CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 8 – BẢNG BĂM ĐỊA CHỈ MỞ

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững các phương pháp băm, hàm băm, phương pháp giải quyết va chạm (đụng độ khóa).
- Cài đặt được kiểu dữ liệu bảng băm địa chỉ mở và các thao tác, phép toán trên bảng băm.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

II. Yêu cầu

- Sinh viên phải hoàn thành bài 1, bài 2 và bài 3 thuộc mục V, 3 bài này tạo chung một project. Xóa các thư mục debug của project này, sau đó chép các project vào thư mục: Lab8_CTK39_HoTen_MSSV_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab8_CTK39_HoTen_MSSV_Nhom#.rar.

Ví dụ: Lab6_CTK39_NguyenVanA_141111_Nhom4.rar.

- Sinh viên sẽ nộp bài Lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. Giới thiệu bảng băm

Bảng băm là cấu trúc dữ liệu để cài đặt kiểu dữ liệu từ điển.

Kiểu dữ liệu từ điển là một tập các đối tượng dữ liệu được thao tác với 3 phép toán cơ bản:

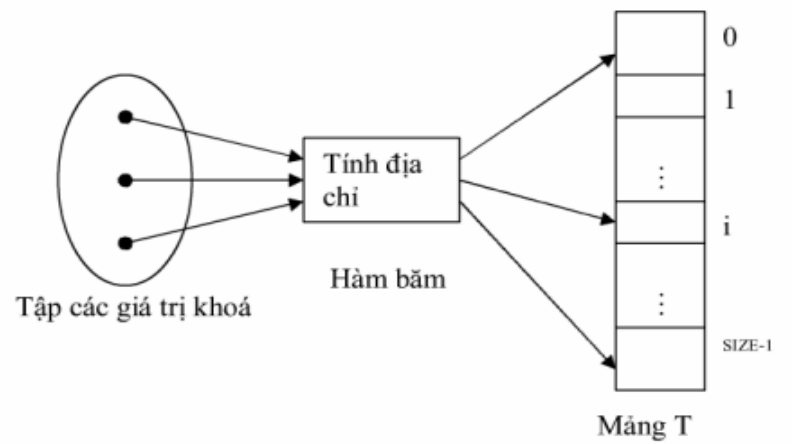
- Tìm kiếm
- Chèn
- Loại bỏ (xóa)

Dạng đơn giản nhất của bảng băm là một mảng các mẫu tin.

Mỗi mẫu tin có một trường dữ liệu đặc biệt gọi là khóa (key)

2. Phương pháp băm

Cho dữ liệu D có khóa K, tìm được chỉ số i sao cho 0 <= I <= SIZE-1. Khi đó, dữ liệu D sẽ được lưu trong thành phần thứ i của mảng (hay bảng băm).



Cần tìm một hàm Hash: index = Hash(key)

Giả sử K là tập giá trị khóa. Khi đó, hàm Hash là một ánh xạ từ tập giá trị khóa (K) vào tập các số nguyên {0, 1, …, SIZE-1}

Hash: K → {0, 1, …, SIZE-1}

Hash được gọi là hàm băm. Hash(key) được gọi là giá trị băm của khóa key. Dữ liệu sẽ được lưu tại vị trí Hash(key)

3. *Các hàm băm*

Trường hợp khóa là số nguyên

- Phương pháp chia

  Hash(k) = k % SIZE

  Trong đó, SIZE là kích thước của bảng băm (thường là một số nguyên tố)

- Phương pháp nhân

  $$hash(k) = \lfloor (\alpha k - \lfloor \alpha k \rfloor).SIZE \rfloor$$

  Trong đó: $\lfloor \alpha k \rfloor$ là phần nguyên của x. Thông thường, α = 0.61803399

Trường hợp khóa là chuỗi

Bước 1: Chuyển xâu ký tự thành số nguyên

"NOTE" → 'N'.128^3 + 'O'.128^2 + 'T'.128 + 'E' =

$78.128^3 + 79.128^2 + 84.128 + 69$

"NOTE" → $78.37^3 + 79.37^2 + 84.37 + 69$ =

$((78.37 + 79).37 + 84).37 + 69$

Bước 2: Áp dụng phương pháp chia (hoặc nhân) để tính giá trị băm

4. *Phương pháp giải quyết va chạm (đụng độ khóa)*

- ❖ Mỗi phần tử của mảng lưu trữ một dữ liệu
- ❖ Khi cần chèn một dữ liệu mới với khóa k, ta
  - Tìm giá trị băm của khóa k (chính là Hash(k))
  - Nếu tại vị trí Hash(k) trống, chèn dữ liệu vào bảng băm.
  - Nếu tại vị trí Hash(k) đã chứa dữ liệu, tiến hành thăm dò một vị trí trống để lưu dữ liệu.
- ❖ Phương pháp tiến hành thăm dò vị trí trống được gọi là phương pháp định địa chỉ mở.
- ❖ Giải pháp tổng quát:
  - Tìm chỉ số i = Hash(k) với k là khóa
  - Lần lượt xem xét các vị trí i_0, i_1, i_2, …, i_m, … trong đó i_0 = i.
  - Dãy các vị trí i_0, i_1, i_2, …, i_m, … được gọi là dãy thăm dò

Thăm dò tuyến tính

$$\begin{cases} i = hash(k) \\ i_m = (i + m)\ \%\ SIZE \end{cases}$$

Thăm dò bình phương

$$\begin{cases} i = hash(k) \\ i_m = (i + m^2)\ \%\ SIZE \end{cases}$$

Băm kép

$$\begin{cases} i = hash(k) \\ i_m = (i + m.h_2(k))\ \%\ SIZE \end{cases}$$

IV. Hướng dẫn thực hành

1. *Tạo dự án*

Sinh viên có thể chọn cài đặt kiểu dữ liệu bảng băm theo ngôn ngữ C# hoặc C++. Các phần sau minh họa cách cài đặt, hướng dẫn bằng mã giả trên ngôn ngữ C++ (bên phải).

- Tạo dự án mới, đặt tên là Lab6_CTK38_HoTen_MSSV_Nhom#
- Trong thư mục Header Files, tạo ra 4 files sau:
  - datatype.h: Định nghĩa các hằng số và kiểu dữ liệu bảng băm
  - hashfunc.h: Định nghĩa các hàm băm
  - hashtable.h: Định nghĩa các thao tác trên bảng băm
  - utility.h: Định nghĩa các hàm tiện ích
- Trong thư mục Source Files, tạo tập tin: program.cpp

2. *Định nghĩa kiểu dữ liệu bảng băm (Cài đặt theo phương pháp định địa chỉ mở)*

- Nhấp đôi chuột vào file datatype.h, định nghĩa các hằng số và kiểu dữ liệu như sau:

```
// ================================================
// Định nghĩa hằng số

#define           EIGHT       8
#define           PRIME       7
#define           MAXSIZE     1000
#define           PHI         0.61803399
#define           EOL         '\r\n'
#define           TAB         '\t'

//
================================================
// Định nghĩa kiểu dữ liệu

typedef           int   KeyType;
typedef           int   DataType;

// Trạng thái của mỗi vị trí trong bảng băm
enum  ItemState
{
      Empty,              // Rỗng
      Active,             // Đang có dữ liệu
      Deleted             // Dữ liệu đã bị xóa
};

// Phương pháp thăm dò
enum  Probing
{
      Linear,             // Thăm dò tuyến tính
      Quadratic,          // Thăm dò bình phương
      Double              // Băm kép
};

// Kiểu dữ liệu thể hiện một phần tử của bảng băm
struct      HashEntry
{
      KeyType     Key;          // Trường khóa
      DataType    Value;        // Giá trị thực
      ItemState   Status;       // Trạng thái
};

typedef HashEntry *EntryPtr;

// Kiểu dữ liệu bảng băm
struct      OpenAddressHashTable
{
      int         Count;              // Số ptử thực tế
      int         Capacity;           // Số ptử tối đa
      EntryPtr*   Elements;           // DSách phần tử
      Probing     ProbingMethod;      // Ppháp thăm dò
};

// Kiểu dữ liệu bảng băm (dùng con trỏ)
typedef OpenAddressHashTable *HashTable;
```

- Trong tập tin program.cpp, nhập đoạn mã sau:

```
#include <iostream>
#include <conio.h>
#include <fstream>
using namespace std;

#include "utility.h"
#include "datatype.h"
#include "hashfunc.h"
#include "hashtable.h"

void main()
{
      _getch();
}
```

Vào menu Build > chọn Build Solution để biên dịch chương trình, kiểm tra lỗi. Nếu có lỗi, kiểm tra lại mã nguồn bạn đã viết. Nếu chương trình không có lỗi, ta sẽ cài đặt các thao tác, phép toán trên bảng băm.

V. Bài tập thực hành

Bài 1. Tạo các hàm tiện ích

a. Hàm kiểm tra số nguyên tố

```
// Hàm kiểm tra số n có phải là số nguyên tố
int IsPrime(unsigned int n)
{
	int sn = (int)sqrt((double)n);
	for (int i=2; i<sn; i++)
	{
		if (n % i == 0)
			return 0;
	}
	return 1;
}
```

b. Tìm số nguyên tố lớn hơn và gần nhất với N

```
// Tìm số nguyên tố lớn hơn và gần nhất với N
unsigned int FindNearestPrime(unsigned int n)
{
	if (n % 2 == 0) n++;
	while (!IsPrime(n))
		n += 2;
	return n;
}
```

Bài 2. Xây dựng các hàm băm và hàm thăm dò

a. Các hàm băm

```
// Hàm băm một giá trị số nguyên
unsigned int Hash(unsigned int value,
                          unsigned int tableSize)
{
      return value % tableSize;
}

// Hàm băm giá trị chuỗi
unsigned int Hash(const char *value,
                          unsigned int tableSize)
{
      int ind = 0;
      unsigned int sum = 0;

      while (value[ind] != NULL)
      {
            sum = (sum * 37) + value[ind];
            sum = sum % tableSize;
            ind++;
      }
      return sum;
}

// Hàm băm thứ cấp, dùng cho băm kép
unsigned int Hash2(unsigned int value)
{
      return EIGHT - value % EIGHT;
}

// Hàm băm thứ cấp, dùng cho băm kép
unsigned int Hash2(const char *value)
{
      int ind = 0;
      unsigned int sum = 0;

      while (value[ind] != NULL)
      {
            sum = (sum * 37) + value[ind];
            sum = sum % EIGHT;
            ind++;
      }
      return EIGHT - sum;
}
```

```
// Hàm băm thứ cấp, dùng cho băm kép
unsigned int Hash3(unsigned int value)
{
      return 1 + value % PRIME;
}
```

b. Hàm thăm dò

```
// Hàm thăm dò tuyến tính
unsigned int LinearProbing(unsigned int curr,
         unsigned int step,  unsigned int tableSize)
{
      return (curr + step) % tableSize;
}

// Hàm thăm dò bình phương
unsigned int SquareProbing(unsigned int curr,
         unsigned int step,  unsigned int tableSize)
{
      return (curr + step * step) % tableSize;
}

// Hàm thăm dò theo phương pháp băm kép
unsigned int DoubleProbing(unsigned int curr,
         unsigned int step,  KeyType key,
         unsigned int tableSize)
{
      unsigned int h2 = Hash2(key);
      return (curr + step * h2) % tableSize;
}
```

Bài 3. Cài đặt các thao tác trên bảng băm

```
// Khai báo nguyên mẫu của hàm thay đổi kích thước bảng băm
void Resize(HashTable &ht, unsigned int newSize);

// Lưu dữ liệu vào bảng băm
void Update(EntryPtr item, KeyType k, DataType d)
{
	item->Key = k;
	item->Value = d;
	item->Status = Active;
}

// Tạo một phần tử mới cho bảng băm
EntryPtr NewEntry(KeyType k, DataType d)
{
	EntryPtr entry = new HashEntry;
	Update(entry, k, d);
	return entry;
}

// Khởi tạo một bảng băm rỗng
HashTable Initialize(unsigned int maxSize,
                     Probing probMethod)
{
	maxSize = FindNearestPrime(maxSize);

	HashTable table = new OpenAddressHashTable;
	table->Count = 0;
	table->Capacity = maxSize;
	table->ProbingMethod = probMethod;
	table->Elements = new EntryPtr[maxSize];

	for (int i=0; i<maxSize; i++)
		table->Elements[i] = NULL;
	return table;
}

// Kiểm tra bảng băm đã ở mức độ đầy 2/3
bool IsFull(HashTable ht)
{
	return (ht->Count / ht->Capacity * 1.0) > 0.66;
}

// Thăm dò
unsigned int DoProbing(HashTable ht, KeyType key,
                       unsigned int currPos,
                       unsigned int step)
{
	switch (ht->ProbingMethod)
	{
		case Linear:
			return LinearProbing(currPos, step, ht->Capacity);
		case Quadratic:
			return SquareProbing(currPos, step, ht->Capacity);
		case Double:
			return DoubleProbing(currPos, step, key, ht->Capacity);
	}
}

// Hàm tìm vị trí của phần tử chứa khóa key.
// Trả về chỉ số của phần tử cuối cùng được kiểm tra.
unsigned int Search(HashTable ht, KeyType key)
{
	unsigned int pos = Hash(key, ht->Capacity),
		index = pos, m = 1;

	while (ht->Elements[index] != NULL &&
		ht->Elements[index]->Key != key)
	{
		index = DoProbing(ht, key, pos, m);
		m++;
	}
	return index;
}

// Kiểm tra phần tử thứ pos trong bảng băm
// có chứa dữ liệu
bool IsActive(HashTable ht, unsigned int index)
{
	return (ht->Elements[index] != NULL) &&
		(ht->Elements[index]->Status == Active);
}

// Kiểm tra bảng băm có chứa dữ liệu với khóa key
bool Contains(HashTable ht, KeyType key)
{
	unsigned int index = Search(ht, key);
	return IsActive(ht, index);
}

// Thêm một phần tử mới vào bảng băm
```

```
bool Insert(HashTable &ht, KeyType key, DataType item)
{
	unsigned int index = Search(ht, key);
	if (IsActive(ht, index))
		return false;
	else
	{
		ht->Count++;
		ht->Elements[index] = NewEntry(key, item);

		if (IsFull(ht))
			Resize(ht, ht->Capacity * 2);

		return true;
	}
}

// Thay đổi giá trị của phần tử có khóa key
bool Update(HashTable ht, KeyType key, DataType item)
{
	unsigned int index = Search(ht, key);
	if (IsActive(ht, index))
		return false;
	else
	{
		Update(ht->Elements[index], key, item);
		return true;
	}
}

// Thay đổi kích thước của bảng băm
void Resize(HashTable &ht, unsigned int newSize)
{
	HashTable table = Initialize(newSize, ht-
>ProbingMethod);

	for (int i=0; i<ht->Capacity; i++)
		if (IsActive(ht, i))
		{
			Insert(table, ht->Elements[i]->Key, ht-
>Elements[i]->Value);
			delete ht->Elements[i];
		}
	delete ht;
	ht = table;
}
```

```
// Loại bỏ một phần tử có khóa key
void Remove(HashTable ht, KeyType key)
{
	unsigned int index = Search(ht, key);
	if (IsActive(ht, index))
	{
		//delete ht->Elements[index];
		//ht->Elements[index] = NULL;
		ht->Elements[index]->Status = Deleted;
		ht->Count--;
	}
}

// Xuất nội dung của một phần tử trong bảng băm
void PrintHashEntry(EntryPtr entry)
{
	cout << entry->Value;
}

// Xuất nội dung toàn bộ bảng băm
void PrintHashTable(HashTable ht)
{
	cout << endl << "Dung luong : " << ht->Capacity;
	cout << endl << "So phan tu : " << ht->Count << endl;
	cout << endl << "Du lieu cua bang bam ";
	cout << endl << "INDEX" << TAB
			<< "KEY" << TAB
			<< "DATA";

	for (int i=0; i<ht->Capacity; i++)
	{
		if (IsActive(ht, i))
		{
			cout << endl << i << TAB
				<< ht->Elements[i]->Key << TAB;
			PrintHashEntry(ht->Elements[i]);
		}
		else
			cout << endl << i << TAB
				<< '-' << TAB << '-';
	}
	cout << endl << endl;
}
```

Bài 4. Kiểm tra chương trình và xem kết quả

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <string>

using namespace std;

#include "utility.h"
#include "datatype.h"
#include "hashfunc.h"
#include "hashtable.h"

void main()
{
	HashTable h = Initialize(11, Quadratic);
	//KeyType key[] = { "monday", "tuesday", "wednesday",
"thursday", "friday", "saturday", "sunday" };
	DataType data[] = {32, 15, 25, 44, 36, 21, 534, 702,
			105, 523, 959, 699, 821, 883, 842,
			686, 658, 4, 20, 382, 570, 344, 125,
			200, 217, 175, 153, 201};
	int n = 28;  // So phan tu cua mang data

	for (int i=0; i<n; i++)
	{
		cout << endl << "Sau khi chen them : "
			<< data[i] << endl;

		Insert(h, data[i], data[i]);
		PrintHashTable(h);
		_getch();
	}
	for (int i=0; i<n; i++)
	{
		cout << endl << "Sau khi xoa : "
			<< data[i] << endl;

		Remove(h, data[i]);
		PrintHashTable(h);
		_getch();
	}
	_getch();
}
```

Bài 5: Cài đặt kiểu dữ liệu bảng băm địa chỉ mở dùng để lưu trữ dữ liệu với khóa có kiểu chuỗi.

VI. Bài tập làm thêm

Bài 1. Thống kê từ trong tập tin văn bản

Các văn bản được lưu trữ thành từng dòng trong các file văn bản, mỗi dòng có chiều dài không quá 127 ký tự. Hãy đề xuất cấu trúc dữ liệu thích hợp để lưu trữ trong bộ nhớ trong của máy tính tên từ và số lần xuất hiện của từ đó trong tập tin văn bản. Với cấu trúc dữ liệu này, hãy trình bày thuật toán và cài đặt chương trình thực hiện việc thống kê xem các từ trong file văn bản xuất hiện với tần suất như thế nào? Cho biết văn bản có bao nhiêu từ, bao nhiêu tên từ?

Bài 2. Từ điển Anh – Việt

Cài đặt kiểu dữ liệu từ điển Anh-Việt bằng bảng băm.

=== HẾT ===