CHƯƠNG 5: BẢNG BĂM (HASH TABLE)

1. KHÁI NIỆM

- **Bảng băm (Hash Table)** là cấu trúc dữ liệu ánh xạ **một khóa (key)** đến **một chỉ số (index)** thông qua **hàm băm (hash function)**.

Tính chất:

- Truy xuất, chèn, xóa nhanh gần như **O(1)** trung bình.
- Ánh xạ key → index dựa vào hàm băm.
- Có thể xảy ra **xung đột (collision)** khi nhiều khóa có cùng vị trí.

---

2. HÀM BĂM

- Hàm băm là hàm ánh xạ từ **key → index trong mảng**

  Ví dụ:

  cpp
  CopyEdit
  int hashFunc(int key, int size) {
  return key % size;
  }

---

3. XỬ LÝ XUNG ĐỘT (COLLISION HANDLING)

3.1 Dò tuyến tính (Linear Probing)

- Khi vị trí băm đã có phần tử, ta tìm vị trí tiếp theo gần nhất còn trống.

Cài đặt Hash Table – Dò tuyến tính

```
#include <iostream>
using namespace std;

const int SIZE = 10;

class HashTable {
private:
   int table[SIZE];

public:
   HashTable() {
```

```
        for (int i = 0; i < SIZE; i++)
            table[i] = -1; // -1 nghĩa là trống
    }

    int hashFunc(int key) {
        return key % SIZE;
    }

    void insert(int key) {
        int index = hashFunc(key);
        int i = 0;
        while (table[(index + i) % SIZE] != -1) {
            i++;
            if (i == SIZE) {
                cout << "Bảng đầy!\n";
                return;
            }
        }
        table[(index + i) % SIZE] = key;
    }

    bool search(int key) {
        int index = hashFunc(key);
        int i = 0;
        while (table[(index + i) % SIZE] != -1) {
            if (table[(index + i) % SIZE] == key)
                return true;
            i++;
            if (i == SIZE) break;
        }
        return false;
    }

    void display() {
        for (int i = 0; i < SIZE; i++)
            cout << i << ": " << table[i] << endl;
    }
};

int main() {
    HashTable ht;
    ht.insert(10);
    ht.insert(20);
    ht.insert(30);
    ht.insert(21);

    ht.display();
```

```
    cout << "Tìm 21? " << (ht.search(21) ? "Có" : "Không") << endl;
    cout << "Tìm 99? " << (ht.search(99) ? "Có" : "Không") << endl;

    return 0;
}
```

3.2 Xử lý bằng danh sách liên kết (Chaining)

- Mỗi vị trí là một danh sách liên kết.
- Khi có xung đột, phần tử mới được **thêm vào danh sách tại vị trí đó**.

Cài đặt Hash Table – Chuỗi liên kết (Chaining)

```
#include <iostream>
#include <list>
using namespace std;

const int SIZE = 10;

class HashTable {
private:
    list<int> table[SIZE];

public:
    int hashFunc(int key) {
        return key % SIZE;
    }

    void insert(int key) {
        int index = hashFunc(key);
        table[index].push_back(key);
    }

    bool search(int key) {
        int index = hashFunc(key);
        for (int x : table[index])
            if (x == key)
                return true;
        return false;
    }

    void display() {
        for (int i = 0; i < SIZE; i++) {
            cout << i << ": ";
            for (int x : table[i])
                cout << x << " -> ";
```

```
            cout << "null\n";
        }
    }
};

int main() {
    HashTable ht;
    ht.insert(15);
    ht.insert(25);
    ht.insert(35);
    ht.insert(10);
    ht.insert(7);

    ht.display();

    cout << "Tìm 25? " << (ht.search(25) ? "Có" : "Không") << endl;
    cout << "Tìm 100? " << (ht.search(100) ? "Có" : "Không") << endl;

    return 0;
}
```