ФГАОУ ВО «Санкт-Петербургский Политехнический университет Петра Великого»

Институт Прикладной Математики и Механики

Высшая школа прикладной математики и вычислительной физики

ПРОИЗВОДСТВЕННАЯ ПРАКТИКА

Фибоначчиева куча (Fibonacci heap)

Выполнил студент:

Чеботин А.А.
группа: 3630102/90003

Преподаватель:

Беляев С.Ю

Санкт-Петербург, 2021 г.

Содержание

Fibonacci heap 1

1.1. Формулировка задачи и ее формализация.

Реализовать операции добавить и забрать min, используя структуру данных "Фибоначчиева куча".

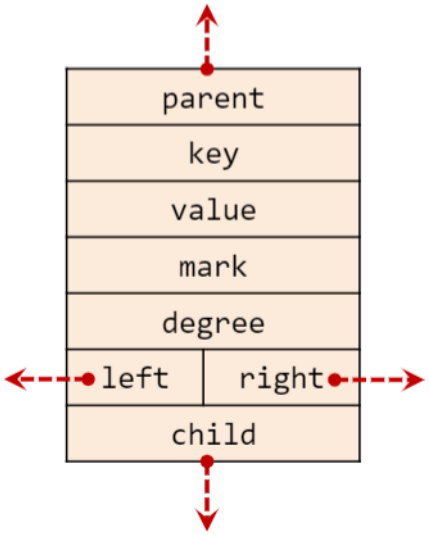
1.2. Описание решения

Фибоначчиева куча (англ. Fibonacci heap) – это совокупность деревьев, которые удовлетворяют свойствам кучи (min-heap или max-heap). В каждом дереве, входящих в данную кучу, выполнено следующее свойство: **ключ каждой вершины не больше ключей ее детей**.

1.2.1. Структура кучи

Каждый узел содержит следующие поля:

- key – приоритет узла (вес, ключ);
- parent – указатель на родительский узел;
- child – указатель на один из дочерних узлов;
- left – указатель на левый сестринский узел;
- right – указатель на правый сестринский узел;
- degree – количество дочерних узлов;
- mark – были ли потери узлом дочерних узлов.



Доступ к фибоначчиевой куче осуществляется по указателю на корень дерева с минимальным ключом. Поэтому в программной реализации структура фибоначчиевой кучи будет состоять из 1) числа узлов; 2) указателя на узел с минимальным ключом. Структура самого узла будет состоять из полей, описанных выше:

```
struct FibonacciHeap {
int number_of_nodes;
node* min;
}

struct node {
int key;
node* parent;
node* child;
node* left;
node* right;
int degree;
bool mark;
};
```

Листинг 1.1: Структура кучи

1.2.2. Добавление элемента

Данная операция (insert) вставляет новый элемент в список корней кучи и при необходимости будет менять указатель на min. Возвращает указатель на вновь созданный узел.

1.2.3. Забрать *min*

Операция забрать *min* (extract-min) не принимает аргументов и возвращает целочисленное значение минимального ключа.

В этой операции минимальный узел будет удален из корневого списка, а его дочерние элементы будут помещены в корневой список. После этой процедуры будут пройдены все узлы в корневом списке, и узлы с одинаковой степенью будут объединены. Эта операция будет продолжаться до тех пор, пока все узлы в корневом списке не будут иметь разные степени.

1.3. Альтернативные реализации

Фибоначчиевы кучи полезны, если количество операций extract-min мало по сравнению с остальными операциями. Есть биномиальный кучи и там идея аналогичная - несколько биномиальных деревьев связываются в список.

1.4. Затраты

Стоимость извлечения минимального узла равна $O(logn)$. Операция удаления - $O(logn)$. Операции, в которых не требуется удаление, имеют амортизированное время работы, равное $O(1)$, что намного лучше двоичных и биномиальных куч.

1.5. Код программы

Программа написана на языке C в среде разработки Visual Studio 2022.

```
struct FibonacciHeap {
int number_of_nodes;
node* min;
}

struct node {
int key;
node* parent;
node* child;
node* left;
node* right;
int degree;
bool mark;
};

void childs_in_root(node *Node) {
        while (Node->child) {
            Node->parent->degree--;
            Node->parent->child = Node->right == Node ? NULL : Node->
   right;
            if (Node->left != Node) {
```

```
21                  Node->right->left = Node->left;
22                  Node->left->right = Node->right;
23              }
24              add(Node, &min);
25              Node->mark = false;
26          }
27      }
28  void remove_root(node* Node) {
29          if (Node->left != Node) {
30              Node->right->left = Node->left;
31              Node->left->right = Node->right;
32          }
33          number_of_nodes--;
34      }
35
36  void merge() {
37          node* A[SIZE];
38          node* x = min;
39          int init_roots = number_of_nodes;
40          int max_degree = 0;
41          for (int i=0;i<init_roots; i++) {
42              int d = x->degree;
43              node* next = x -> right;
44              while (A[d]) {
45                  node* y = A[d];
46                  if (y->key<x->key)
47                      swap(x,y);
48                  remove_root(x);
49                  add(x, &y->child, y);
50                  x->mark = false;
51                  A[d++] = NULL;
52              }
53              A[d] = x;
54              max_degree = max(max_degree,d);
55              x = next;
56          }
57          min = NULL;
58          number_of_nodes = 0;
59          for (int i=0;i<=max_degree; i++) {
60              if (A[i]) {
61                  add(A[i],&min);
62              }
63          }
64      }
65
66  void insert(node* Node, node** brother, node* par = NULL) {
67          if (*brother == NULL) {
68              *brother = Node;
69              Node->left = Node;
70              Node->right = Node;
71          }
72          else {
73              Node->right = (*brother)->right;
```

```
74              Node->right->left = Node;
75              Node->left = *brother;
76              (*brother)->right = Node;
77          }
78          if (less(Node, *brother))
79              *brother = Node;
80
81          if(*brother == min) {
82              roots_amount++;
83              Node->parent = NULL;
84          }
85          if (par){
86              par->degree++;
87              Node->parent = par;
88          }
89      }
90
91  node* add(int key) {
92          node* Node = new node(key);
93          add(Node,&min);
94          return Node;
95      }
96
97  void extract_min() {
98          node* res = min;
99          if (res) {
100             childs_in_root(res);
101             remove_root(res);
102             if (res->right == res)
103                 min = 0;
104             else {
105                 min = min->right;
106                 merge();
107             }
108         }
109         delete res;
110     }
```

Листинг 1.2: Код программы

1.6. Источники

Конспект лекций курса "Алгоритмы и структуры данных"Беляев С.Ю.

Т. Кормен Алгоритмы: построение и анализ