```
/*
Name: Bang bam dia chi mo - do tuyen tinh
Author: Nhan V.T
Last update: 2022-12-16
Note:  +su dung ky thuat cap phat dong va bam lai
       +luu con tro thay vi node truc tiep de tiet kiem vung nho
*/
#include <stdio.h>
#include <stdlib.h>
#define DEFAULT_LOAD_FACTOR 0.8

//cau truc node
struct Node {
	int key;
	int value;
};
typedef Node* NodePtr;
//cap phat node moi
Node* createNode(int key, int value)
{
	Node* p = new Node;
	p->key = key;
	p->value = value;
	return p;
}

//tat ca cac node bi xoa se tham chieu den 1 node dac biet DELETED_NODE
//=> giup tiet kiem vung nho va chi phi tim kiem
static NodePtr getUniqueDeletedNode()
{
	//deleted node chi tao 1 lan duy nhat trong suot chuong trinh
	static NodePtr deletedNode = NULL;
	if (deletedNode == NULL)
		deletedNode = createNode(-1, -1);
	return deletedNode;
}
```

```
//opening adressing hash table
struct HashTable{
	NodePtr * arr;
	int count; //so luong phan tu
	int size; //kich thuoc bang bam
};
//ham bam
int hash(int key, int tableSize)
{
	return key % tableSize;
}
//ham do tuyen tinh
int prob(int i)
{
	return i; //do tuyen tinh P(i) = i
	//return i*i; //do bac 2
}

//khoi tao bang bam
void HT_init(HashTable &ht, int capacity)
{
```

```
	ht.size = capacity;
	ht.count = 0;
	ht.arr = new NodePtr[ht.size];

	for (int i = 0; i < ht.size; i++)
	{
		ht.arr[i] = NULL;
	}
}

//giai phong bang bam
void HT_free(HashTable &ht)
{
	for (int i = 0; i < ht.size; i++)
	{
		delete ht.arr[i];
	}
	ht.size = 0;
	ht.count = 0;
	delete[] ht.arr;
}

//them cap phan tu <key, value> vao bang bam
void HT_push(HashTable& ht, int key, int value);
//bam lai bang bam voi kich thuoc moi
void HT_rehashing(HashTable& ht, int capacity_new)
{
	HashTable ht_new;
	HT_init(ht_new, capacity_new);

	//chuyen du lieu tu bang bam cu sang bang bam moi
	for (int i = 0; i < ht.size; i++)
	{
		if (ht.arr[i] != NULL && ht.arr[i] != getUniqueDeletedNode())
		{
			int key = ht.arr[i]->key;
			int value = ht.arr[i]->value;
			HT_push(ht_new, key, value);
		}
	}

	//cap nhat lai bang bam
	HashTable ht_old = ht;
	ht = ht_new;
	HT_free(ht_old);
}
void HT_push(HashTable& ht, int key, int value)
{
	if (ht.count == ht.size) {
		printf("Bang bam day!\n");
		return;
	}

	int h = hash(key, ht.size);
	int pos;
	for (int i = 0; i < ht.size; i++){
		pos = (h + prob(i)) % ht.size;
		if (ht.arr[pos] != NULL && ht.arr[pos]->key == key) {
			printf("Khoa bi trung!\n");
			return;
		}
```

```
                if (ht.arr[pos] == NULL || ht.arr[pos] == getUniqueDeletedNode()) {
                        ht.arr[pos] = createNode(key, value);
                        ht.count++;
                        break;
                }
        }

        //tinh chi so tai
        double loadFactor = ht.count *1.0 / ht.size;
        if (loadFactor > DEFAULT_LOAD_FACTOR) {
                HT_rehashing(ht, ht.size * 2);
        }
}

void HT_remove(HashTable& ht, int key)
{
        int h = hash(key, ht.size);
        int pos;

        for (int i = 0; i < ht.size; i++)
        {
                pos = (h + prob(i)) % ht.size;
                if (ht.arr[pos] != NULL && ht.arr[pos] != getUniqueDeletedNode())
                {
                        Node* tmp = ht.arr[pos];

                        //gan node bi xoa thanh node DeletedNode
                        //de tiet kiem chi phi tim kiem
                        ht.arr[pos] = getUniqueDeletedNode();

                        delete tmp; //thu hoi vung nho
                        break;
                }
        }
}
//tra cuu phan tu dua tren key
Node* HT_get(HashTable ht, int key)
{
        int h = hash(key, ht.size);
        int pos;

        for (int i = 0; i < ht.size; i++)
        {
                pos = (h + prob(i)) % ht.size;

                if (ht.arr[pos] == NULL)
                        return NULL;
                if (ht.arr[pos] != NULL && ht.arr[pos]->key == key) {
                        return ht.arr[pos];
                }
        }
        return NULL;
}
```

```
//hien thi bang bam
void HT_traverse(HashTable ht)
{
        for (int i = 0; i < ht.size; i++)
        {
                int key = -1, value = -1;
                if (ht.arr[i] != NULL && ht.arr[i] != getUniqueDeletedNode()){
```

```
                        key = ht.arr[i]->key;
                        value = ht.arr[i]->value;
                }
                printf("[%d]: <%4d,%4d>\n", i, key, value);
        }

        double loadFactor = 0;
        if (ht.count != 0 )
                loadFactor = ht.count*1.0 / ht.size;
        printf("He so tai hien tai: [%.2f]% \n", loadFactor);
}
```

```
void menu()
{
        printf("\n----Menu---");
        printf("\n0.Thoat");
        printf("\n1.Tai du lieu mac dinh");
        printf("\n2.Them mot phan tu");
        printf("\n3.Xoa mot phan tu");
        printf("\n4.Tra cuu phan tu theo key:");
}

void main()
{
        //khai bao bang bam: loai dia chi moi, dung pp do tuyen tinh, kich thuoc ban dau = 5
        HashTable ht;
        int tableSize = 5;

        //khoi tao bang bam
        HT_init(ht, tableSize);

        int choose = 0;
        do
        {
                system("cls");
                HT_traverse(ht);
                menu();
                printf("\nChon: ");
                scanf_s("%d", &choose);

                switch (choose)
                {
                case 0:
                {
                                break;
                }
                case 1:
                {
                                int a[] = { 30, 21, 4, 13, 15, 18, 22, 28, 24, 45 };
                                int n = sizeof(a) / sizeof(a[0]);

                                for (int i = 0; i < n; i++)
                                {
                                        HT_push(ht, a[i], a[i]);
                                }

                                break;
                }
                case 2:
                {
                                int key;
```

```
                                printf("\nNhap key: ");
                                scanf_s("%d", &key);
                                HT_push(ht, key, key); //lay value trung key
                                break;
                        }
                        case 3:
                        {
                                int key;
                                printf("\nNhap key: ");
                                scanf_s("%d", &key);
                                HT_remove(ht, key);
                                break;
                        }
                        case 4:
                        {
                                int key;
                                printf("\nNhap key: ");
                                scanf_s("%d", &key);
                                Node* p = HT_get(ht, key);
                                if (p != NULL)
                                {
                                        printf("Tim thay %d", key);
                                }
                                else
                                {
                                        printf("Khoa %d khong ton tai", key);
                                }
                                break;
                        }

                        default:
                                break;
                        }

                        system("pause");

                } while (choose != 0);

                //huy bang bam
                HT_free(ht);
}
```