Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 14

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы c открытой адресацией»

Выполнила:

Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: асс. Андронова М.В.

2023, Минск

4. В соответствии со своим вариантом построить хеш-таблицы разного размера (например, 16, 32 или 32, 64, 128) с коллизиями. Исследовать время поиска в хеш-таблицах. В приложении Excel построить соответствующие графики.

Вариант 2

Использовать в проекте функцию универсального хеширования.

|  |
| --- |
| Исходный код |
| Главный файл |
| // 2. Использовать в проекте функцию универсального хеширования.  #include "Header.h"  #include <iostream>  #include <windows.h>  using namespace std;  struct AAA  {  int key;  char\* mas;  AAA(int k, char\* z)  {  key = k; mas = z;  } AAA() {}  };  //-------------------------------  int key(void\* d)  {  AAA\* f = (AAA\*)d; return f->key;  }  //-------------------------------  void AAA\_print(void\* d)  {  cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;  }  //-------------------------------  int main()  {  setlocale(LC\_ALL, "rus");  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  clock\_t start, end;  double searching\_time;  AAA a1(1, (char\*)"one"), a2(2, (char\*)"two"), a3(4, (char\*)"three"), a4(2, (char\*)"fo");  int siz = 10, choice, k = 0;  cout << "Введите размер хеш-таблицы" << endl; cin >> siz;  Object H = create(siz, key);  cout << "1 - вывод хеш-таблицы" << endl;  cout << "2 - добавление элемента" << endl;  cout << "3 - удаление элемента" << endl;  cout << "4 - поиск элемента" << endl;  cout << "0 - выход" << endl;  for (;;)  {  cout << "сделайте выбор" << endl;  cin >> choice;  switch (choice)  {  case 0: exit(0);  case 1: H.scan(AAA\_print); break;  case 2: { AAA\* a = new AAA;  char\* str = new char[20];  cout << "введите ключ" << endl;  while (k != siz) {  cout << "Ключ: " << k << "\n";  a->key = k;  cout << "введите строку" << endl;  cin >> str;  a->mas = str;  if (H.N == H.size)  cout << "Таблица заполнена" << endl;  else  H.insert(a);  k++;  }  } break;  case 3: { cout << "введите ключ для удаления" << endl;  cin >> k;  H.deleteByKey(k);  } break;  case 4: { cout << "введите ключ для поиска" << endl;  cin >> k;  start = clock();  if (H.search(k) == NULL)  cout << "Элемент не найден" << endl;  else  AAA\_print(H.search(k));  end = clock();  searching\_time = (double)(end - start) / CLOCKS\_PER\_SEC;  cout << "время поиска: " << searching\_time << endl;  } break;  }  }  return 0;  } |
| Файл с функциями |
| #include "Header.h"  #include <iostream>  int HashFunction(int key, int size, int p) // Хеш-функция  {  double key2 = 5 \* ((0.6180339887499 \* key) - int((0.6180339887499 \* key))); // алгоритм хеш-функции  return (p + key) % size;  }  int Next\_hash(int hash, int size, int p) // для след  {  return (hash + 5 \* p + 3 \* p \* p) % size;  }  //-------------------------------  int universal(int key, int size, int p) // обеспечивает равномерное хеширование: для очередного ключа вероятности помещения его в любую ячейку совпадают  {  int h = 0, a = 314445, b = 37542;  h = (a \* h + key) % size;  return (h < 0) ? (h + size) : h;  }  Object create(int size, int(\*getkey)(void\*))  {  return \*(new Object(size, getkey)); // выделение памяти  }  //-------------------------------  Object::Object(int size, int(\*getkey)(void\*))  {  N = 0;  this->size = size;  this->getKey = getkey;  this->data = new void\* [size];  for (int i = 0; i < size; ++i)  data[i] = NULL;  }  //-------------------------------  bool Object::insert(void\* d) // добавление элементов  {  bool b = false;  if (N != size)  for (int i = 0, t = getKey(d), j = universal(t, size, 0);  i != size && !b; j = Next\_hash(j, size, ++i))  if (data[j] == NULL || data[j] == DEL)  {  data[j] = d; // заполняем поля  N++;  b = true;  }  return b;  }  //-------------------------------  int Object::searchInd(int key) // функция для поиска по ключу  {  int t = -1;  bool b = false; // переменная для поиска  if (N != 0)  // поиск будет происходить, пока не найдем элемент и b не станет равно true  for (int i = 0, j = universal(key, size, 0); data[j] != NULL && i != size && !b; j = universal(key, size, ++i))  if (data[j] != DEL)  if (getKey(data[j]) == key)  {  t = j; b = true;  }  return t;  }  //-------------------------------  void\* Object::search(int key) // вывод результата поиска  {  int t = searchInd(key);  return(t >= 0) ? (data[t]) : (NULL);  }  //-------------------------------  void\* Object::deleteByKey(int key) // функция для удаления  {  int i = searchInd(key); // ищем элемент  void\* t = data[i]; // присваиваем указатель на этот элемент  if (t != NULL)  {  data[i] = DEL; // удаляем  N--;  }  return t;  }  //-------------------------------  bool Object::deleteByValue(void\* d)  {  return(deleteByKey(getKey(d)) != NULL);  }  //-------------------------------  void Object::scan(void(\*f)(void\*)) // вывод элементов  {  for (int i = 0; i < this->size; i++) // перебираем с помощью цикла  {  std::cout << " Элемент " << i; // и выводим  if ((this->data)[i] == NULL)  std::cout << " Пусто!" << std::endl;  else  if ((this->data)[i] == DEL)  std::cout << " Удален" << std::endl;  else  f((this->data)[i]);  }  } |
| Заголовочный файл |
| #pragma once  #define HASHDEL (void\*) -1  struct Object  {  void\*\* data;  Object(int, int(\*)(void\*));  int size; int N;  int(\*getKey)(void\*);  bool insert(void\*);  int searchInd(int key);  void\* search(int key);  void\* deleteByKey(int key);  bool deleteByValue(void\*);  void scan(void(\*f)(void\*));  };  static void\* DEL = (void\*)HASHDEL;  Object create(int size, int(\*getkey)(void\*));  #undef HASHDEL |
| Консоль |
|  |

Вариант 3

Изменить функцию вычисления хеш на мультипликативную функцию, которая строится на основе формулы: H(key) = [hashTableSize(key ∙ A mod 1)], где A = (sqrt(5) - 1) / 2 = 0,6180339887499.

|  |
| --- |
| Исходный код |
| Главный файл |
| #include <iostream>  #include "Header.h"  using namespace std;  struct AAA  {  int key;  const char\* mas;  AAA(int k, const char\* z)  {  key = k; mas = z;  } AAA() {}  };  //-------------------------------  int key(void\* d)  {  AAA\* f = (AAA\*)d; return f->key;  }  //-------------------------------  void AAA\_print(void\* d)  {  cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;  }  //-------------------------------  int main()  {  setlocale(LC\_ALL, "ru");  AAA a1(1, "one"), a2(2, "two"), a3(4, "three"), a4(2, "four");  int siz = 10, choice, k;  cout << "Введите размер хеш-таблицы" << endl; cin >> siz;  Object H = create(siz, key);  for (;;)  {  cout << "1 - вывод хеш-таблицы" << endl;  cout << "2 - добавление элемента" << endl;  cout << "3 - удаление элемента" << endl;  cout << "4 - поиск элемента" << endl;  cout << "0 - выход" << endl;  cout << "сделайте выбор" << endl; cin >> choice;  switch (choice)  {  case 0: exit(0);  case 1: H.scan(AAA\_print); break;  case 2: { AAA\* a = new AAA;  char\* str = new char[20];  cout << "введите ключ" << endl; cin >> k;  a->key = k;  cout << "введите строку" << endl; cin >> str;  a->mas = str;  if (H.N == H.size)  cout << "Таблица заполнена" << endl;  else  H.insert(a);  } break;  case 3: { cout << "введите ключ для удаления" << endl;  cin >> k;  H.deleteByKey(k);  } break;  case 4: { cout << "введите ключ для поиска" << endl;  cin >> k;  if (H.search(k) == NULL)  cout << "Элемент не найден" << endl;  else  AAA\_print(H.search(k));  } break;  }  }  return 0;  } |
| Файл с функциями |
| #include "Header.h"  #include <iostream>  // При мультипликативном методе (multiplicative method) используется случайное действительное число f в  // диапазоне от 0<f<1. Дробная часть произведения f \* key лежит в диапазоне от 0 до 1. Если это произведение  // умножить на n (размер хеш-таблицы), то целая часть полученного произведения даст значение хеш-функции в диапазоне 0...n-1.  int Multiplicative(int key, int size, int p)  {  int A;  double buf;  A = (sqrt(5) - 1) / 2;  int key2 = size \* (modf(key \* A, &buf));  return (key2 + p) % size;  }  //-------------------------------  int Next\_hash(int hash, int size, int p) // для след  {  return (hash + 5 \* p + 3 \* p \* p) % size;  }  //-------------------------------  Object create(int size, int(\*getkey)(void\*))  {  return \*(new Object(size, getkey));// выделение памяти  }  //-------------------------------  Object::Object(int size, int(\*getkey)(void\*))  {  N = 0;  this->size = size;  this->getKey = getkey;  this->data = new void\* [size];  for (int i = 0; i < size; ++i)  data[i] = NULL;  }  //-------------------------------  bool Object::insert(void\* d)// добавление элементов  {  bool b = false;  if (N != size)  for (int i = 0, t = getKey(d), j = Multiplicative(t, size, 0);  i != size && !b; j = Next\_hash(j, size, ++i))  if (data[j] == NULL || data[j] == DEL)  {  data[j] = d;// заполняем поля  N++;  b = true;  }  return b;  }  //-------------------------------  int Object::searchInd(int key)// функция для поиска по ключу  {  int t = -1;  bool b = false;// переменная для поиска  if (N != 0)  // поиск будет происходить, пока не найдем элемент и b не станет равно true  for (int i = 0, j = Multiplicative(key, size, 0); data[j] != NULL && i != size && !b; j = Multiplicative(key, size, ++i))  if (data[j] != DEL)  if (getKey(data[j]) == key)  {  t = j; b = true;  }  return t;  }  //-------------------------------  void\* Object::search(int key)// вывод результата поиска  {  int t = searchInd(key);  return(t >= 0) ? (data[t]) : (NULL);  }  //-------------------------------  void\* Object::deleteByKey(int key) // функция для удаления  {  int i = searchInd(key); // ищем элемент  void\* t = data[i]; // присваиваем указатель на этот элемент  if (t != NULL)  {  data[i] = DEL; // удаляем  N--;  }  return t;  }  //-------------------------------  bool Object::deleteByValue(void\* d)  {  return(deleteByKey(getKey(d)) != NULL);  }  //-------------------------------  void Object::scan(void(\*f)(void\*)) // вывод элементов  {  for (int i = 0; i < this->size; i++) // перебираем с помощью цикла  {  std::cout << " Элемент " << i; // и выводим  if ((this->data)[i] == NULL)  std::cout << " Пусто!" << std::endl;  else  if ((this->data)[i] == DEL)  std::cout << " Удален" << std::endl;  else  f((this->data)[i]);  }  } |
| Заголовочный файл |
| #pragma once  #define HASHDEL (void\*) -1  struct Object  {  void\*\* data;  Object(int, int(\*)(void\*));  int size; int N;  int(\*getKey)(void\*);  bool insert(void\*);  int searchInd(int key);  void\* search(int key);  void\* deleteByKey(int key);  bool deleteByValue(void\*);  void scan(void(\*f)(void\*));  };  static void\* DEL = (void\*)HASHDEL;  Object create(int size, int(\*getkey)(void\*));  #undef HASHDEL |
| Вывод в консоль |
|  |

Вариант 6

Использовать в проекте функции мультипликативного и модульного хеширования. Сравнить время поиска информации.

|  |
| --- |
| Исходный код |
| Главный файл |
| #include "Header.h"  #include <iostream>  using namespace std;  struct AAA  {  int key;  char\* mas;  AAA(int k, char\* z)  {  key = k; mas = z;  } AAA() {}  };  //-------------------------------  int key(void\* d)  {  AAA\* f = (AAA\*)d; return f->key;  }  //-------------------------------  void AAA\_print(void\* d)  {  cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;  }  //-------------------------------  int main()  {  setlocale(LC\_ALL, "rus");  AAA a1(1, (char\*)"one"), a2(2, (char\*)"two"), a3(4, (char\*)"three"), a4(2, (char\*)"fo");  int siz = 10, choice, k;  cout << "Введите размер хеш-таблицы" << endl; cin >> siz;  Object H = create(siz, key);  for (;;)  {  cout << "1 - вывод хеш-таблицы" << endl;  cout << "2 - добавление элемента мультипликативным методом" << endl;  cout << "3 - добавление элемента методом модульного деления" << endl;  cout << "4 - удаление элемента" << endl;  cout << "5 - поиск элемента" << endl;  cout << "0 - выход" << endl;  cout << "сделайте выбор" << endl; cin >> choice;  switch (choice)  {  case 0: exit(0);  case 1: H.scan(AAA\_print); break;  case 2: { AAA\* a = new AAA;  char\* str = new char[20];  cout << "введите ключ" << endl; cin >> k;  a->key = k;  cout << "введите строку" << endl; cin >> str;  a->mas = str;  if (H.N == H.size)  cout << "Таблица заполнена" << endl;  else  H.insertMul(a);  } break;  case 3: { AAA\* a = new AAA;  char\* str = new char[20];  cout << "введите ключ" << endl; cin >> k;  a->key = k;  cout << "введите строку" << endl; cin >> str;  a->mas = str;  if (H.N == H.size)  cout << "Таблица заполнена" << endl;  else  H.insertMod(a);  } break;  case 4: { cout << "введите ключ для удаления" << endl;  cin >> k;  H.deleteByKey(k);  } break;  case 5: { cout << "введите ключ для поиска" << endl;  cin >> k;  if (H.search(k) == NULL)  cout << "Элемент не найден" << endl;  else  AAA\_print(H.search(k));  } break;  }  }  return 0;  } |
| Файл с функциями |
| #include "Header.h"  #include <iostream>  int HashFunction(int key, int size, int p)  {  double key2 = 5 \* ((0.6180339887499 \* key) - int((0.6180339887499 \* key)));  return (p + key) % size;  }  int Multiplicative(int key, int size, int p)  {  double const c = 0.6180339887499;  return floor(size \* (key \* c - floor(key \* c)));  }  //-------------------------------  int Modul(int key, int size, int p)  {  return key % size;  }  //-------------------------------  int Next\_hash(int hash, int size, int p)  {  return (hash + 5 \* p + 3 \* p \* p) % size;  }  //-------------------------------  Object create(int size, int(\*getkey)(void\*))  {  return \*(new Object(size, getkey));// выделение памяти  }  //-------------------------------  Object::Object(int size, int(\*getkey)(void\*))  {  N = 0;  this->size = size;  this->getKey = getkey;  this->data = new void\* [size];  for (int i = 0; i < size; ++i)  data[i] = NULL;  }  //-------------------------------  bool Object::insertMul(void\* d)// добавление элементов  {  bool b = false;  if (N != size)  for (int i = 0, t = getKey(d), j = Multiplicative(t, size, 0);  i != size && !b; j = Next\_hash(j, size, ++i))  if (data[j] == NULL || data[j] == DEL)  {  data[j] = d;// заполняем поля  N++;  b = true;  }  return b;  }  //-------------------------------  bool Object::insertMod(void\* d)// добавление элементов  {  bool b = false;  if (N != size)  for (int i = 0, t = getKey(d), j = Modul(t, size, 0);  i != size && !b; j = Next\_hash(j, size, ++i))  if (data[j] == NULL || data[j] == DEL)  {  data[j] = d; // заполняем поля  N++;  b = true;  }  return b;  }  //-------------------------------  int Object::searchInd(int key)// функция для поиска по ключу  {  int t = -1;  bool b = false;// переменная для поиска  if (N != 0)  // поиск будет происходить, пока не найдем элемент и b не станет равно true  for (int i = 0, j = Multiplicative(key, size, 0); data[j] != NULL && i != size && !b; j = Multiplicative(key, size, ++i))  if (data[j] != DEL)  if (getKey(data[j]) == key)  {  t = j; b = true;  }  return t;  }  //-------------------------------  void\* Object::search(int key) // вывод результата поиска  {  int t = searchInd(key);  return(t >= 0) ? (data[t]) : (NULL);  }  //-------------------------------  void\* Object::deleteByKey(int key) // функция для удаления  {  int i = searchInd(key); // ищем элемент  void\* t = data[i]; // присваиваем указатель на этот элемент  if (t != NULL)  {  data[i] = DEL; // удаляем  N--;  }  return t;  }  //-------------------------------  bool Object::deleteByValue(void\* d)  {  return(deleteByKey(getKey(d)) != NULL);  }  //-------------------------------  void Object::scan(void(\*f)(void\*)) // вывод элементов  {  for (int i = 0; i < this->size; i++) // перебираем с помощью цикла  {  std::cout << " Элемент " << i; // и выводим  if ((this->data)[i] == NULL)  std::cout << " Пусто!" << std::endl;  else  if ((this->data)[i] == DEL)  std::cout << " Удален" << std::endl;  else  f((this->data)[i]);  }  } |
| Заголовочный файл |
| #pragma once  #define HASHDEL (void\*) -1  struct Object  {  void\*\* data;  Object(int, int(\*)(void\*));  int size; int N;  int(\*getKey)(void\*);  bool insertMul(void\*);  bool insertMod(void\*);  int searchInd(int key);  void\* search(int key);  void\* deleteByKey(int key);  bool deleteByValue(void\*);  void scan(void(\*f)(void\*));  };  static void\* DEL = (void\*)HASHDEL;  Object create(int size, int(\*getkey)(void\*));  #undef HASHDEL |
| Вывод в консоль |
|  |

Вариант 9

Изменить хеш-функцию в проекте на модульную.

|  |
| --- |
| Исходный код |
| Главная функция |
| #include "Header.h"  #include <iostream>  using namespace std;  struct AAA  {  int key;  char\* mas;  AAA(int k, char\* z)  {  key = k; mas = z;  } AAA() {}  };  //-------------------------------  int key(void\* d)  {  AAA\* f = (AAA\*)d; return f->key;  }  //-------------------------------  void AAA\_print(void\* d)  {  cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;  }  //-------------------------------  int main()  {  setlocale(LC\_ALL, "rus");  AAA a1(1, (char\*)"one"), a2(2, (char\*)"two"), a3(4, (char\*)"three"), a4(2, (char\*)"fo");  int siz = 10, choice, k;  cout << "Введите размер хеш-таблицы" << endl; cin >> siz;  Object H = create(siz, key);  for (;;)  {  cout << "1 - вывод хеш-таблицы" << endl;  cout << "2 - добавление элемента" << endl;  cout << "3 - удаление элемента" << endl;  cout << "4 - поиск элемента" << endl;  cout << "0 - выход" << endl;  cout << "сделайте выбор" << endl; cin >> choice;  switch (choice)  {  case 0: exit(0);  case 1: H.scan(AAA\_print); break;  case 2: { AAA\* a = new AAA;  char\* str = new char[20];  cout << "введите ключ" << endl; cin >> k;  a->key = k;  cout << "введите строку" << endl; cin >> str;  a->mas = str;  if (H.N == H.size)  cout << "Таблица заполнена" << endl;  else  H.insert(a);  } break;  case 3: { cout << "введите ключ для удаления" << endl;  cin >> k;  H.deleteByKey(k);  } break;  case 4: { cout << "введите ключ для поиска" << endl;  cin >> k;  if (H.search(k) == NULL)  cout << "Элемент не найден" << endl;  else  AAA\_print(H.search(k));  } break;  }  }  return 0;  } |
| Файл с функциями |
| #include "Header.h"  #include <iostream>  int HashFunction(int key, int size, int p)  {  return key % size;  }  //-------------------------------  int Next\_hash(int hash, int size, int p)  {  return (hash + 5 \* p + 3 \* p \* p) % size;  }  //-------------------------------  Object create(int size, int(\*getkey)(void\*))  {  return \*(new Object(size, getkey));  }  //-------------------------------  Object::Object(int size, int(\*getkey)(void\*))  {  N = 0;  this->size = size;  this->getKey = getkey;  this->data = new void\* [size];  for (int i = 0; i < size; ++i)  data[i] = NULL;  }  //-------------------------------  bool Object::insert(void\* d)  {  bool b = false;  if (N != size)  for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0);  i != size && !b; j = Next\_hash(j, size, ++i))  if (data[j] == NULL || data[j] == DEL)  {  data[j] = d;  N++;  b = true;  }  return b;  }  //-------------------------------  int Object::searchInd(int key)  {  int t = -1;  bool b = false;  if (N != 0)  for (int i = 0, j = HashFunction(key, size, 0); data[j] != NULL && i != size && !b; j = HashFunction(key, size, ++i))  if (data[j] != DEL)  if (getKey(data[j]) == key)  {  t = j; b = true;  }  return t;  }  //-------------------------------  void\* Object::search(int key)  {  int t = searchInd(key);  return(t >= 0) ? (data[t]) : (NULL);  }  //-------------------------------  void\* Object::deleteByKey(int key)  {  int i = searchInd(key);  void\* t = data[i];  if (t != NULL)  {  data[i] = DEL;  N--;  }  return t;  }  //-------------------------------  bool Object::deleteByValue(void\* d)  {  return(deleteByKey(getKey(d)) != NULL);  }  //-------------------------------  void Object::scan(void(\*f)(void\*))  {  for (int i = 0; i < this->size; i++)  {  std::cout << " Элемент " << i;  if ((this->data)[i] == NULL)  std::cout << " Пусто!" << std::endl;  else  if ((this->data)[i] == DEL)  std::cout << " Удален" << std::endl;  else  f((this->data)[i]);  }  } |
| Заголовочный файл |
| #pragma once  #define HASHDEL (void\*) -1  struct Object  {  void\*\* data;  Object(int, int(\*)(void\*));  int size; int N;  int(\*getKey)(void\*);  bool insert(void\*);  int searchInd(int key);  void\* search(int key);  void\* deleteByKey(int key);  bool deleteByValue(void\*);  void scan(void(\*f)(void\*));  };  static void\* DEL = (void\*)HASHDEL;  Object create(int size, int(\*getkey)(void\*));  #undef HASHDEL |
| Вывод в консоль |
|  |