1.В чем отличия между открытым и закрытым хешированием данных?

При закрытом хешировании в таблице сегментов хранятся непосредственно элементы словаря, а не заголовки списков. Поэтому в каждом сегменте может храниться только один элемент словаря. При закрытом хешировании применяется методика повторного хеширования.

Открытое хеширование или Метод цепочек – это технология разрешения коллизий, которая состоит в том, что элементы множества с равными хеш-значениями связываются в цепочку-список.

2. Объясните суть коллизий при закрытом хешировании данных.

Если произойдет попытка поместить элемент *х* в сегмент с номером *h(x)*, который уже занят другим элементом (такая ситуация называется коллизией), то в соответствии с методикой повторного хеширования выбирается последовательность других номеров сегментов

3. Перечислите способы борьбы с коллизиями

При заполнении таблицы могут возникать коллизии, для борьбы с которыми разработаны специальные методы, которые в основном сводятся к методам "цепочек" и "открытой адресации".

В методе цепочек для разрешения коллизий во все записи вводятся указатели, используемые для организации списков – "цепочек переполнения". В случае возникновения коллизии при заполнении таблицы в список для требуемого адреса хеш-таблицы добавляется еще один элемент.

Метод открытой адресации состоит в том, чтобы, пользуясь каким-либо алгоритмом, обеспечивающим перебор элементов таблицы, просматривать их в поисках свободного места для новой записи.

using System;

using System.Collections.Generic;

namespace lab\_five

{

internal class Program

{

public static List<int>[] HashArray { get; set; }

public static void Add(int value)

{

var k = GetHash(value);

if (HashArray[k] == null)

{

HashArray[k] = new List<int>{ value };

}

else

{

HashArray[k].Add(value);

}

}

public static string Search(int value)

{

var k = GetHash(value);

if (HashArray[k] != null)

{

var result = Convert.ToBoolean(HashArray[k].Contains(value)) ? "Exist" : "Not exist";

return $"Hash: {k}; Result: {result}";

}

else

{

return "Not found";

}

}

private static int GetHash(int value)

{

int result = 0;

if (Type.GetTypeCode(value.GetType()) == TypeCode.Int32)

{

int temp = Convert.ToInt32(value);

while (temp != 0)

{

result += (temp % 10);

temp /= 10;

}

}

return result % HashArray.Length;

}

private static void Main(string[] args)

{

Console.WriteLine("Enter length of array: ");

int arrayLength = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("\nEnter length of hashtable");

int hashTableLength = Convert.ToInt32(Console.ReadLine());

int[] array = new int[arrayLength];

Console.WriteLine("\nEnter range");

Console.Write("Start: ");

int startRange = Convert.ToInt32(Console.ReadLine());

Console.Write("End: ");

int endRange = Convert.ToInt32(Console.ReadLine());

HashArray = new List<int>[hashTableLength];

Console.WriteLine();

for (int i = 0; i < arrayLength; i++)

{

Console.Write($"Element {i + 1}: ");

int value = -1;

while (value < startRange || value > endRange)

{

value = Convert.ToInt32(Console.ReadLine());

}

array[i] = value;

Add(value);

}

Console.WriteLine("\nSource array");

for (int i = 0; i < arrayLength; i++)

{

Console.Write($" {array[i]} | ");

}

Console.WriteLine("\n\nHash-table");

for (int i = 0; i < HashArray.Length; i++)

{

Console.Write($"{i}: ");

if (HashArray[i] != null)

{

for (int j = 0; j < HashArray[i].Count; j++)

{

Console.Write($"{HashArray[i][j]} ");

}

}

Console.WriteLine();

}

Console.WriteLine("\nSearch:");

int searchValue = Convert.ToInt32(Console.ReadLine());

Console.WriteLine($"{Search(searchValue)}");

Console.ReadKey();

}

}

}

