**Київський національний університет імені Тараса Шевченка Факультет комп’ютерних наук та кібернетики**

**Кафедра інтелектуальних програмних систем**

**Алгоритми та складність**

**Завдання № 8**

**Варіант № 3**

**Звіт**

**Виконав:**

студент групи К-29

Грищенко Юрій Анатолійович

**Київ-2020**

**Завдання**

Реалізація піраміди Фібоначі.

**Предметна область**: відділ кадрів (варіант 3). Об’єкти: відділення фірми, працівники. Маємо множину відділень, у кожному відділенні зберігається множина працівників.

**Теорія**

Припустимо, що нам необхідно зберігати інформацію про множину працівників, впорядкованих за певним критерієм (наприклад, у алфавітному порядку). Нехай нам треба швидко вставляти нові елементи, шукати мінімальний елемент та зливати дві множини між собою.

В попередній лабораторній роботі ми поставили перед собою ідентичну задачу, і потім розглядали біноміальні піраміди. Зараз же скористаємося **пірамідами Фібоначі** і порівняємо їх з біноміальними пірамідами.

Декілька слів про піраміди Фібоначі:

* Слабша структура, ніж у біноміальної піраміди.
* Амортизований час операцій, що не використовують видалення, дорівнює O(1).
* Можуть виявитися корисними в алгоритмах, де частка операцій EXTRACT\_MIN та DELETE відносно мала (ряд алгоритмів на графах: наприклад, пошук мінімального кістякового дерева, найкоротший шлях з однієї вершини, алгоритми, що використовують DECREASE\_KEY для кожного ребра в майже повних графах).
* Складність реалізації, порівняно з бінарними (та k-арними) пірамідами, вища.
* Значення констант у формулах часу виконаннятакож високі.

**Структура даних**

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.

Для піраміди Фібоначі H позначимо

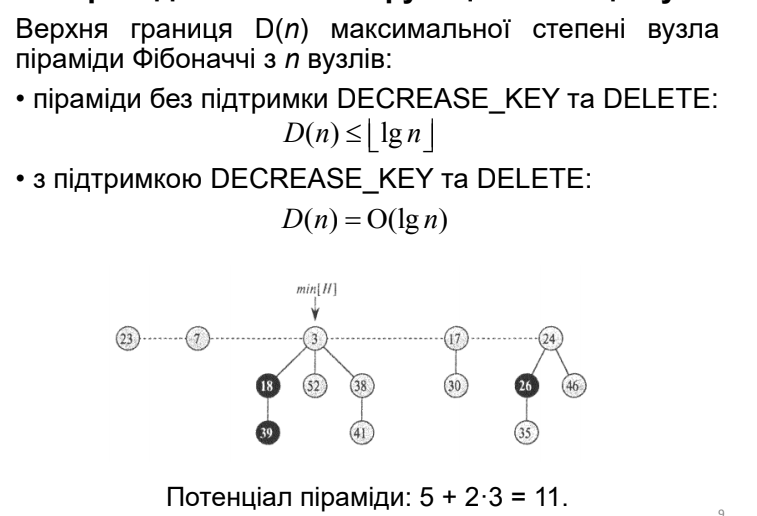
**t(H)** – кількість дерев у списку коренів H,

**m(H)** – кількість вузлів з мітками в піраміді H.

Тоді потенціал піраміди визначається як Ф(H) = t(H) + 2m(H).

Потенціал множини пірамід – сума потенціалів пірамід-складників.

Вважаємо, що одиниці потенціалу достатньо для покриття вартості довільної операції часу О(1). На початку роботи піраміда порожня, тобто початковий потенціал дорівнює 0. Тоді в подальшому потенціал завжди буде залишатися невід’ємним. Верхня границя загальної амортизованої вартості є верхньою границею загальної фактичної вартості послідовності операцій.



Якщо підтримуються лише операції MAKE\_HEAP, INSERT, MINIMUM, EXTRACT\_MIN та UNION, кожна піраміда Фібоначі буде набором невпорядкованих біноміальних дерев (unordered binomial tree):

невпорядковане біноміальне дерево U0 складається з єдиного вузла;

невпорядковане біноміальне дерево Uk складається з двох невпорядкованих біноміальних дерев Uk-1, причому корінь одного з них є довільним дочірнім вузлом іншого.

**Особливість пірамід Фібоначі:** об’єднання дерев у піраміді буде відбуватися лише під час операції EXTRACT\_MIN.

**Властивості невпорядкованих біноміальних дерев:**

Невпорядковане біноміальне дерево Uk

1. має 2k вузлів;

2. має висоту k;

3. має Сki вузлів на глибині i = 0,1...k;

4\* має корінь степеня k, а степінь інших вузлів буде менша; при цьому синами кореня є корені піддерев U0, U1, ..., Uk-1 в деякому порядку.

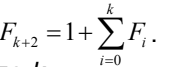
• Для піраміди Фібоначі з n вузлами максимальна степінь вузла

****

**Оцінка максимальної степені вузла:**

Для кожного вузла х визначимо size(x) – кількість вузлів у піддереві з коренем х, включно з ним.

**Лема 1.** Нехай х – довільний вузол піраміди Фібоначі та у1,...,уk – дочірні вузли х в порядку їх зв’язування з х від ранніх до пізніх. Тоді degree[y1] ≥ 0 та degree[yi] ≥ i – 2 при і=2,3,...,k.



**Лема 2.** Для всіх цілих k ≥ 0:

**Лема 3.** Нехай х – довільний вузол піраміди Фібоначі, а k = degree[x] – його степінь.

Тоді size(x) ≥ Fk+2 ≥ φk, де

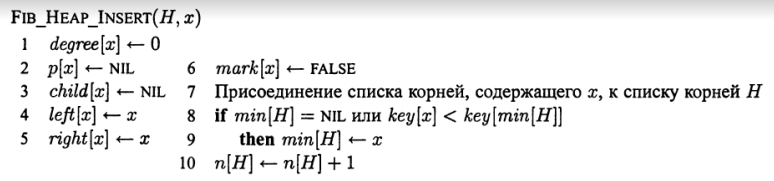
**Наслідок.** Максимальна степінь D(n) довільного вузла в піраміді Фібоначі з n вузлами дорівнює O(lg n).

**Алгоритми**

**Створення піраміди Фібоначі:**

Повертає нову порожню піраміду Фібоначі H з n[H]=0 та min[H]=NIL. Час О(1).

**Вставка вузла:**



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. Додавання вузла відбувається за **постійний час О(1).**

**Пошук мінімального вузла:**

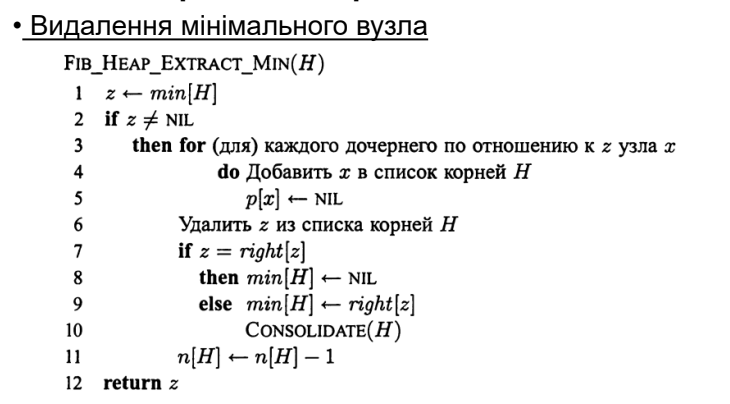
На нього вказує min[H], тому пошук відбувається за час О(1).

**Об’єднання двох пірамід:**

Списки коренів пірамід Н1 та Н2 просто об’єднуються і шукається новий мінімальний вузол. Фактичний час виконання О(1)

**Видалення мінімального вузла:**

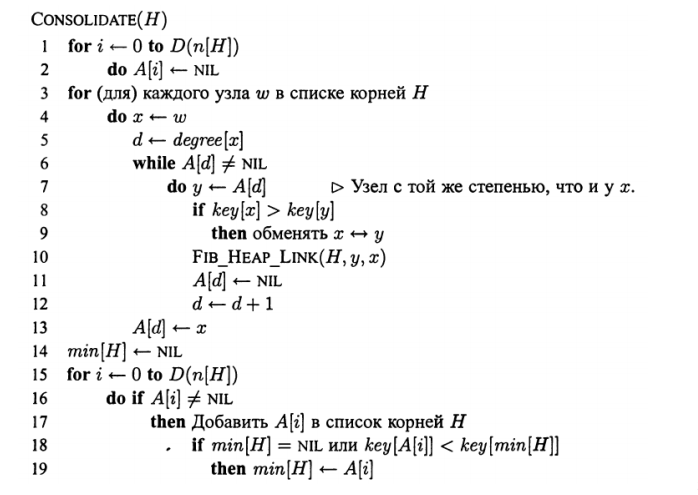
Спочатку всі дочірні вузли мінімального вузла переміщуються в список коренів піраміди, який потім ущільнюється процедурою CONSOLIDATE, щоб не було коренів однакової степені.

****

Ущільнення полягає у повторному виконанні наступних кроків, поки всі корені в списку не матимуть різні значення поля degree:

1. Знайти два корені x та y однакової степені, причому key[x] ≤ key[y].
2. Прив’язати (link) y до x: видалити y зі списку коренів та зробити сином x. Поле degree[x] при цьому збільшується, а мітка mark[y], якщо вона була, знімається.

Процедура CONSOLIDATE використовує допоміжний масив A[0..D(n(H))]. Якщо A[i]=y, то y в даний момент є коренем степені degree[y]=i.

****

По завершенні циклу for в списку коренів залишиться не більше одного кореня кожної степені, і елементи масиву А вказуватимуть на ці корені.

Якщо перед викликом FIB\_HEAP\_EXTRACT\_MIN всі дерева піраміди були невпорядкованими біноміальними деревами, то після виконання процедури вони такими і залишаться. Шляхи отримання нових дерев:

* дочірні дерева вузла, що видаляється (вони за умовою є невпорядкованими біноміальними деревами);
* дерево, отримане об’єднанням двох дерев однакової степені (якщо вони мають структуру Uk, то їх об’єднання процедурою FIB\_HEAP\_LINK дає дерево типу Uk+1).

Потенціал до вилучення мінімального вузла t(H)+2m(H), а після – не перевищує (D(n)+1)+2m(H), бо залишається не більше (D(n)+1) коренів і нових міток не утворюється.

Верхня оцінка амортизованої вартості:

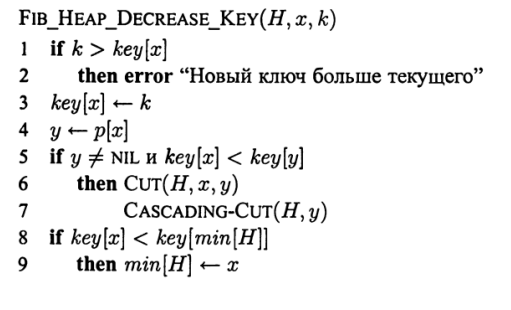
О(t(H)+D(n)) + ((D(n)+1)+2m(H)) – (t(H)+2m(H)) =

= О(D(n)) + O(t(H)) – t(H) = О(D(n)).

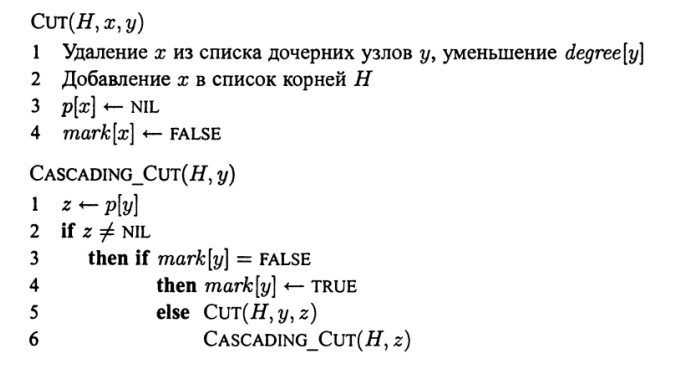
Остання рівність справедлива, оскільки можна масштабувати одиниці потенціалу так, щоб знехтувати константою, прихованою в O(t(H)).

**Раніше ми показали, що D(n) = O(lg n), тому амортизована вартість процедури складе O(lg n).**

**Зменшення ключа:**

Операція порушує властивість того, що піраміда Фібоначі складається з невпорядкованих біноміальних дерев (втім, вони близькі до них).

Якщо властивість піраміди в дереві порушена, відбувається операція вирізання і, можливо, каскадного вирізання.



Вирізання означає, що утворюється нове дерево з коренем х.

Каскадне вирізання робить вершину новим коренем, якщо вона була помічена, і рекурсивно піднімається вище, або помічає її і зупиняється. Процес також зупиняється при досягненні кореня.

Тобто при каскадному видаленні вершина вирізатиметься, якщо вона перед цим втратила другого сина.

Таким чином, поле мітки mark[x] буде змінюватися в таких випадках:

1. х став коренем (скидання мітки);

2. х прив’язали до іншого вузла (скидання мітки);

3. вирізали сина х (мічення).

В кінці процедури FIB\_HEAP\_DECREASE\_KEY може відбутися оновлення мінімального вузла піраміди. Єдиний інший кандидат – перший вирізаний вузол.

Кожен рекурсивний виклик CASCADING\_CUT, крім останнього, здійснює вирізання та скидає мітку. Після цього отримуємо (t(H)+c) дерев (початкові t(H), (с–1) вирізане та одне з коренем х) та не більше (m(H)–c+2) помічених вузла (каскадом зняли мітку в (с–1) вузла та один можливо помітили в кінці).

Зміна потенціалу складе

((t(H)+c)+2(m(H)–c+2)) – (t(H)+2m(H)) = 4–c.

Амортизована вартість не перевищить О(с) + 4 – с = **О(1).**

Коли мічений вузол видаляється в процесі каскадного вирізання, скидання мітки призводить до зменшення потенціалу на 2: одна одиниця оплачує вирізання і скидання мітки, друга компенсує збільшення потенціалу через появу нового кореня. Тому функція потенціалу містить як член подвоєну кількість помічених вузлів.

**Основні модулі програми**

**Мова програмування: C++11.**

У своїй програмі я розбив модулі на дві категорії: ті, що стосуються саме предметної області (HumanResources.\*) і ті, що стосуються біноміальної піраміди (FibonacciHeap.hpp). main.cpp містить код для інтерактивного режиму.

FibonacciHeap<T> реалізовано як template-клас, тобто він може зберігати об’єкти будь-якого класу. Єдина умова: для класу Т мають бути реалізовані оператори порівняння (>. ≥, <. ≤), за допомогою яких будуть сортуватися елементи, і оператор <<, за допомогою якого елементи дерева виводяться на екран.

template<typename T>

FibonacciHeap<T>::~FibonacciHeap()

**- створення пустої піраміди.**

template<typename T>

FibonacciHeap<T>::~FibonacciHeap()

**- деструктор**, звільняє використану пам’ять

template<typename T>

T FibonacciHeap<T>::peek() const

- **пошук найменшого елемента**

template<typename T>

void FibonacciHeap<T>::merge(FibonacciHeap<T>& heap)

- **злиття двох куп**, heap стає порожнім і всі його елементи переносяться в this

template<typename T>

void FibonacciHeap<T>::insert(const T& element)

- **вставка новго ключа**

template<typename T>

T FibonacciHeap<T>::pop()

- **вилучення найменшого елемента**

template<typename T>

void FibonacciHeap<T>::printStructure(std::ostream& os) const

**- виводить на екран** елементи у струкурованому вигляді

template<typename T>

bool FibonacciHeap<T>::contains(const T& key) const

- **пошук елемента у купі** (для біноміальних куп не є ефективним, виконується за O(n))

template<typename T>

void FibonacciHeap<T>::decreaseKey(const T& key, const T& newKey)

- **зменшення значення ключа,** якщо не відомий вузол (node). Спочатку виконує пошук елемента, потім викликає попередню функцію (час О(n))

template<typename T>

void FibonacciHeap<T>::deleteKey(const T& key)

- **видалення ключа з купи**

**Інтерфейс користувача, тестові приклади**

У програмі є інтерактивний режим, який дозволяє користувачеві створити купу Фібоначі, що збергіає імена працівників, і виконувати всі основні операції над нею (окрім merge)

Available commands: insert, peek, pop, search, decrease, delete, print, clear, help, exit

> **insert**

Enter employee name: **Yurii**

Inserted Yurii.

> **insert**

Enter employee name: **Alice**

Inserted Alice.

> **insert**

Enter employee name: **Bob**

Inserted Bob.

> **peek**

First employee is Alice

> **print**

{ Name: Alice }

{ Name: Yurii }

{ Name: Bob }

> **insert**

Enter employee name: **5**

Inserted 5.

> **insert**

Enter employee name: **3**

Inserted 3.

> **insert**

Enter employee name: **4**

Inserted 4.

> **insert**

Enter employee name: **1**

Inserted 1.

> **insert**

Enter employee name: **2**

Inserted 2.

> **peek**

First employee is **1**

> **print**

{ Name: 1 }

{ Name: 3 }

{ Name: 5 }

{ Name: Alice }

{ Name: Yurii }

{ Name: Bob }

{ Name: 4 }

{ Name: 2 }

> **pop**

Popped 1

> **print**

{ Name: 2 }

{ Name: 3 }

{ Name: 5 }

{ Name: Alice }

{ Name: Yurii }

{ Name: 4 }

{ Name: Bob }

> **delete**

Enter employee name: **Yurii**

Deleted key

> **print**

{ Name: 2 }

{ Name: 3 }

{ Name: 5 }

{ Name: Alice }(\*)

{ Name: 4 }

{ Name: Bob }

> **decrease**

Enter employee name: **5**

Enter new name (lower alphabetically): 0

Decreased key

> **print**

{ Name: 0 }

{ Name: 2 }

{ Name: 3 }

{ Name: Alice }(\*)

{ Name: 4 }

{ Name: Bob }

> **peek**

First employee is 0

> **pop**

Popped 0

> **print**

{ Name: 2 }

{ Name: 3 }

{ Name: Alice }(\*)

{ Name: 4 }

{ Name: Bob }

> **exit**

**Висновки**

Ми дослідили стрктуру і властивості пірамід Фібоначі, проаналізували амортизовану вартість операцій, і побачили, що операції, які не пов`язані з видаленням елементів, мають амортизовану вартість O(1) (інші ж мають O(log n)).

Це означає, що ця структура даних є дуже корисною для певних алгоритмів, де рідко виконуються операції pop чи deleteKey, наприклад, алгоритми на графах. У цих випадках піраміди Фібоначи дають значну перевагу в швидкодії у порівнянні з біноміальними пірамідами.

**Список використаних джерел**

Лекція 6 з курсу “Алгоритми та складність” Шкільняк О.С. 2019-2020