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-28
Гуща Дмитро Сергійович

2021

Предметна область

Варіант 4
Предметна область: Учбовий відділ
Об’єкти: Групи, Студенти
Примітка: Маємо множину учбових груп. Кожна група містить в собі множину студентів

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[y], right[y]. 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_MINIMUM** - Процедура повертає вказівник на min[H].
- **FIB_HEAP_UNION -** Списки коренів пірамід H1 та H2 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** - цілком аналогічна BINOMIAL_HEAP_DELETE: робить x мінімальним значенням піраміди і вилучає його.

Складність

- **MAKE_FIB_HEAP** – O(1)
- **FIB_HEAP_INSERT** – O(1)
- **FIB_HEAP_MINIMUM** - O(1)
- **FIB_HEAP_UNION** - O(1)
- **FIB_HEAP_EXTRACT_MIN** – O(log(n))
- **FIB_DECREASE_KEY** – O(1)
- **FIB_HEAP_DELETE** - O(log(n))

Мова програмування

C++

Модулі програми

student.h

```
class Student{}; //Класс опису студента
std::string getName(); // метод повертає ім’я студента
void getStudent();//метод виводить ID та ім'я студента в консоль
void setName(std::string name); //метод змінює ім'я студента
```

group.h

```
class Group {};
Group() : title("NULL"); //конструктор пустої групи
Group(std::string title); //конструктор з початковою назвою групи
Group(std::string title, Student* first_student); //конструктор з початковою назвою
групи та першим студентом
std::string getGroupTitle(); //модуль повертає назву групи
std::vector<Student*> getGroupStudents(); //модуль повертає множину студентів
void setGroupTitle(std::string title); //модуль змінює назву групи
void setGroupStudents(std::vector<Student*> students); //модуль змінює множину
студентів
void addStudent(Student* student); //додати нового студента
void printStudents(); //вивід у консоль усіх студентів групи
```

FibonacciHeap.h

```
class Node; //класс для опису вузла піраміди фібоначі
void print(int countTabs, Node<T>* head) const; //функція для виводу вузла в
консоль
void clear(Node<T>* head); //функція очистки вузла
void insertBetween(Node<T>* _left, Node<T>* _right); //вставка вузла між
іншими двома вузлами
void extractBetween(); //витягення вузла між іншими двома вузлами

class FibonacciHeap; //клас для опису піраміди Фібоначчі
void cut(Node<T>* son, Node<T>* parent); //робить розріз піраміди між двума
вузлами
void cascadingCut(Node<T>* node); //робить розріз піраміди
void consolidate(); //Функція для відновлення властивостей піраміди фібоначчі
після різних операцій
void link(Node<T>* lower, Node<T>* higher); //відновлення властивості
неспадання вузлів
void print() const; //функція виводить піраміду на консоль
void unionHeaps(FibonacciHeap<T>* first, FibonacciHeap<T>* second);
//з'єднання піраміди Фібоначчі
```

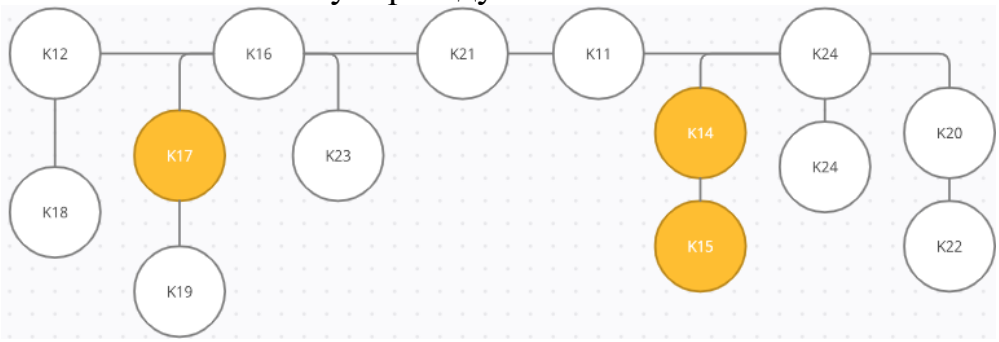
```
void extractMin(); //виконує операцію витягнення найменшого вузла
void decreaseKey(Node<T>* node, T newValue); //виконує операцію збільшення
вузла
Node<T>* insert(T value); //виконує операцію вставки вузла у піраміду
Node<T>* insert(Node<T>* node); //виконує операцію вставки вузла у піраміду
```

Інтерфейс користувача

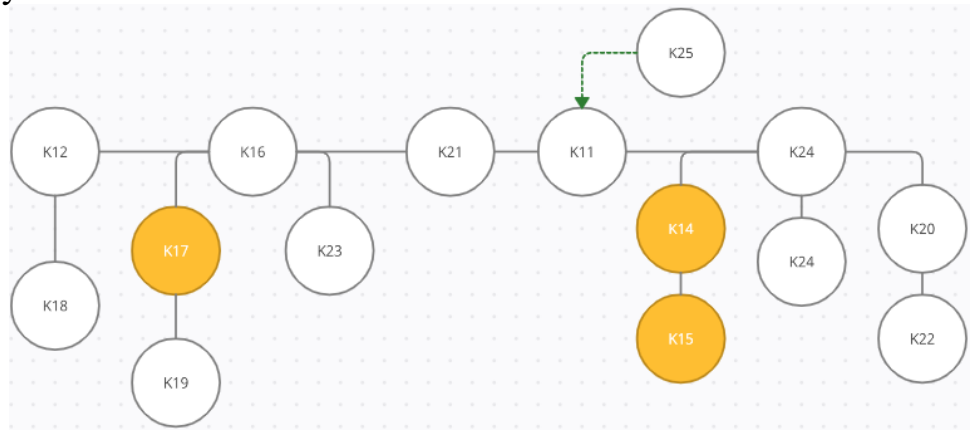
Вхідні дані вводяться і виводяться у консоль.

Тестовий приклад

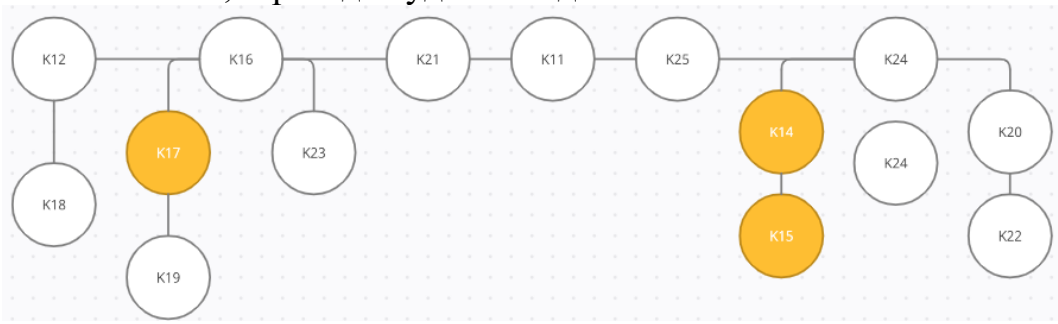
Нехай маємо початкову піраміду



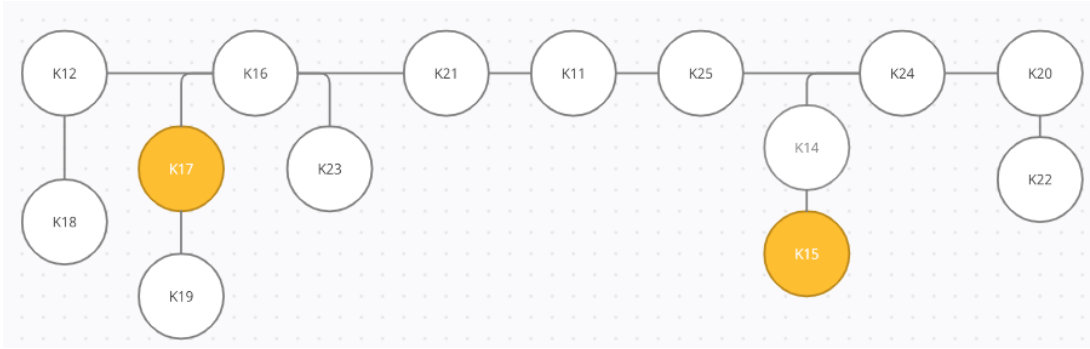
Де сама піраміда це є база даних груп. Спочатку виконаємо вставку нового вузла



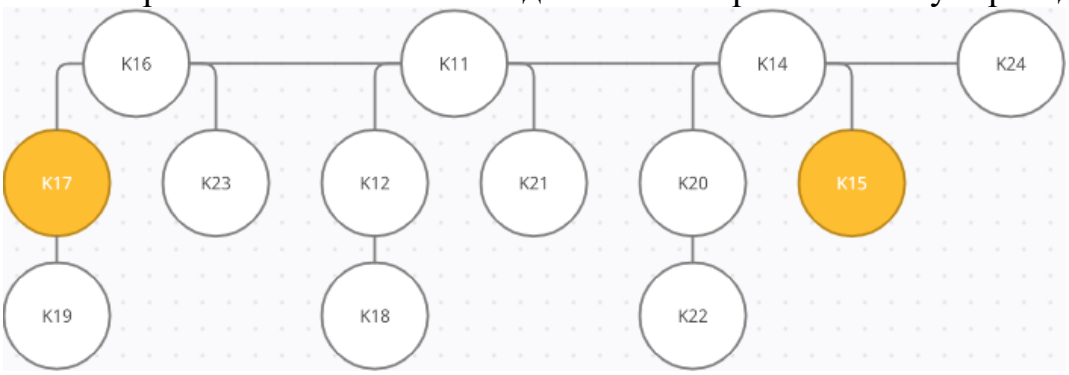
Після вставки, піраміда буде виглядати таким чином



Далі видалимо найменший вузол



Після порівняння степенів і об’єднання ми отримаємо таку піраміду



Висновок:

Отже, з теоретичної точки зору купи Фібоначчі особливо варто використовувати, коли кількість Extract-Min і Delete операцій 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://www.programiz.com/dsa/fibonacci-heap**