


Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. Môn học: TH Cấu trúc dữ liệu & giải thuật	BÀI 11. BẢNG BĂM <i>(Tiếp theo)</i>	
---	---	--

A. MỤC TIÊU:

- Giải thích được cơ chế giải quyết đụng độ bảng băm bằng phương pháp địa chỉ mở (open addressing)
- Cài đặt được cấu trúc bảng băm theo cơ chế địa chỉ mở.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Khái niệm về bảng băm

Phương pháp địa chỉ mở (open addressing): Khi có sự đụng độ, các phần tử trùng khóa sẽ không được lưu vào danh sách liên kết mà sẽ tìm cách để lưu ở vị trí khác trên mảng lưu trữ.

Để có thể xác định vị trí mới của phần tử khi khóa được hashing bị trùng người ta dùng thêm một hàm để xác định vị trí mới của phần tử cần thêm vào bảng băm là $P(x)$. Khi đó vị trí mới sẽ của khóa k_2 sẽ được xác định bằng:

$$\text{Index (mới)} = \text{HashFunction}(\text{index(cũ)} + P(x))$$

Với x là số lần trùng khóa của k_2 khi đã được băm

Tùy vào $P(x)$ mà người ta chia cách xử lý ra làm nhiều loại. Trong nội dung của bài này, chúng ta quan tâm 2 phương pháp là **Linear Probing** và **Quadratic Probing**.

❖ **Linear probing:** Khi thêm vào bảng băm, nếu chỉ mục đó đã có phần tử rồi, chỉ mục sẽ được tính lại theo cơ chế tuần tự: $P(x)$ sẽ có dạng $P(x) = ax + b$. Thường người ta hay chọn $P(x) = x$ nên việc xác định lại index sẽ được tính:

$\text{index} = \text{index} \% \text{hashTableSize}$
 $\text{index} = (\text{index} + 1) \% \text{hashTableSize}$
 $\text{index} = (\text{index} + 2) \% \text{hashTableSize}$
 $\text{index} = (\text{index} + 3) \% \text{hashTableSize}$

Thêm 14 $14\%7=0$	Thêm 8 $8\%7=1$	Thêm 15 $15\%7=1$	Thêm 21 $21\%7=0$	Thêm 12 $12\%7=5$	Thêm 19 $19\%7=5$
0 14	0 14	0 14	0 14	0 14	0 14
1	1 8	1 8	1 8	1 8	1 8
2	2	2 15	2 15	2 15	2 15
3	3	3	3 21	3 21	3 21
4	4	4	4	4	4
5	5	5	5	5 12	5 12
6	6	6	6	6	6 19

❖ **Quadratic Probing:** Tương tự như phương pháp Linear Probing nhưng cách xác định index khi bị trùng sử dụng hàm bậc 2, $P(x) = ax^2 + bx + c$. Giả sử ta chọn $P(x) = x^2$ thì cách xác định index được thực hiện như sau:

$\text{index} = \text{index} \% \text{hashTableSize}$
 $\text{index} = (\text{index} + 12) \% \text{hashTableSize}$
 $\text{index} = (\text{index} + 22) \% \text{hashTableSize}$
 $\text{index} = (\text{index} + 32) \% \text{hashTableSize}$

Thêm 14 $14\%7=0$	Thêm 8 $8\%7=1$	Thêm 15 $15\%7=1$	Thêm 21 $21\%7=0$	Thêm 12 $12\%7=5$
0 14	0 14	0 14	0 14	0 14
1	1 8	1 8	1 8	1 8
2	2	2 15	2 15	2 15
3	3	3	3	3
4	4	4	4 21	4 21
5	5	5	5	5 12
6	6	6	6	6

2. Cấu trúc bảng băm

Do việc các phần tử đều lưu trên mảng nên sử dụng phương pháp địa chỉ mở ta không cần sử dụng danh sách liên kết vì vậy bảng băm chỉ là một mảng 1 chiều. Từ lý do đó số lượng phần tử trong mảng phải lớn hơn số lượng các phần tử cần xử lý. Thông thường khi số lượng phần tử trong bảng băm đã lớn hơn 60% thì ta cần phải mở rộng số lượng phần tử của bảng băm lên gấp đôi.

3. Thao tác trên bảng băm

a. Linear probing

```
#define hashTableSize 17
#define deletedItem -2
int *arr;
int size = 0; //so phan tu trong hashTable
void khoiTao()
{
    arr = new int[hashTableSize];
    for (int i = 0; i < hashTableSize; i++)
        arr[i] = -1;
}

int hashCode(int key)
{
    return key % hashTableSize;
}

void insertNode(int key)
{
    //B1: tìm index của key muốn thêm qua hàm băm
    //B2: Nếu index đã có lưu giá trị thì tìm giá trị mới sử dụng
    linear probing
    //B3: chèn key vào vị trí tìm thấy và tăng số lượng phần tử
    đã lưu trong bảng băm
}

//xoa phan tu co khóa k
```

```

int deleteNode(int key)
{
    //B1: xác định index của key muốn xóa
    //B2: Nếu index của key không trùng với key cần xóa thì
    tìm tiếp tục tìm sử dụng linear probing
    //B3: chèn key vào vị trí tìm thấy và tăng số lượng phần tử
    đã lưu trong bảng băm
    //B4: Nếu tìm thấy, gán vị trí của key là deletedItem và
    giảm số lượng phần tử đang có của bảng băm
    //B5: Nếu tìm không thấy vị trí của key thì: return 0
}
//xóa bảng băm
void display()
{
    for (int i = 0; i < hashTableSize; i++) {
        if (arr[i] != -1)
            printf("%5d", arr[i]);
        printf("\n");
    }
}

```

b. **Quadratic probing:** làm tương tự linear probing.

II. Bài tập hướng dẫn mẫu

Bài 1. Hãy tạo một bảng băm theo cơ chế xử lý xung đột bằng linear probing lưu các số nguyên với kích thước bảng băm là MAX=21. Mở rộng bảng băm khi có số lượng phần tử lớn hơn 60% bảng băm. Hãy xây dựng các hàm sau

- Thêm một giá trị x vào bảng băm
- Xóa một phần tử trong bảng băm
- Tìm kiếm một phần tử trong bảng băm
- Xuất toàn bộ bảng băm

III. Bài tập ở lớp

Bài 2. Thực hiện lại **bài 1** với bảng băm sử dụng cơ chế Quadratic

Bài 3. Cho một chuỗi s chứa các ký tự. Sử dụng bảng băm cho biết các ký tự trong chuỗi xuất hiện bao nhiêu lần (không phân biệt chữ hoa, chữ thường)

VD: $s = \text{"ababcdabc"}$ có số lần xuất hiện $a=3, b=3, c=2, d=1$

Bài 4. Xây dựng bảng băm sử dụng phương pháp xử lý xung đột bằng linear probing lưu thông tin của các cuốn sách trong một thư viện. Biết rằng cấu trúc của một cuốn sách gồm

- Mã sách một chuỗi có 4 ký tự là các chữ cái (Sử dụng là key cho bảng băm)
- Tên sách là một chuỗi
- Số trang là một số nguyên
- Giá bán là một số thực

- a. Nhập thông tin cho bảng băm từ file
- b. Tìm kiếm thông tin của một cuốn sách khi biết mã
- c. Thêm một cuốn sách vào bảng băm
- d. Tăng kích thước của bảng băm
- e. Xóa một cuốn sách theo mã.
- f. Xuất thông tin của tất cả các cuốn sách ra màn hình.

IV. Bài tập về nhà

Bài 5. Cài đặt Bài 4 sử dụng phương pháp xử lý xung đột Quadratic.

Bài 6. Cho 1 chuỗi s . Hãy tìm chuỗi con dài nhất trong s mà không có ký tự nào được lặp lại, cho biết độ dài của chuỗi con đó.

VD: $s = \text{"abcabcbb"}$ có chuỗi con dài nhất là "abc" với chiều dài 3.

-- HẾT --