Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 13

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные кучи»

Выполнила:

Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: асс. Андронова М.В.

2023, Минск

5.  В проект добавить следующие функции: удаление минимального **extractMin**; удаление i-ого элемента **extractI**; объединение **unionHeap** двух куч в одну.

|  |
| --- |
| Исходный код |
| Главный файл |
| #include "Header.h"  #include <iostream>  using namespace std;  using namespace heap;  heap::CMP cmpAAA(void\* a1, void\* a2) //Функция сравнения  {  #define A1 ((AAA\*)a1)  #define A2 ((AAA\*)a2)  heap::CMP  rc = heap::EQUAL;  if (A1->x > A2->x)  rc = heap::GREAT;  else  if (A2->x > A1->x)  rc = heap::LESS;  return rc;  #undef A2  #undef A1  }  int main()  {  setlocale(LC\_ALL, "rus");  int k = 1, choice;  heap::Heap h1 = heap::create(30, cmpAAA);  heap::Heap h2 = heap::create(30, cmpAAA);  Heap\* ph2 = &h2;  for (;;)  {  cout << "1 - Вывод кучи на экран" << endl;  cout << "2 - Добавить элемент" << endl;  cout << "3 - Удалить максимальный элемент" << endl;  cout << "4 - Удалить минимальный элемент" << endl;  cout << "5 - Удалить i-ый элемент" << endl;  cout << "6 - Объединение двух бинарных куч" << endl;  cout << "0 - Выход" << endl;  cout << "Cделайте выбор:" << endl; cin >> choice;  switch (choice)  {  case 0: exit(0);  case 1: h1.scan(0);  break;  case 2:  while (k > 0)  {  AAA\* a = new AAA;  cout << "Введите ключ:" << endl;  cin >> k;  if (k > 0)  {  a->x = k;  h1.insert(a);  }  }  break;  case 3: h1.extractMax();  break;  case 4: h1.extractMin();  break;  case 5:  int i;  cout << "Введите i:\n";  cin >> i;  h1.extractI(i);  break;  case 6:  int j = 1;  cout << "Введите вторую кучу:\n";  while (j > 0)  {  AAA\* b = new AAA;  cout << "Введите ключ:" << endl; cin >> j;  if (j > 0)  {  b->x = j;  h2.insert(b);  }  }  h1.unionHeap(ph2);  break;  }  } return 0;  } |
| Файл с функциями |
| #include "Header.h"  #include <iostream>  #include <iomanip>  void AAA::print()  {  std::cout << x;  }  int AAA::getPriority() const  {  return x;  }  namespace heap  {  Heap create(int maxsize, CMP(\*f)(void\*, void\*)) // создание кучи  {  return \*(new Heap(maxsize, f)); // выделение памяти  }  int Heap::left(int ix)  {  return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1); // если левый больше размера, то возвращаем -1, иначе возвращаем левого  }  int Heap::right(int ix)  {  return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2); // то же самое  }  int Heap::parent(int ix)  {  return (ix + 1) / 2 - 1; // для каждого родителя  }  void Heap::swap(int i, int j) // сортировка  {  void\* buf = storage[i]; // вспомогательная переменная  storage[i] = storage[j]; // обмен данными  storage[j] = buf;  }  void Heap::heapify(int ix) // записб кучи в виде дерева каждый дочерний меньше или равен родительскому  {  int l = left(ix), r = right(ix), irl = ix; // левый, правый дочерние и корень  if (l > 0)  {  if (isGreat(storage[l], storage[ix])) irl = l;  if (r > 0 && isGreat(storage[r], storage[irl])) irl = r;  if (irl != ix)  {  swap(ix, irl);  heapify(irl);  }  }  }  void Heap::insert(void\* x) // добавление элемента  {  int i;  if (!isFull())  {  storage[i = ++size - 1] = x;  while (i > 0 && isLess(storage[parent(i)], storage[i]))  {  swap(parent(i), i);  i = parent(i);  }  }  }  void\* Heap::extractMax() // ищем максимальный и удаляем  {  void\* rc = nullptr;  if (!isEmpty())  {  rc = storage[0];  storage[0] = storage[size - 1];  size--;  heapify(0);  } return rc;  }  void Heap::scan(int i) const // вывод на экран  {  int probel = 20;  std::cout << '\n';  if (size == 0)  std::cout << "Пусто";  for (int u = 0, y = 0; u < size; u++)  {  std::cout << std::setw(probel + 10) << std::setfill(' ');  ((AAA\*)storage[u])->print();  if (u == y)  {  std::cout << '\n';  if (y == 0)  y = 2;  else  y += y \* 2;  }  probel /= 2;  }  std::cout << '\n';  }  void\* Heap::extractMin() // ищем минимальный и удаляем  {  void\* rc = nullptr;  if (!isEmpty())  {  rc = storage[size - 1];  storage[size - 1] = storage[0];  size--;  heapify(size - 1);  } return rc;  }  void\* Heap::extractI(int i) // ищем i-й и удаляем  {  void\* rc = nullptr;  if (!isEmpty())  {  rc = storage[i];  storage[i] = storage[size - 1];  size--;  heapify(i);  } return rc;  }  void Heap::unionHeap(Heap\* h2) // объединение куч  {  void\* rc = nullptr;  for (int i = 0; i < h2->size; i++)  {  rc = h2->storage[i]; // перемещаем указатель  insert(rc); // и добавляем  }  heapify(0);  }  } |
| Заголовочный файл |
| #pragma once  struct AAA  {  int x;  void print();  int getPriority() const;  };  namespace heap  {  enum CMP  {  LESS = -1, EQUAL = 0, GREAT = 1  };  struct Heap  {  int size;  int maxSize; // максимальный размер  void\*\* storage; // данные  CMP(\*compare)(void\*, void\*);  Heap(int maxsize, CMP(\*f)(void\*, void\*))  {  size = 0;  storage = new void\* [maxSize = maxsize];  compare = f;  };  int left(int ix);  int right(int ix);  int parent(int ix);  bool isFull() const  {  return (size >= maxSize);  };  bool isEmpty() const  {  return (size <= 0);  };  bool isLess(void\* x1, void\* x2) const  {  return compare(x1, x2) == LESS;  };  bool isGreat(void\* x1, void\* x2) const  {  return compare(x1, x2) == GREAT;  };  bool isEqual(void\* x1, void\* x2) const  {  return compare(x1, x2) == EQUAL;  };  void swap(int i, int j);  void heapify(int ix);  void insert(void\* x);  void\* extractMax();  void\* extractMin();  void\* extractI(int i);  void scan(int i) const;  void unionHeap(Heap\* h2);  };  Heap create(int maxsize, CMP(\*f)(void\*, void\*));  }; |
| Консоль |
|  |

Дополнительные задания

1. Пусть дано бинарное дерево, необходимо преобразовать его в бинарную кучу.

|  |
| --- |
| Исходный код |
| // Преобразование двоичного дерева поиска в минимальную кучу  #include <iostream>  #include <vector>  #include <queue>  #include <string>  #include <utility>  #include <iomanip>  using namespace std;  // Структура данных для хранения узла бинарного дерева  struct Node  {  int data;  Node\* left = nullptr, \* right = nullptr;  Node() {}  Node(int data) : data(data) {}  };  // Рекурсивная функция для вставки ключа в BST  Node\* insert(Node\* root, int key)  {  // если корень нулевой, создаем новый узел и возвращаем его  if (root == nullptr) {  return new Node(key);  }  // если заданный ключ меньше корневого узла, повторить для левого поддерева  if (key < root->data) {  root->left = insert(root->left, key);  }  // если данный ключ больше, чем корневой узел, повторить для правого поддерева  else {  root->right = insert(root->right, key);  }  return root;  }  // Вспомогательная функция для обхода бинарного дерева по порядку уровней  void printLevelOrderTraversal(Node\* root)  {  int probel = 20;  std::cout << '\n';  // базовый случай: пустое дерево  if (root == nullptr) {  return;  }  queue<Node\*> q;  q.push(root);  while (!q.empty())  {    int n = q.size();  while (n--)  {  std::cout << std::setw(probel + 10) << std::setfill(' ');  Node\* front = q.front();  q.pop();  cout << front->data << ' ';  if (front->left) {  q.push(front->left);  }  if (front->right) {  q.push(front->right);  }  probel /= 2;  }  cout << endl;  }  }  // Функция для выполнения неупорядоченного обхода заданного бинарного дерева и  // поставить в queue все узлы (в порядке их появления)  void inorder(Node\* root, queue<int>& keys)  {  if (root == nullptr) {  return;  }  inorder(root->left, keys);  keys.push(root->data);  inorder(root->right, keys);  }  // Функция для обхода заданного бинарного дерева в прямом порядке.  // Назначаем каждому встреченному узлу следующий ключ из queue  void preorder(Node\* root, queue<int>& keys)  {  // базовый случай: пустое дерево  if (root == nullptr) {  return;  }  // заменяем значение корневого ключа на следующий ключ из queue  root->data = keys.front();  keys.pop();  // обрабатываем левое поддерево  preorder(root->left, keys);  // обрабатываем правое поддерево  preorder(root->right, keys);  }  // Функция для преобразования BST в минимальную кучу  void convert(Node\* root)  {  // базовый вариант  if (root == nullptr) {  return;  }  // поддерживаем queue для хранения неупорядоченного обхода дерева  queue<int> keys;  // заполняем queue по порядку  inorder(root, keys);  // обход дерева в прямом порядке и для каждого встреченного узла  // извлекаем ключ из очереди и назначаем его узлу  preorder(root, keys);  }  int main()  {  setlocale(LC\_ALL, "rus");  vector<int> keys = { 15, 18, 9, 10, 8, 19, 17 }; // вектор со значениями, которые будут образующими для дерева и кучи  cout << "Преобразование дерева со значениями: ";  Node\* root = nullptr;  for (int key : keys) {  cout << key << " ";  root = insert(root, key);  }  convert(root);  printLevelOrderTraversal(root);  return 0;  } |
| Вывод в консоль |
|  |

1. Дано дерево, определить, является она минимальной кучей или нет.

|  |
| --- |
| Исходный код |
| // является ли бинарное дерево минимальной кучей или нет  #include <iostream>  #include <vector>  #include <queue>  #include <iomanip>  using namespace std;  // Структура данных для хранения узла бинарного дерева  struct Node  {  int data;  Node\* left, \* right;  Node(int data)  {  this->data = data;  this->left = this->right = nullptr;  }  };  // Функция для нахождения общего количества узлов в бинарном дереве  int size(Node\* root)  {  if (root == nullptr) {  return 0;  }  return 1 + size(root->left) + size(root->right);  }  // Функция для проверки, является ли данное бинарное дерево полным бинарным деревом  // и каждый узел имеет более высокое значение, чем его родитель  bool isHeap(Node\* root, int i, int n)  {  // базовый вариант  if (root == nullptr) {  return true;  }  // неполное бинарное дерево: вне допустимого диапазона индекса  if (i >= n) {  return false;  }  // текущий узел имеет большее значение, чем его левый или правый потомок  if ((root->left && root->left->data <= root->data) ||  (root->right && root->right->data <= root->data)) {  return false;  }  // проверка левого и правого поддерева  return isHeap(root->left, 2 \* i + 1, n) && isHeap(root->right, 2 \* i + 2, n);  }  // Функция для проверки того, является ли заданное бинарное дерево минимальной кучей или нет  bool isHeap(Node\* root)  {  int i = 0;  return isHeap(root, i, size(root));  }  int main()  {  int data1, datastorage[7];  setlocale(LC\_ALL, "rus");  cout << "Элементы для кучи: ";  cin >> data1;  datastorage[0] = data1;  Node\* root = new Node(data1); // выделяем память под новые узлы  cin >> data1;  datastorage[1] = data1;  root->left = new Node(data1);  cin >> data1;  datastorage[2] = data1;  root->right = new Node(data1);  cin >> data1;  datastorage[3] = data1;  root->left->left = new Node(data1);  cin >> data1;  datastorage[4] = data1;  root->left->right = new Node(data1);  cin >> data1;  datastorage[5] = data1;  root->right->left = new Node(data1);  cin >> data1;  datastorage[6] = data1;  root->right->right = new Node(data1);  cout << "дерево состоит из элементов: ";  for (int i = 0; i < 7; i++) {  cout << datastorage[i] << " ";  }  if (isHeap(root)) {  cout << "\nбинарное дерево является минимальной кучей\n";  }  else {  cout << "\nбинарное дерево минимальной кучей не является\n";  }  return 0;  } |
| Вывод в консоль |
|  |

1. Дан динамический массив. Определить составляет ли он минимальную кучу или нет.

|  |
| --- |
| Исходный код |
| // представляет ли массив минимальную бинарную кучу  #include <iostream>  #include <vector>  using namespace std;  // Функция для проверки, представляет ли данный массив минимальную кучу или нет  bool checkMinHeap(vector<int> const A, int i)  {  // если `i` является листовым узлом, вернуть true, так как каждый листовой узел является кучей  if (2 \* i + 2 > A.size()) {  return true;  }  // если `i` является внутренним узлом  // рекурсивно проверяем, является ли левый потомок кучей  bool left = (A[i] <= A[2 \* i + 1]) && checkMinHeap(A, 2 \* i + 1);  // рекурсивно проверяем, является ли правый потомок кучей (чтобы избежать вывода индекса массива  // границ, сначала проверьте, существует ли правильный дочерний элемент или нет)  bool right = (2 \* i + 2 == A.size()) ||  (A[i] <= A[2 \* i + 2] && checkMinHeap(A, 2 \* i + 2));  // вернуть true, если и левый, и правый дочерние элементы являются кучами  return left && right;  }  int main()  {  setlocale(LC\_ALL, "rus");  vector<int> A = { 1, 2, 3, 4, 5, 6 };  vector<int> A1 = { 6, 5, 4, 3, 2, 1 };  // начинаем с индекса 0 (корень кучи)  int index = 0;  if (checkMinHeap(A, index)) {  for (auto item : A) cout << item << " ";  cout << "\nМассив представляет собой минимальную кучу\n";  }  else {  for (auto item : A) cout << item << " ";  cout << "\nМассив не представляет собой минимальную кучу\n";  }  if (checkMinHeap(A1, index)) {  for (auto item : A1) cout << item << " ";  cout << "\nМассив представляет собой минимальную кучу\n";  }  else {  for (auto item : A1) cout << item << " ";  cout << "\nМассив не представляет собой минимальную кучу\n";  }  return 0;  } |
| Вывод в консоль |
|  |