**Отчет**

1. Название структуры данных: Фибоначчиева куча.

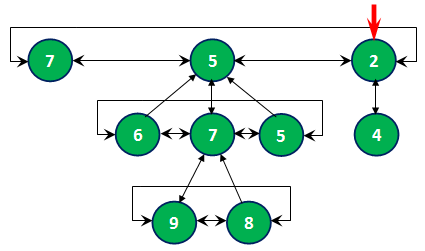
Краткая историческая справка:

* Фибоначчиева куча (англ. Fibonacci heap) — структура данных, представляющая собой набор деревьев, упорядоченных в соответствии со свойством неубывающей пирамиды. Фибоначчиевы кучи были введены Майклом Фредманом и Робертом Тарьяном в 1984 году.

1. Основной принцип устройства. Особенности.

* В отличие от биномиальных деревьев, здесь дети любого узла могут записываться в любом порядке.
* Доступ к куче производится ссылкой на узел с минимальным ключом
* Каждый узел фибоначчиева дерева, помимо указателей на левого, правого брата, на родителя и на одного из своих сыновей содержит информацию о количестве дочерних узлов.)
* *Фибоначчиева куча представляет собой набор фибоначчиевых деревьев*
* (прим. Фибоначчиево дерево – это k-ичное дерево, для каждого элемента которого выполняется правило: «дочерний элемент не превышает своего родителя». )
* Корни фибоначчиевых деревьев хранятся в виде кольцевого списка.

Её пример:



1. Оценка временной сложности (с обоснованием)

К сожалению, скрытые константы в асимптотических оценках трудоемкости велики и использование фибоначчиевых куч редко оказывается целесообразным: обычные двоичные кучи на практике эффективнее. С практической точки зрения желательно придумать структуру данных с теми же асимптотическими оценками, но с меньшими константами.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

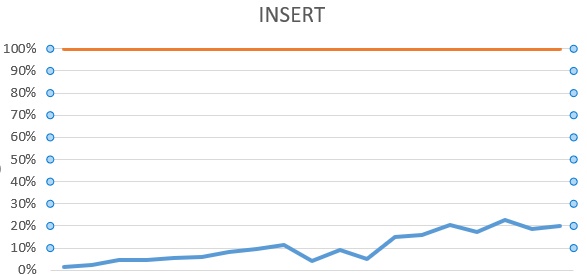
Автоматически созданное описание

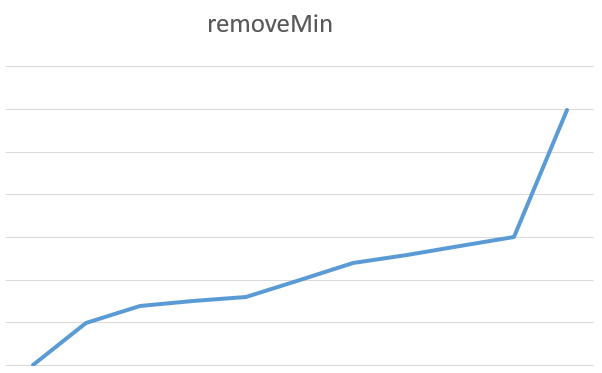
Изображение выглядит как текст

Автоматически созданное описание

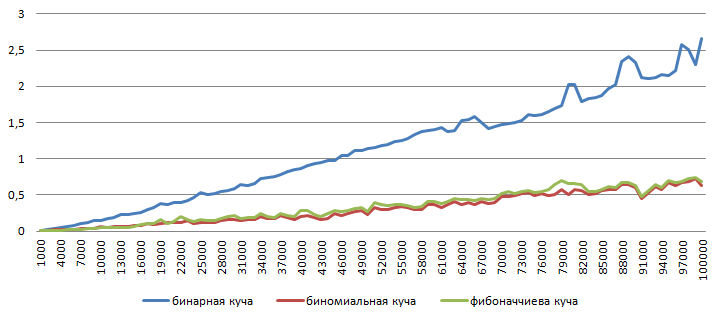
Основными операциями в данной структуре данных являются insert, merge, extractMin / removeMin, (decreaseKey)

1. Графики:





**MERGE(Красная полоса)**



1. Выводы. Плюсы и минусы структуры данных, ее применимость.

**Недостатки:**

Большое потребление памяти на узел

Большая константа времени работы, что делает ее малоприменимой для реальных задач.

Некоторые операции в худшем случае могут работать за O(n) времени

**Достоинства**:

Одно из лучших асимптотических времен работы для всех операций

Существует много задач, где применяется работа с графами. При такой работе более целесообразно использовать фибоначчиевы кучи.

При помощи фибоначчиевых куч можно легко проводить сортировку, удалять, добавлять,  уменьшать ключи, вершины, элементы.

**Применимость(Области применения):**

Кучи являются основной структурой данных во многих приложениях. В том числе, они применяются:

* при сортировке элементов;
* в алгоритмах выбора, для поиска минимума и/или максимума, медианы;
* в алгоритмах на графах, в частности, при построении минимального остовного дерева алгоритмом Крускала (Joseph Kruskal), при нахождении кратчайшего пути алгоритмом Дейкстры (Edsger W. Dijkstra).

1. Источники:

* Статья "Приоритетная очередь на основе бинарной, биномиальной и фибонначиевой куч": <https://www.rsdn.org/article/submit/heap/heaps.xml#ERFAE>
* Статья от ИТМО:[**https://neerc.ifmo.ru/wiki/index.php?title=Фибоначчиева\_куча**](https://neerc.ifmo.ru/wiki/index.php?title=Фибоначчиева_куча)
* [https://ru.wikipedia.org](https://ru.wikipedia.org/) "Фибоначчиева куча"
* Лекция(устройство, особенности): <https://www.youtube.com/watch?v=IXcdJuRbqPU&t=3530s>



import java.util.ArrayList;  
import java.util.List;  
  
public class FibonacciHeap {  
  
 private Node min;  
  
 private int n;  
  
 public boolean isEmpty() { return min == null; }  
  
 public Node getMin() { return min; }  
  
 public int getSize() { return n; }  
  
 public void clean() {  
 for (int i = 0; i < n; i++) {  
 removeMin();  
 }  
 }  
  
 public FibonacciHeap merge(FibonacciHeap H1, FibonacciHeap H2) {  
 FibonacciHeap H = new FibonacciHeap();  
 H.min = null;  
 H.n = 0;  
 if(H1 == null && H2 == null) return H;  
 if(H1 != null) {  
 H.min = H1.min;  
 H.n = H1.n;  
 if(H2!=null) {  
 Node R1 = H1.min;  
 Node R2 = H1.min.getRight();  
  
 Node L1 = H2.min;  
 Node L2 = H2.min.getRight();  
  
 R1.setLeft(L1);;  
 L1.setLeft(R1);  
  
 R2.setRight(L2);  
 L2.setRight(R2);  
  
 if(H1.min.getKey() > H2.min.getKey()) {  
 H.min = H2.min;  
 }  
 }  
 } else {  
 H.min = H2.min;  
 H.n = H2.n;  
 }  
 return H;  
 }  
  
 public Object removeMin() {  
 Node z = min;  
 if(z==null) return null;  
  
 Node c = z.getChild();  
 if(c!=null) {  
 c.setParent(null);  
 for(Node x = c.getRight(); x!=c; x = x.getRight()) {  
 x.setParent(null);  
 }  
  
 Node R1 = c;  
 Node R2 = c.getRight();  
  
 Node L1 = min;  
 Node L2 = min.getRight();  
  
 R1.setLeft(L1);;  
 L1.setLeft(R1);  
  
 R2.setRight(L2);  
 L2.setRight(R2);  
 }  
  
 z.getLeft().setRight(z.getRight());  
 z.getRight().setLeft(z.getLeft());  
  
 if(z == z.getRight()) {  
 min = null;  
 } else {  
 min = z.getRight();  
 consolidate();  
 }  
 n--;  
 return z.getData();  
 }  
  
 public Node insert(Object data, Integer key) {  
 Node node = new Node(data, key);  
 if(min==null) {  
 min = node;  
 } else {  
 node.setRight(min.getRight());  
 node.setLeft(min);  
 min.getRight().setLeft(node);  
 min.setRight(node);  
 if(key < min.getKey()) {  
 min = node;  
 }  
 }  
 n++;  
 return node;  
 }  
  
  
 private void decreaseKey(Node x, Integer key, boolean del) {  
 if(key > x.getKey()) throw new IllegalArgumentException("Нельзя увеличить ключ");  
  
 Node y = x.getParent();  
 if(y!=null && (del || key < y.getKey())) {  
 y.cut(x, min);  
 y.cascadeCut(min);  
 }  
 if(del || key < min.getKey()) {  
 min = x;  
 }  
 }  
  
 public void decreaseKey(Node x, Integer key) {  
 decreaseKey(x, key, false);  
 }  
  
 public void delete(Node x) {  
 decreaseKey(x, Integer.*MIN\_VALUE*, true);  
 removeMin();  
 }  
  
  
 public void consolidate() {  
 Node[] A = new Node[45];  
 Node start = min;  
 Node w = min;  
  
 do {  
 Node x = w;  
 Node nextW = w.getRight();  
 int d = x.getDegree();  
 while(A[d]!=null) {  
 Node y = A[d];  
 if(x.getKey() > y.getKey()) {  
 Node tmp = x;  
 x = y;  
 y = tmp;  
 }  
 if(y==start) start = start.getRight();  
 if(y==nextW) nextW = nextW.getRight();  
  
 y.link(x);  
  
 A[d] = null;  
 d++;  
 }  
 A[d] = x;  
 w = nextW;  
 } while(w!=start);  
  
 min = start;  
  
 for(Node a : A) {  
 if(a!=null && a.getKey() < min.getKey()) {  
 min = a;  
 }  
 }  
 }  
}

import java.util.List;  
  
public class Node {  
  
 private Object data;  
 private Integer key;  
  
 private Node parent, child, left, right;  
  
 private int degree;  
  
 private boolean mark;  
  
 public Node(Object data, Integer key) {  
 this.data = data;  
 this.key = key;  
 left = this;  
 right = this;  
 }  
  
 public void link(Node parent) {  
 left.right = right;  
 right.left = left;  
  
 this.parent = parent;  
  
 if(parent.child == null) {  
 parent.child = this;  
 right = this;  
 left = this;  
 } else {  
 left = parent.child;  
 right = parent.child.right;  
 parent.child.right.left = this;  
 parent.child.right = this;  
 }  
  
 parent.degree++;  
 mark = false;  
 }  
  
 public void cut(Node removeChild, Node min) {  
 removeChild.left.right = removeChild.right;  
 removeChild.right.left = removeChild.left;  
  
 degree--;  
  
 if(degree == 0) {  
 child = null;  
 } else {  
 if(removeChild == child) child = removeChild.right;  
 }  
  
 removeChild.left = min.left;  
 removeChild.right = min;  
 min.left.right = removeChild;  
 min.left = removeChild;  
  
 removeChild.parent = null;  
 removeChild.mark = false;  
 }  
  
 public void cascadeCut(Node min) {  
 Node z = parent;  
 while (z!=null) {  
 if(mark) {  
 z.cut(this, min);  
 z.cascadeCut(min);  
 } else {  
 mark = true;  
 }  
 }  
 }  
  
 public Integer getKey() { return key; }  
  
 public Object getData() { return data; }  
  
 public Node getLeft() { return left; }  
  
 public Node getRight() { return right; }  
  
 public void setLeft(Node l) { left = l; }  
  
 public void setRight(Node r) { right = r; }  
  
 public Node getChild() { return child; }  
  
 public void setParent(Node p) { parent = p; }  
  
 public Node getParent() { return parent; }  
  
 public void setKey(Integer k) { key = k; }  
  
 public int getDegree() { return degree; }  
}

//использовал Junit

import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeAll;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.util.Random;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class StructTest2 {  
  
 private static final Long *SEED* = 123123L;  
 private static final Random *RAND* = new Random(*SEED*);  
  
 private static int rand(){  
 return *RAND*.nextInt();  
 }  
  
 private static Struct *testing*;  
  
 private static long *start*;  
  
 private static final long *I* = 500;  
  
 private void fill(int k) throws InterruptedException {  
 for(int i = 0; i < k; i++){  
 *testing*.insert(*rand*());  
 }  
 }  
  
 private void prepare(String method, int k) throws InterruptedException {  
 System.*out*.print("testing " + method + " " + k + " : ");  
 fill(k);  
 *start* = System.*nanoTime*();  
 }  
  
 @BeforeAll  
 static void prepare() {  
 *testing* = new Struct();  
 }  
  
 @BeforeEach  
 void beforeEach(){  
 *testing* = new Struct();  
 *testing*.clean();  
 *start* = System.*nanoTime*();  
 }  
  
 @Test  
 void insert10000() throws InterruptedException {  
 int j = 10000;  
 prepare("insert", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void insert20000() throws InterruptedException {  
 int j = 20000;  
 prepare("insert", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void insert30000() throws InterruptedException {  
 int j = 30000;  
 prepare("insert", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void insert40000() throws InterruptedException {  
 int j = 40000;  
 prepare("insert", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void contains10000() throws InterruptedException {  
 int j = 10000;  
 prepare("contains", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void contains20000() throws InterruptedException {  
 int j = 20000;  
 prepare("contains", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void contains30000() throws InterruptedException {  
 int j = 30000;  
 prepare("contains", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void contains40000() throws InterruptedException {  
 int j = 40000;  
 prepare("contains", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.contains(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void remove10000() throws InterruptedException {  
 int j = 10000;  
 prepare("remove", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.remove(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void remove20000() throws InterruptedException {  
 int j = 20000;  
 prepare("remove", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.remove(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void remove30000() throws InterruptedException {  
 int j = 30000;  
 prepare("remove", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.remove(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @Test  
 void remove40000() throws InterruptedException {  
 int j = 40000;  
 prepare("remove", j);  
 for(int i = 0; i < *I*; i++){  
 *testing*.remove(*rand*());  
 }  
 *assertEquals*(true, true);  
 }  
  
 @AfterEach  
 void time(){  
 long t = System.*nanoTime*() - *start*;  
 System.*out*.println(t);  
 }  
  
}

1. Ссылка на репозиторий:

https://github.com/w1nway/A-SD