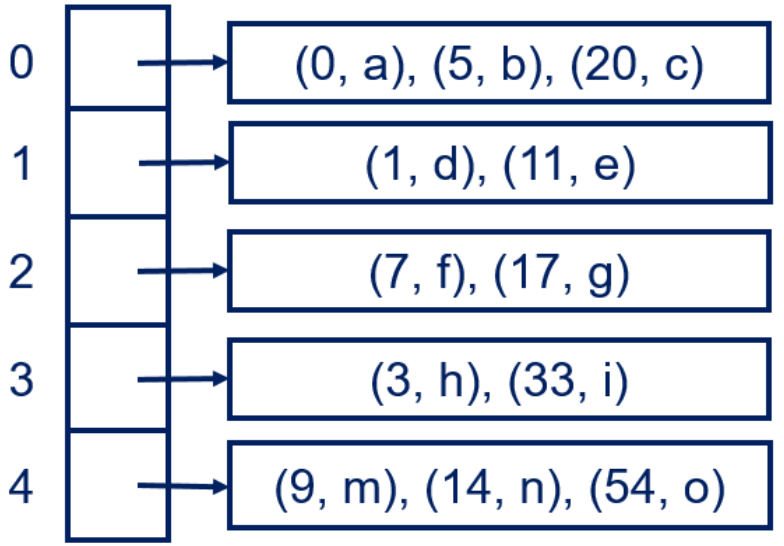
ONE LOVE. ONE FUTURE.

Content

- Hash table and Hash function
- Hashing a string (P.06.15.01)
- Storing and searching strings (P.06.15.02)
- Counting word frequencies in a document (P.06.15.03)

HASH TABLE AND HASH FUNCTION

- A hash table: A data structure to store objects, each object has a key to search effectively
  - Has slots numbering with 0, 1, 2, . . ., m-1 (m is a parameter)
- A hash function h(k) receives input as a key of an object and returns the corresponding index in the hash table (the value of $h(k)$ is called the hash code of k)
- Key collision: Two different keys k_1 and k_2 have the same hash code $h(k_1) = h(k_2)$
  - Chaining method: objects having the same hash code are stored in a the same chain (group)



HASHING STRINGS (P.06.15.01)

- Given a positive integer *m*, a string *s*[1..*k*] (characters are from: a, b, …, z) : Having hash code calculated by:

$$h(s[1..k]) = (s[1]*256^{k-1} + s[2]*256^{k-2} + \ldots + s[k]*256^{0}) \bmod m$$

- Data
  - Line 1: Two positive integers *n* and *m* (1 <= *n*, *m* <= 1000000)
  - *Next n* lines, each line a string (Characters are from a, b, …, z)
- Result
  - Each line a hash code corresponding with a line from input data

stdin	stdout
4 1000 a ab abc abcd	97 930 179 924

HASHING STRINGS - PSEUDOCODE

- Horner flow:

$h(s[1..k]) = s[1]*256^{k-1} + s[2]*256^{k-2} + \ldots + s[k]*256^{0}$

$= 256*(s[1]*256^{k-2} + s[2]*256^{k-3} + \ldots + s[k-1]*256^{0}) + s[k]$

$= 256*h(s[1..k-1]) + s[k]$

```
h(s[1..k], m) {
  code = 0
  for i = 1 to k do {
    code = (code * 256 + s[i]) mod m;
  }
  return code;
}
```

STORING AND SEARCHING STRINGS (P.06.15.02)

- A data set D includes n keys $k_1, k_2, \ldots, k_n$ (a key is a string of characters with length from 1 to 50). Perform a series of operations consisting of 1 of the following 2 types:
  - find k: returns 1 if key k exists in D and returns 0, otherwise
  - insert k: inserts keys k and D (if k does not already exist in D) and returns 1; and returns 0 if k exists in D
- Data
  - Contains two information blocks
    - Block 1: Contains lines, each line is for a key. Block 1 ends with a line "*"
    - Block 2: Contain lines, each line is an operation of one of the above two types. Block 2 ends with a line "***"
- Result
  - Each line is the result of the corresponding operation to input data

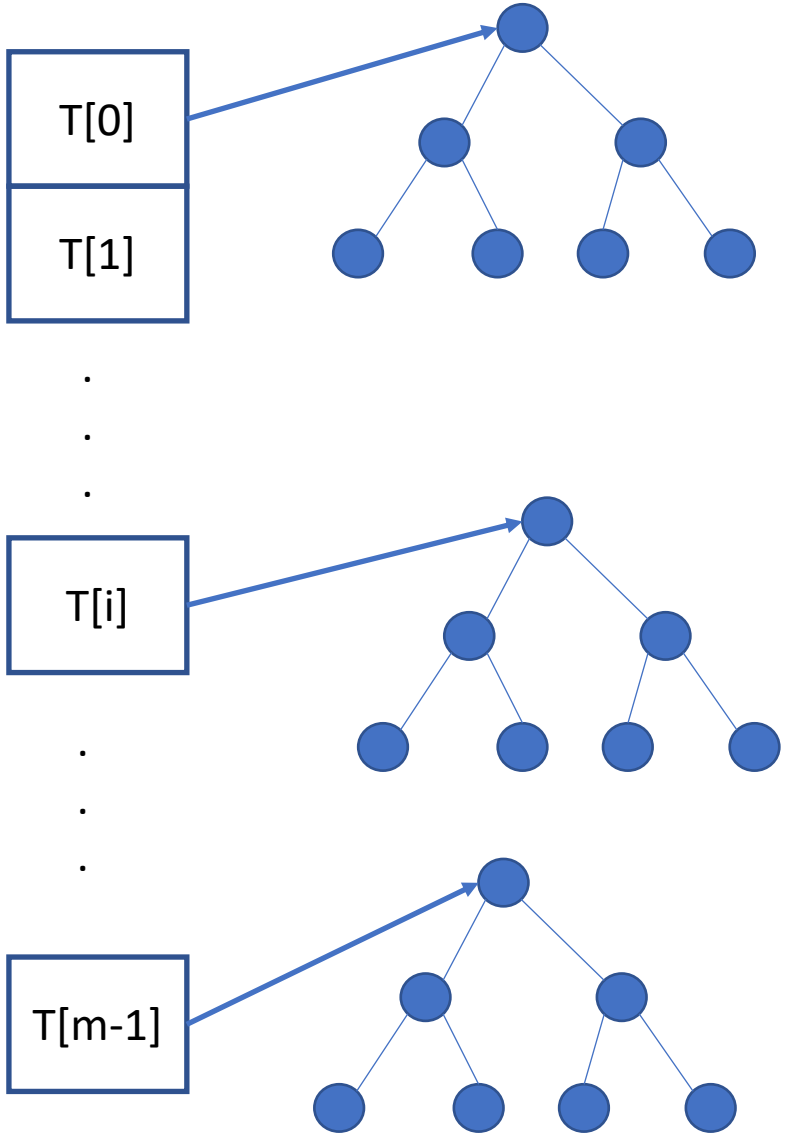
stdin	stdout
computer university school technology phone * find school find book insert book find algorithm find book insert book ***	1 0 1 0 1 0

STORING AND SEARCHING STRINGS - PSEUDOCODE

- Use a hash table and binary search trees
  - The hash table *T* hash m elements: element i^{th} of table *T*[*i*] is a pointer to a binary search tree
  - Data structure for nodes in binary search trees

```
struct Node {
   key; //
   leftChild;
   rightChild;
}
```

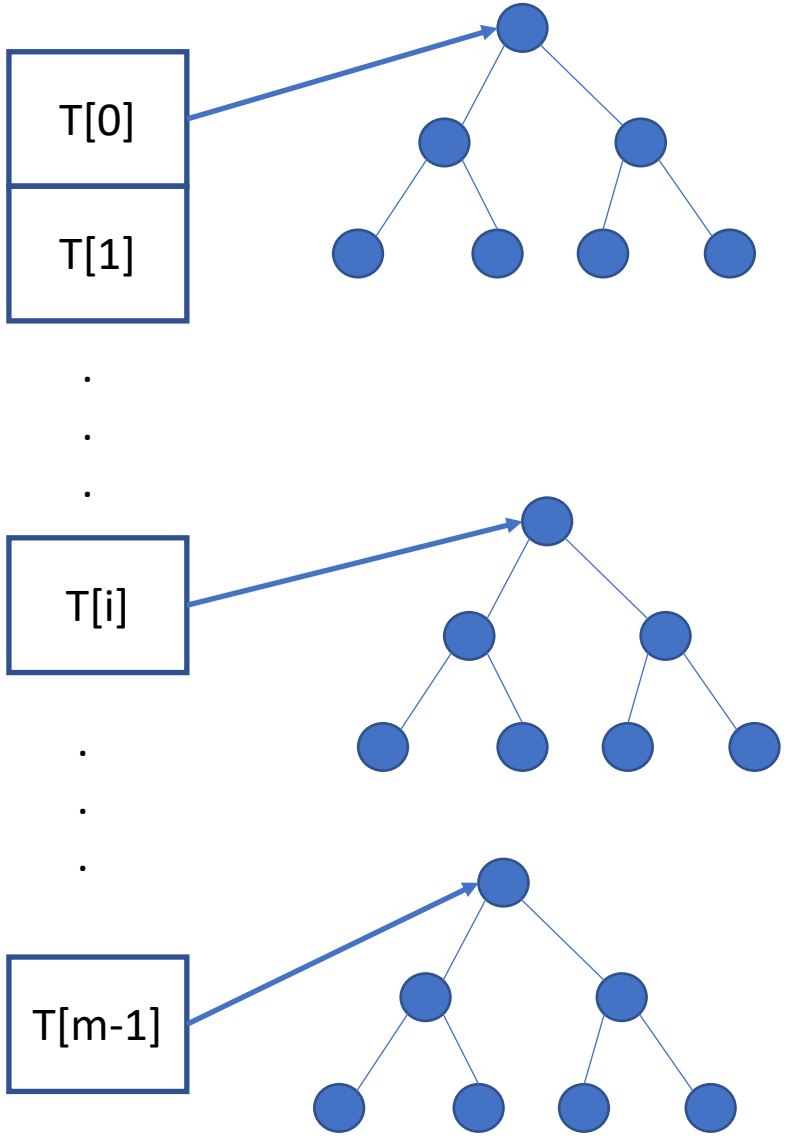
```
h(s[1..k], m) {// hash a string
  code = 0
  for i = 1 to k do
    code = (code * 256 + s[i]) mod m;
  return code;
}
```



STORING AND SEARCHING STRINGS - PSEUDOCODE

- Searching key k
  - Calculating a hash code: $i = h(k)$
  - Do binary search with key k in the binary search tree with the root node T[i]

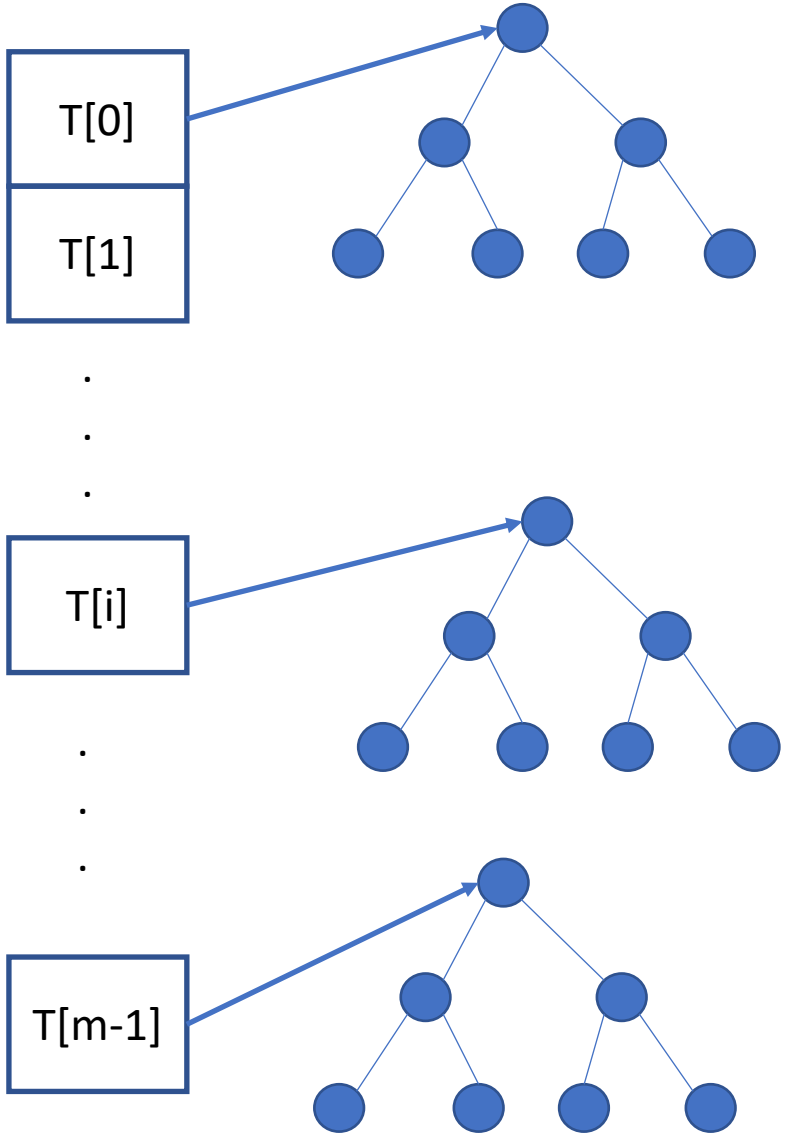
```
FindBST(r, k){
  if r = NULL then return NULL;
  if r.key = k then return r;
  if r.key < k then return FindBST(r.rightChild, k);
  else return FindBST(r.leftChild, k);
}
Find(k){
  i = h(k);// tính mã băm
  node = FindBST(T[i], k); // tìm k trên BST gốc T[i]
  if node = NULL then return 0; else return 1;
}
```



STORING AND SEARCHING STRINGS - PSEUDOCODE

- Inserting a key k
  - Calculating a hash code $i = h(k)$
  - Insert key k into a binary search tree with the root node T[i]

```
InsertBST(r, k){
  if r = NULL then return Node(k);
  if r.key < k then r.rightChild = InsertBST(r.rightChild, k);
  else r.leftChild  = InsertBST(r.leftChild, k);
  return r;
}
Insert(k){
  i = h(k);
  if FindBST(T[i], k) != NULL then return 0;// k already existed
  T[i] = InsertBST(T[i], k);// k does not exist
  return 1;
}
```



COUTING WORD FREQUENCIES IN A DOCUMENT (P.06.15.03)

- Given a text T consisting of a sequence of words: a word is a consecutive sequence of characters taken from {A, B, . . ., Z}, {a, b, c, . . ., z} and {0, 1, 2, . . ., 9}, the remaining characters are not counted as part of the word (but are considered separators between words). Find words in T along with their number of occurrences.
- Data
  - Contains the sequence of characters of the text T (knowing that the words are no more than 20 in length)
- Result
  - Write out in each line a word and the frequency of the word (the words are in lexicographic order)

stdin	stdout
abc def abc abc abcd def	abc 3 abcd 1 def 2

COUTING WORD FREQUENCIES IN A DOCUMENT - PSEUDOCODE

- Algorithm
  - Iterate T to tokenize words
  - Each word is stored in a binary search tree with the key as the word and the value as the frequency of the word
    - If a tokenized word **w** from T already existed in the tree then increase the corresponding value by 1
    - Otherwise, insert a new node with key **w** and the value 1 into the tree

```
struct Node {
  word; // key
  occ; // frequency
  leftChild;
  rightChild;
}
```

```
MakeNode(w) {
  p = Allocate Node;
  p.word = w; p.occ = 1;
  p.leftChild =  NULL;  p.rightChild = NULL;
  return p;
}
Insert(r, word){// chèn 1 từ mới vào BST gốc r
  if r = NULL then r = MakeNode(word);
  if r.word = word then {// từ đã tồn tại
     r.occ = r.occ + 1; // tăng số lần xuất hiện
  }else if r.word < word then {
     r.rightChild = Insert(r.rightChild, word);
  }else {
     r.leftChild = Insert(r.leftChild, word);
  }
  return r;
}
```

COUTING WORD FREQUENCIES IN A DOCUMENT - PSEUDOCODE

- Algorithm
  - Iterate T to tokenize words
  - Each word is stored in a binary search tree with the key as the word and the value as the frequency of the word
    - If a tokenized word **w** from T already existed in the tree then increase the corresponding value by 1
    - Otherwise, insert a new node with key **w** and the value 1 into the tree

```
struct Node {
  word; // key
  occ; // frequency
  leftChild;
  rightChild;
}
```

```
extractWords(T[0..n-1]) {
  root = NULL; end = -1; word = “”;
  for i = 0 to n-1 do {
    if legal(T[i]) then {
       end = end + 1;
       word = word::T[i]; // thêm T[i] vào từ
    }else{
       if end != -1 then {
          root = Insert(root, word);
       }
       end = -1;
    }
  }
}
```

COUTING WORD FREQUENCIES IN A DOCUMENT - PSEUDOCODE

- Algorithm
  - After finishing tokenization and insertion (into the binary search tree), do inorder traversal in the tree: for each visited node, print the word (key) with the corresponding frequency (value)

```
InOrder(r){
  if r = NULL then return;
  InOrder(r.leftChild);
  print(r.word, ‘ ‘, r.occ);
  InOrder(r.rightChild);
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !