



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

BẢNG BĂM

ONE LOVE. ONE FUTURE.

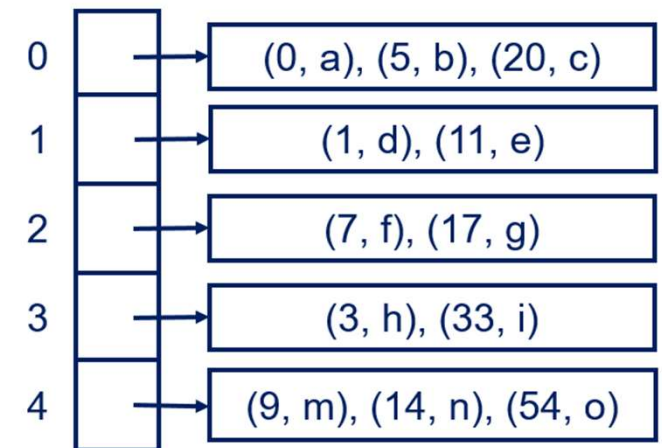
NỘI DUNG

- Bảng băm và hàm băm
- Bài tập tính mã băm chuỗi ký tự (P.06.15.01)
- Bài tập lưu trữ và tìm kiếm các xâu ký tự (P.06.15.02)
- Bài tập đếm số lần xuất hiện các từ trong văn bản (P.06.15.03)



BẢNG BĂM VÀ HÀM BĂM

- Cấu trúc dữ liệu lưu trữ các đối tượng, mỗi đối tượng có 1 khóa, phục vụ truy vấn tìm kiếm nhanh chóng
- Bảng lưu trữ được chia thành các slot được đánh số 0, 1, 2, ..., $m-1$ (m là tham số cấu hình)
- Hàm băm $h(k)$ nhận đầu vào là 1 khóa của đối tượng, hàm sẽ trả ra giá trị chỉ số (từ 0 đến $m-1$) để xác định slot nơi sẽ lưu trữ đối tượng (giá trị $h(k)$ còn được gọi là mã băm của khóa k)
- Xung đột khóa (collision): hai khóa k_1 và k_2 khác nhau nhưng $h(k_1) = h(k_2)$
 - Cơ chế tạo chuỗi (chaining): các đối tượng có cùng mã băm của khóa sẽ được lưu chung vào 1 cụm



BÀI TẬP TÍNH MÃ BĂM CỦA CHUỖI KÝ TỰ (P.06.15.01)

- Cho số nguyên dương m , chuỗi ký tự $s[1..k]$ (các ký tự lấy từ a, b, ..., z) có mã băm được tính theo công thức

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

- Dữ liệu
 - Dòng 1 chứa số nguyên dương n và m ($1 \leq n, m \leq 1000000$)
 - n dòng tiếp theo, mỗi dòng ghi 1 chuỗi ký tự (các ký tự lấy từ a, b, ..., z)
- Kết quả
 - Ghi ra trên mỗi dòng mã băm tương ứng với chuỗi ký tự ở đầu vào

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

BÀI TẬP TÍNH MÃ BẮM CỦA CHUỖI KÝ TỰ - MÃ GIẢ

- Sơ đồ Horner:

$$\begin{aligned}h(s[1..k]) &= s[1]*256^{k-1} + s[2]*256^{k-2} + \dots + s[k]*256^0 \\&= 256*(s[1]*256^{k-2} + s[2]*256^{k-3} + \dots + s[k-1]*256^0) + s[k] \\&= 256*h(s[1..k-1]) + s[k]\end{aligned}$$

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



BÀI TẬP TÍNH MÃ BẮM CỦA CHUỖI KÝ TỰ - CODE

```
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define MAX 200
int n,m;
int hashCode(char* k){
    int c = 0;
    for(int i = 0; i < strlen(k); i++){
        c = (c*256 + k[i]) % m;
    }
    return c;
}
```

```
void solve(){
    scanf("%d%d",&n,&m);
    char k[200];
    for(int i = 0; i < n; i++){
        scanf("%s",k);
        int h = hashCode(k);
        printf("%d\n",h);
    }
}
int main(){
    solve();
    return 0;
}
```



BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ (P.06.15.02)

- Một tập dữ liệu D bao gồm n khóa k_1, k_2, \dots, k_n (khóa là một chuỗi ký tự độ dài từ 1 đến 50). Thực hiện 1 dãy các hành động gồm 1 trong 2 dạng sau:
 - find k : trả về 1 nếu khóa k tồn tại trong D và trả về 0, nếu ngược lại
 - insert k : chèn khóa k vào D (nếu k chưa tồn tại trong D) và trả về 1; và trả về 0 nếu k đã tồn tại trong D
- Dữ liệu
 - Bao gồm 2 khối thông tin
 - Khối thông tin thứ nhất bao gồm các dòng, mỗi dòng ghi 1 khóa mô tả ở trên. Khối thông tin thứ nhất kết thúc bởi dòng chứa *
 - Khối thông tin thứ 2 gồm các dòng, mỗi dòng chứa 1 thao tác thuộc 1 trong 2 dạng nêu trên. Khối thông tin thứ 2 kết thúc bởi 1 dòng chứa ***
- Kết quả
 - Ghi ra trên mỗi dòng kết quả của thao tác tương ứng đọc được ở đầu vào

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

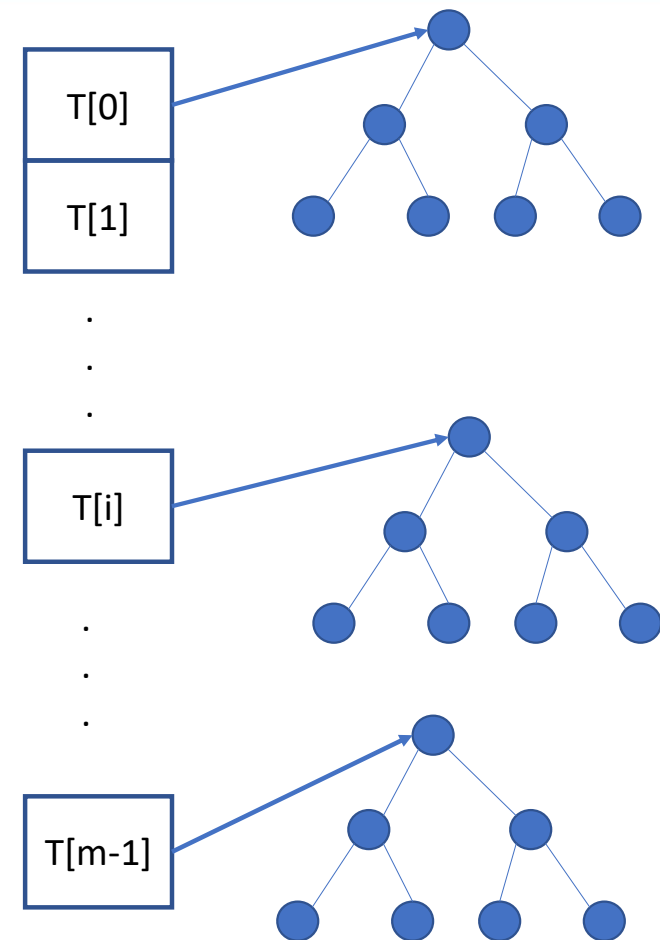


BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ - THUẬT TOÁN - MÃ

- Tổ chức bảng băm kết hợp cây nhị phân tìm kiếm
 - Bảng T được chia thành m phần tử: phần tử thứ i của bảng $T[i]$ là con trỏ tới nút gốc của cây tìm kiếm
 - Cấu trúc dữ liệu mỗi nút (Node) trên cây nhị phân tìm kiếm

```
struct Node {  
    key; // khóa của nút  
    leftChild; // con trỏ đến con trái  
    rightChild; // con trỏ đến con phải  
}
```

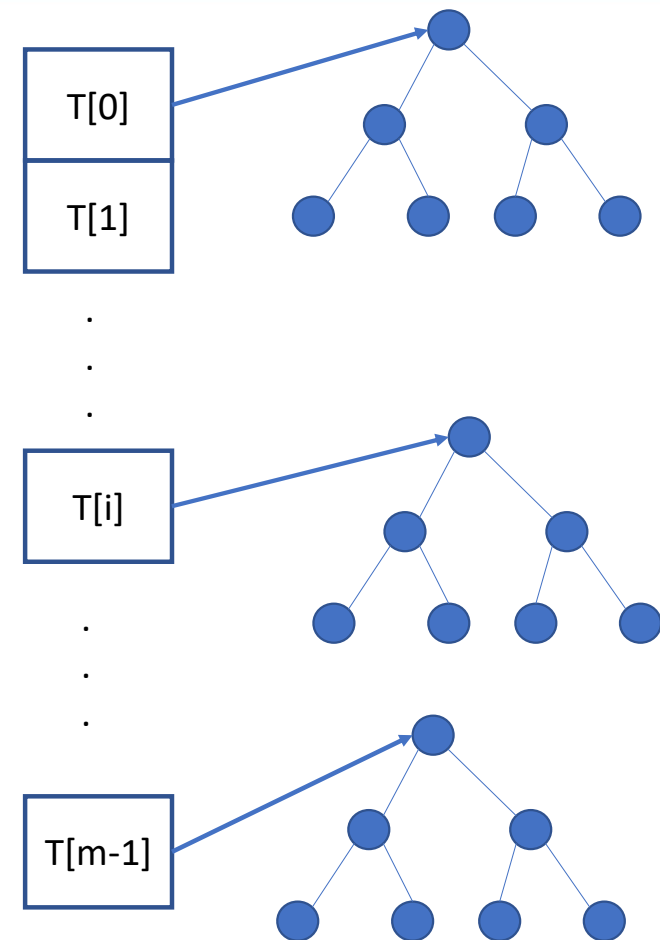
```
h(s[1..k], m) { // hàm băm xâu ký tự  
    code = 0  
    for i = 1 to k do  
        code = (code * 256 + s[i]) mod m;  
    return code;  
}
```



BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ - THUẬT TOÁN - MÃ

- Thao tác tìm kiếm khóa k
 - Tính mã băm $i = h(k)$
 - Thực hiện tìm kiếm khóa k trên cây nhị phân tìm kiếm gốc là $T[i]$

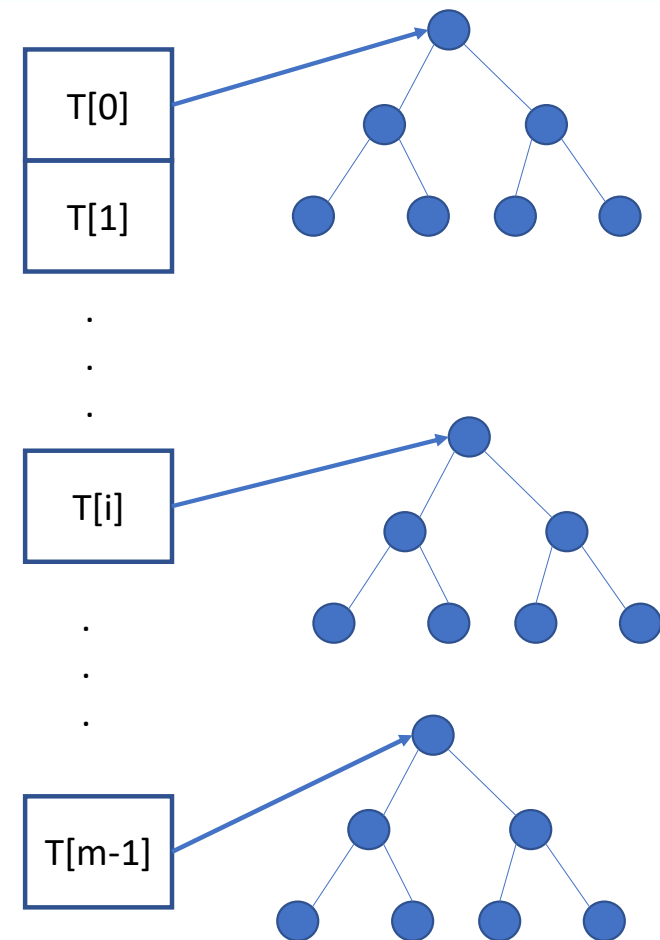
```
FindBST(r, k){  
    if r = NULL then return NULL;  
    if r.key = k then return r;  
    if r.key < k then return FindBST(r.rightChild, k);  
    else return FindBST(r.leftChild, k);  
}  
Find(k){  
    i = h(k); // tính mã băm  
    node = FindBST(T[i], k); // tìm k trên BST gốc T[i]  
    if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



BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ - THUẬT TOÁN - MÃ

- Thao tác chèn khóa k
 - Tính mã băm $i = h(k)$
 - Thực hiện chèn khóa k vào cây nhị phân tìm kiếm gốc là $T[i]$

```
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 đã tồn tại  
    T[i] = InsertBST(T[i], k); // k chưa tồn tại, thực hiện chèn  
    return 1;  
}
```



BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ - CODE

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 50
#define m 10000
typedef struct Node{
    char key[MAX];
    struct Node* leftChild;
    struct Node* rightChild;
}Node;
Node* T[m];
```

```
Node* makeNode(char* k){
    Node* p = (Node*)malloc(sizeof(Node));
    strcpy(p->key,k); p->leftChild = NULL;
    p->rightChild = NULL; return p;
}

int h(char* k){
    int c = 0;
    for(int i = 0; i < strlen(k); i++)
        c = (c*256 + k[i]) % m;
    return c;
}
```



BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ - CODE

```
Node* findBST(Node* r, char* k){
    if(r == NULL) return NULL;
    int c = strcmp(r->key,k);
    if(c == 0) return r;
    if(c < 0) return findBST(r->rightChild,k);
    else return findBST(r->leftChild,k);
}

Node* insertBST(Node* r, char* k){
    if(r == NULL) return makeNode(k);
    if(strcmp(r->key,k) < 0)
        r->rightChild = insertBST(r->rightChild,k);
    else
        r->leftChild = insertBST(r->leftChild, k);
    return r;
}
```

```
int find(char* k){
    int i = h(k);
    Node* p = findBST(T[i],k);
    if(p == NULL) return 0;
    return 1;
}

int insert(char* k){// put(k)
    int i = h(k);
    Node* p = findBST(T[i],k);
    if(p != NULL) return 0;
    T[i] = insertBST(T[i],k);
    return 1;
}
```

BÀI TẬP LƯU TRỮ VÀ TÌM KIẾM CÁC XÂU KÝ TỰ - CODE

```
int main(){
    char k[MAX], cmd[20];
    while(1){
        scanf("%s",k);    if(strcmp(k,"*") == 0) break;
        insert(k);// put k into the hash table (+BST)
    }
    while(1){
        scanf("%s",cmd);
        if(strcmp(cmd,"find") == 0){
            scanf("%s",k);    int ans = find(k);    printf("%d\n",ans);
        }else if(strcmp(cmd,"insert") == 0){
            scanf("%s",k);    int ans = insert(k);    printf("%d\n",ans);
        }else if(strcmp(cmd,"***") == 0){    break;    }
    }
    return 0;
}
```



BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN (P.06.15.03)

- Cho 1 văn bản T bao gồm 1 chuỗi các từ: từ là 1 dãy liên tiếp các ký tự lấy từ {A, B, . . . , Z}, {a, b, c, . . . , z} và {0, 1, 2, . . . , 9}, các ký tự còn lại không được tính là thành phần của từ (mà được coi là các dấu ngăn cách giữa các từ). Hãy tìm các từ trong T cùng với số lần xuất hiện của nó.
- Dữ liệu
 - Chứa dãy các ký tự của văn bản T (biết rằng các từ có độ dài không quá 20)
- Kết quả
 - Ghi ra trên mỗi dòng một từ và số lần xuất hiện của từ đó (các từ được xuất hiện theo thứ tự từ điển)

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



BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN – THUẬT TOÁN - MÃ CHÌ

- Thuật toán

- Duyệt chuỗi ký tự T để tách ra các từ
- Mỗi từ tách được sẽ được lưu vào 1 cây nhị phân tìm kiếm: mỗi nút chứa khóa là 1 từ và số lần xuất hiện của từ đó
 - Nếu 1 từ w tách ra từ T đã tồn tại trên cây nhị phân tìm kiếm thì tăng số lần xuất hiện lên 1
 - Ngược lại, thêm nút mới chứa từ w với số lần xuất hiện bằng 1 vào cây nhị phân tìm kiếm

```
struct Node {  
    word; // khóa (từ của văn bản)  
    occ; // số lần xuất hiện  
    leftChild; // con trỏ đến con trái  
    rightChild; // con trỏ đến con phải  
}
```

```
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;  
}
```

BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN – THUẬT TOÁN - MÃ CHÌA

- Thuật toán

- Duyệt chuỗi ký tự T để tách ra các từ
- Mỗi từ tách được sẽ được lưu vào 1 cây nhị phân tìm kiếm: mỗi nút chứa khóa là 1 từ và số lần xuất hiện của từ đó
 - Nếu 1 từ w tách ra từ T đã tồn tại trên cây nhị phân tìm kiếm thì tăng số lần xuất hiện lên 1
 - Ngược lại, thêm nút mới chứa từ w với số lần xuất hiện bằng 1 vào cây nhị phân tìm kiếm

```
struct Node {  
    word; // khóa (từ của văn bản)  
    occ; // số lần xuất hiện  
    leftChild; // con trỏ đến con trái  
    rightChild; // con trỏ đến con phải  
}
```

```
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;  
        }  
    }  
}
```



BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN – THUẬT TOÁN - MÃ CHẢ

- Thuật toán
 - Sau khi tách hết các từ và chèn vào cây nhị phân tìm kiếm, ta sẽ duyệt cây nhị phân tìm kiếm theo thứ tự giữa: với mỗi nút được thăm, ta sẽ in từ của nút đó với số lần xuất hiện

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



BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN – CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 1000001
typedef struct Node{
    char word[20];
    int occ;
    struct Node* leftChild;
    struct Node* rightChild;
}Node;

Node* root;
char T[N];
int n;
```

```
void input(){
    n = 0;
    while(!feof(stdin)){
        char c = fgetc(stdin);
        T[n] = c;
        n += 1;
    }
    T[n-1] = '\\0';
}
```



BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN – CODE

```
Node* makeNode(char* w){
    Node* nod = (Node*)malloc(sizeof(Node));
    strcpy(nod->word,w);
    nod->occ = 1;
    nod->leftChild = NULL;
    nod->rightChild = NULL;
    return nod;
}
```

```
Node* insert(Node* r, char* w){
    if(r == NULL) return makeNode(w);
    int c = strcmp(r->word,w);
    if(c == 0){
        r->occ += 1; return r;
    }
    if(c < 0)
        r->rightChild = insert(r->rightChild,w);
    else
        r->leftChild = insert(r->leftChild,w);
    return r;
}
```



BÀI TẬP ĐẾM SỐ LẦN XUẤT HIỆN TRONG VĂN BẢN – CODE

```
int legal(char c){
    return (c >= 'a' && c <= 'z' || c >= 'A' && c
        <= 'Z' || c >= '0' && c <= '9');
}
void solve(){
    root = NULL; char word[30]; int end = -1;
    for(int i = 0; i < n; i++){
        if(legal(T[i])){ end++; word[end] = T[i]; }
        else{
            if(end != -1){
                word[end+1]='\0'; root = insert(root,word);
            }
            end = -1;
        }
    }
}
```

```
void inOrder(Node* r){
    if(r == NULL) return;
    inOrder(r->leftChild);
    printf("%s %d\n",r->word,r->occ);
    inOrder(r->rightChild);
}

int main(){
    input();
    solve();
    inOrder(root);
    return 0;
}
```





HUST

 hust.edu.vn  fb.com/dhbkhn

THANK YOU !