Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання №8

“ Реалізація піраміди Фібоначчі”

Виконав студент 2-го курсу

Групи К-29

Жутовський Глiб Євгенович

2021

**Предметна область**

Предметна область: Навчальний відділ

Об’єкти:  Групи, Студенти

Примітка: Існує множина навчальних груп. Кожна група включає в себе множину студентів.

**Завдання**

Реалізувати піраміду Фібоначчі. У вузлах буде зберігатись інформація про групи.

**Теорія**

Піраміда Фібоначчі - невпорядкований набір дерев з коренем, кожне з

яких є незростаючою пірамідою. Кожен вузол [x] містить вказівник на батька p[x] і вказівник на одного з синів child[x]. Дочірні вузли вершини об’єднані в двозв’язний циклічний список (child list). (Операції видалення елемента та об’єднання двох таких списків займають константний час.) Кожен дочірній вузол y має вказівники на лівого та правого своїх братів: left[x], right[x]. Порядок братських вузлів довільний. Кожен вузол містить поле кількості синів degree[x] та логічне поле mark[x] – ознаку наявності втрат дочірніх вузлів з моменту, коли x сам став дочірнім вузлом. Значення mark[x] для новостворених вузлів FALSE; наявна мітка знімається, коли вузол стає дочірнім. Всі корені дерев також зв’язані в двозв’язний циклічний список (root list). Звернення до піраміди H іде через корінь дерева з мінімальним ключем min[H] (мінімальний вузол). Кількість вузлів піраміди зберігається в n[H].

**Алгоритм**

* **MAKE\_FIB\_HEAP** - Повертає нову порожню піраміду Фібоначчі H з n[H]=0 та min[H]=NIL.
* **FIB\_HEAP\_INSERT –** Вставляє новий вузол. Процедура не намагається об’єднати дерева в піраміді. Послідовне виконання FIB\_HEAP\_INSERT k разів призведе до додавання до списку коренів k дерев з одного вузла.
* **FIB\_HEAP\_UNION -** Списки коренів пірамід Н1 та Н2 просто об’єднуються і знаходяться новий мінімальний вузол.
* **FIB\_HEAP\_EXTRACT\_MIN -** Спочатку всі дочірні вузли мінімального вузла переміщуються в список коренів піраміди, який потім ущільняється процедурою CONSOLIDATE, щоб не було коренів однакової степені. Ущільнення полягає у повторному виконанні наступних кроків, поки всі корені в списку не матимуть різні значення поля degree.
* **FIB\_DECREASE\_KEY -** Операція порушує властивість того, що піраміда Фібоначчі складається з невпорядкованих біноміальних дерев (втім, вони близькі до них). Якщо властивість піраміди в дереві порушена, відбувається операція вирізання і, можливо, каскадного вирізання.
* **FIB\_HEAP\_DELETE** - робить х мінімальним значенням піраміди і вилучає його.

**Складність**

* **MAKE\_FIB\_HEAP –** О(1)
* **FIB\_HEAP\_INSERT –** О(1)
* **FIB\_HEAP\_MINIMUM -** О(1)
* **FIB\_HEAP\_UNION -** О(1)
* **FIB\_HEAP\_EXTRACT\_MIN –** О(log(n))
* **FIB\_DECREASE\_KEY –** О(1)
* **FIB\_HEAP\_DELETE -** О(log(n))

**Мова програмування**

С++

**Модулі програми**

//Class for implementation node of heap

template <typename T>

class Node

{

private:

T value;

Node\* parent;

Node\* left;

Node\* right;

Node\* child;

bool mark{};

int degree{};

public:

//Constructors

Node();

explicit Node(T newValue);

//Destructor

~Node() = default;

//Output in console

void Print(int countTabs, Node<T>\* head) const;

//Function for clearing node

void Clear(Node<T>\* head);

//Insertion node between nodes

void InsertBetween(Node<T>\* \_left, Node<T>\* \_right);

//Deletion node

void ExtractBetween();

//Overloaded operator for output

friend std::ostream& operator<< (std::ostream& out, const Node& node)};

//Class for implementation Fiboncсi Heap

template <typename T>

class FibonacciHeap

{

private:

Node<T>\* min;

int count\_nodes{};

//Some helpful fuctions

public:

//Constructors

FibonacciHeap();

explicit FibonacciHeap(T value);

//Destructor

~FibonacciHeap();

//Output in console

void Print() const;

//Union 2 heaps

void UnionHeaps(FibonacciHeap<T>\* first, FibonacciHeap<T>\* second);

//Removing the smallest element

void ExtractMin();

//Changing node value

void DecreaseKey(Node<T>\* node, T newValue)

//Deletion current node

void DeleteNode(Node<T>\* node);

//Insertion new element

Node<T>\* Insert(T value) ;

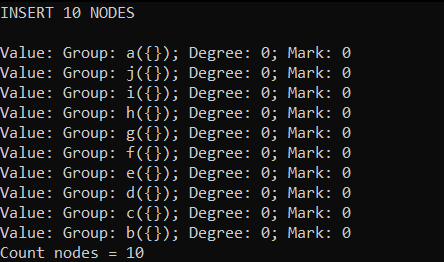
Node<T>\* Insert(Node<T>\* node);

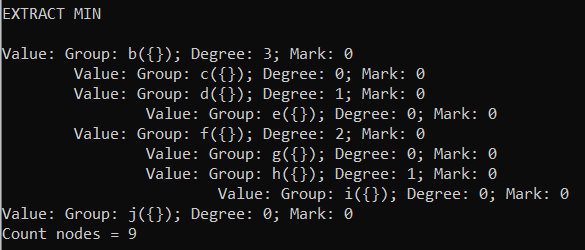
};

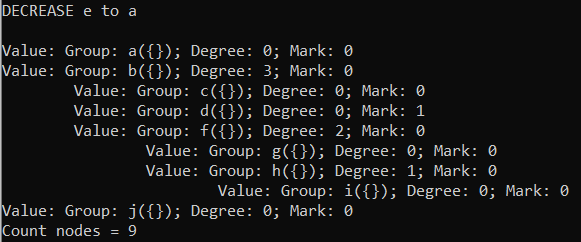
**Інтерфейс користувача**

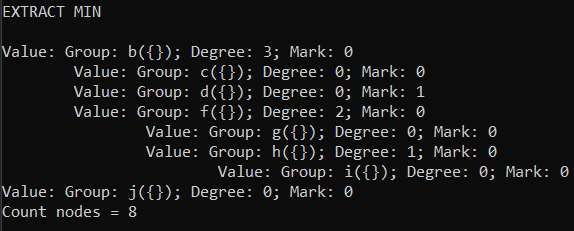
Вхідні дані генеруються програмою і виводяться на консоль. Реалізовані функції вводу з консолі.

**Приклад виводу програми**

****

****

****

****

**Тестовий приклад**

Нехай маємо початкову піраміду

G

F

E

C

B

A

O

N

K

I

H

D

L

M

J

Видалимо найменший елемент

K

O

D

G

F

E

C

B

M

L

N

I

H

Об’єднуємо:

J

K

O

D

G

E

C

B

F

M

L

N

I

H

J

Об’єднуємо:

K

O

D

G

E

B

C

F

M

L

N

H

I

J

E

K

O

D

B

H

G

J

N

C

F

M

L

I

Кінцева піраміда

D

K

B

E

L

M

H

G

J

N

C

F

O

I

**Висновок:**

Отже, з теоретичної точки зору купи Фібоначчі ефективні у використанні, коли кількість операцій ExtractMin і Delete мала порівняно з кількістю інших операцій. Операції, в яких не треба видалення займають константний час виконання.

**Література**

* [**https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8%D0%B5%D0%B2%D0%B0\_%D0%BA%D1%83%D1%87%D0%B0**](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8%D0%B5%D0%B2%D0%B0_%D0%BA%D1%83%D1%87%D0%B0)
* [**https://www.programiz.com/dsa/fibonacci-heap**](https://www.programiz.com/dsa/fibonacci-heap)