**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра інтелектуальних програмних систем**

**Алгоритми та складність**

**Завдання № 7**

**Варіант № 3**

**Звіт**

**Виконав:**

студент групи К-29

Грищенко Юрій Анатолійович

**Київ-2020**

**Завдання**

Реалізація біноміальної піраміди

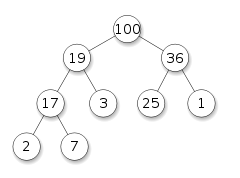
**Предметна область**: відділ кадрів (варіант 3). Об’єкти: відділення фірми, працівники. Маємо множину відділень, у кожному відділенні зберігається множина працівників.

**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.

Взагалі, абстрактний клас множин, який використовується для таких задач, називається **чергою з пріорітетом**[1]. Існують різні структури даних для черг з пріорітетами, але їх часто реалізують за допомогою **пірамід** **або куп** **(heap).**

Загалом, **неспадна піраміда** — це спеціалізована деревовидна структура даних, в якій існують певні властивості впорядкованості: якщо *В* — вузол нащадок *A* — тоді ключ(*A*) ≤ ключ(*B*).[2]

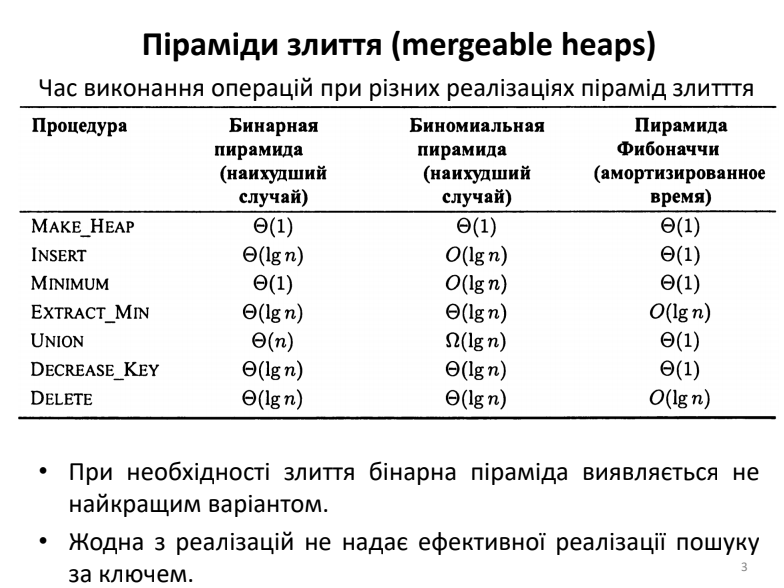


(приклад спадної піраміди)

Зазвичай вони підтримують такі функції:

* **MAKE\_HEAP():** створення нової порожньої піраміди.
* **INSERT(H,x):** вставка готового вузла x в піраміду H.
* **MINIMUM(H)**: повертає вказівник на вузол піраміди H з найменшим ключем.
* **EXTRACT\_MIN(H)**: видаляє вузол піраміди H з найменшим ключем і повертає вказівник на нього.
* **UNION(H1 ,H2)**: повертає нову піраміду – результат злиття H1, H2 (вони не зберігаються).
* **DECREASE\_KEY(H,x,k):** присвоєння вузлу x піраміди H значення ключа k, що є меншим за поточне.
* **DELETE(H,x):** видалення вузла x з піраміди H.

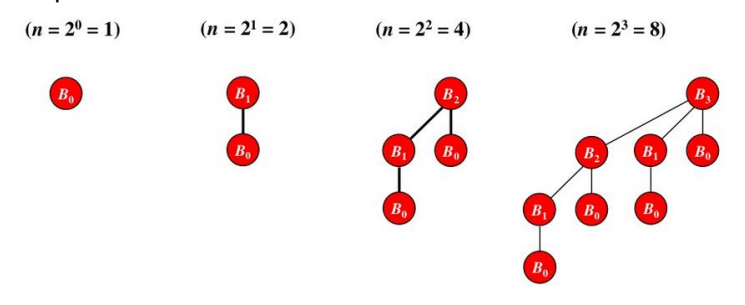
Проте звичайні піраміди не завжди ефективно виконують ці операції. Існують більш оптимальні структури даних, також основані на пірамідах:



В цій роботі розглянемо **біноміальні піраміди**. Вони складаються з **біноміальних дерев.**

**Біноміальні дерева (binomial trees):**

* Впорядковане дерево, що визначається рекурсивно.
* Біноміальне дерево B0 порядку 0 складається з єдиної вершини.
* Біноміальне дерево Bk порядку k складається з двох зв’язаних біноміальних дерев Bk–1: корінь одного є крайнім лівим сином іншого.



Біноміальне дерево Bk:

1. має 2k вузлів;
2. має висоту k;
3. має рівно вузлів на глибині i = 0,1...k;
4. має корінь степені k, а степінь синів менша степеня кореня; при цьому якщо синів пронумерувати зліва направо числами (k–1), (k–2), ..., 0, то i-й син є коренем біноміального дерева Bi

(доведення наведені в [3])

**Наслідок**. Максимальна степінь вузла біноміального дерева з n вершинами складає lg n.

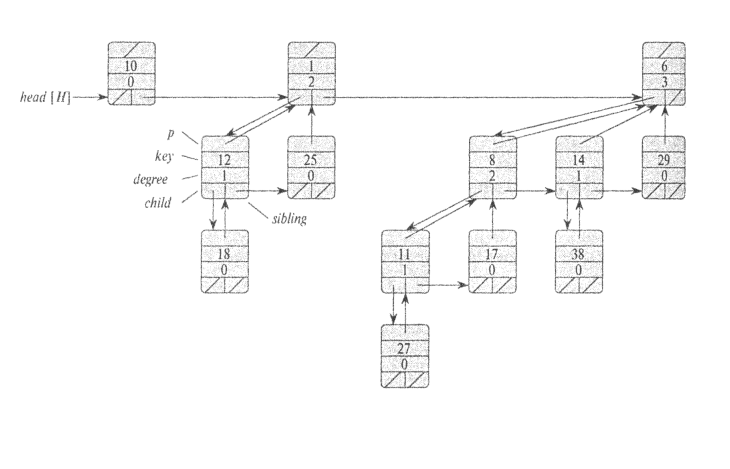
**Біноміальна піраміда (біноміальна купа)** Н – множина біноміальних дерев, що задовольняють властивостям біноміальних пірамід:

1. кожне біноміальне дерево в Н є неспадаючою пірамідою (мінімальний елемент на вершині);
2. для довільного невід’ємного k в Н існує не більше одного біноміального дерева відповідного порядку.

Біноміальна піраміда з n вузлів складається не більше ніж з (lg n + 1) біноміальних дерев.

**Представлення біноміальних пірамід:**

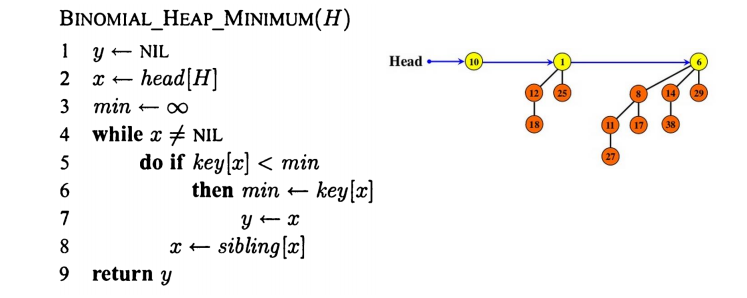
* Кожне біноміальне дерево зберігається у представленні з лівим дочірнім та правим сестринським вузлами.
* Ключ key[x], вказівник на батька p[x], вказівник на найлівішого сина child[x], вказівник на правого брата sibling[x], кількість дочірніх вузлів degree[x].
* Біноміальна піраміда представлена списком коренів її біноміальних дерев впорядкованим за зростанням степенів дерев.
* Вказівник на перший корінь біноміальної піраміди H: head[H].
* Якщо x – корінь, то sibling[x] вказує на наступний корінь у списку.



**Алгоритми**

**Створення порожньої біноміальної піраміди.** Виставляємо head[H] = nil. Час роботи Θ(1).

**Пошук мінімального ключа** (вважаємо, що відсутні ключі зі значенням):

 Час роботи O(lg n), бо перевіряємо не більше (lg n + 1) коренів.

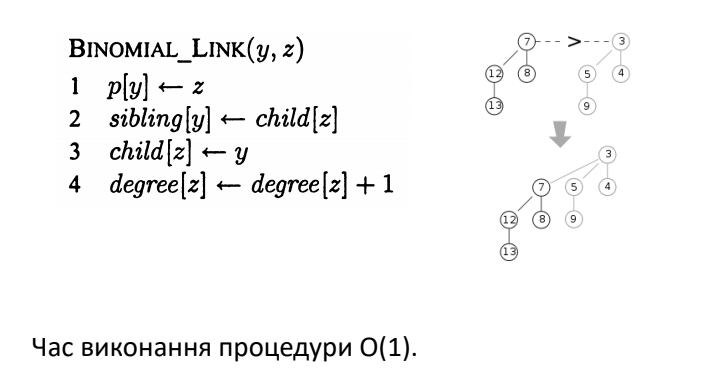
**Злиття двох біноміальних пірамід.**

1. Злити списки коренів H1 та H2 в упорядкований список (BINOMIAL\_HEAP\_MERGE).
2. Відновити властивості біноміальної піраміди H.

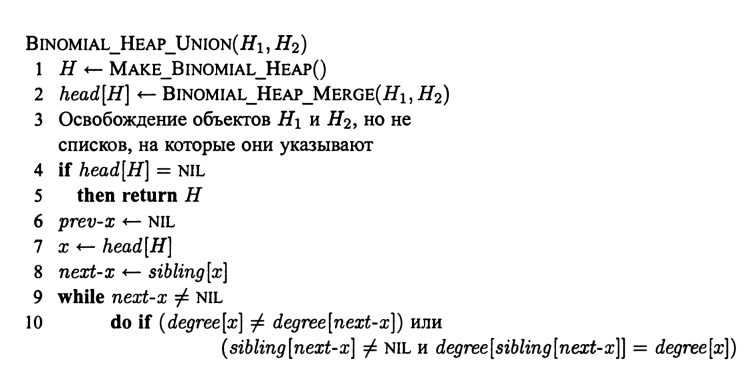
Процедура **BINOMIAL\_HEAP\_MERGE** діє аналогічно етапу злиття в сортуванні злиттям, на кожному кроці переміщаючи в результуючий список дерево меншого порядку. Час її роботи Ο(lg n), де n – сумарна кількість вершин в двох пірамідах.

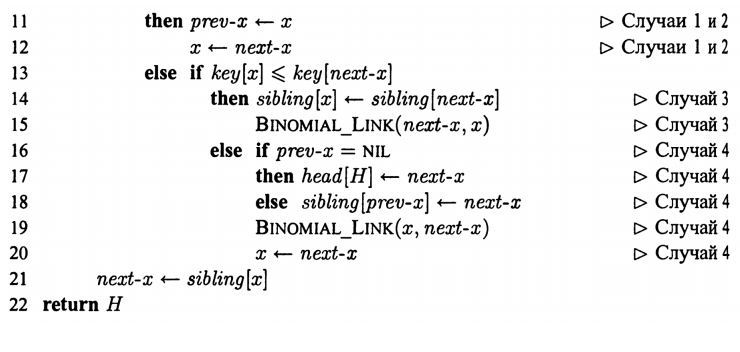
Після злиття списків коренів відомо, що купа H містить не більше двох коренів однакової степені, і вони стоять підряд. Тому будемо зв’язувати корені однієї степені поки всі корені не отримають різні степені.

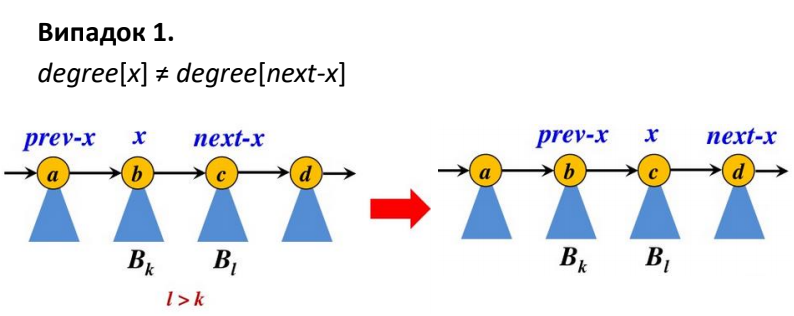
**Допоміжна операція зв’язування** двох біноміальних дерев одного порядку Bk-1 в біноміальне дерево Bk (дерево з коренем y "підчеплюється" до дерева z):

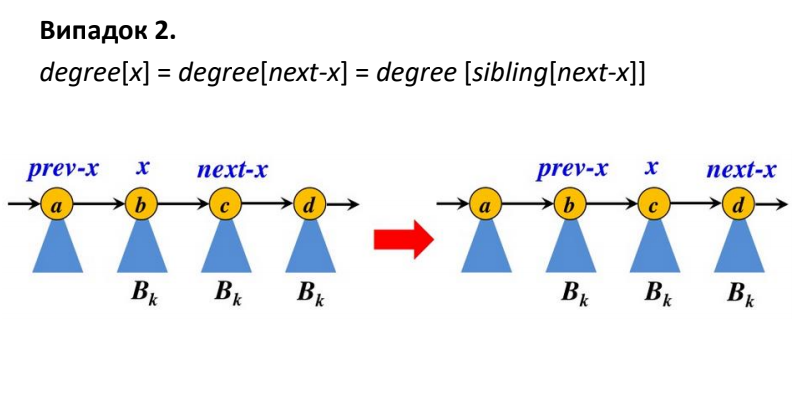
****

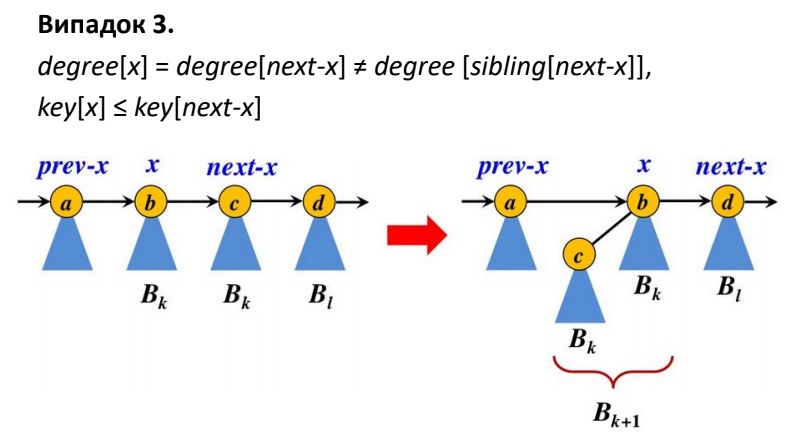
Увесь алгоритм буде виглядати так:

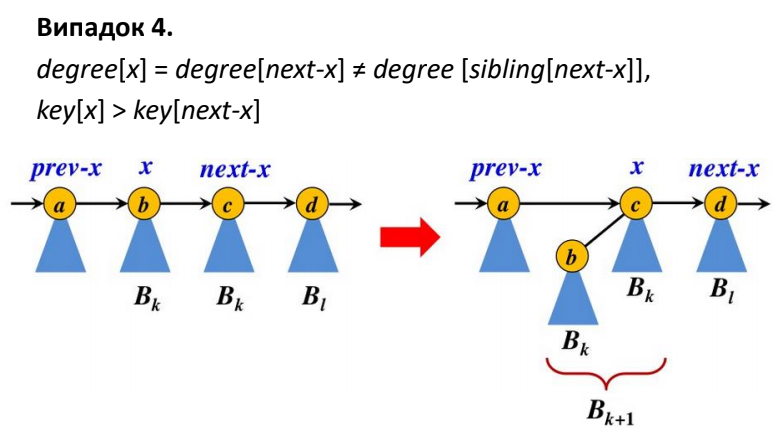
****

 Для відновлення властивостей біноміальної піраміди використаємо три допоміжні вказівники: x – поточний корінь, prev-x – попередник x, next-x – наступник x. Далі рухаємось списком коренів. Можливі ситуації:



****

****



Порахуємо час роботи BINOMIAL\_HEAP\_UNION. Нехай біноміальна купа Н1 містить n1 вузлів, а Н2 – n2 вузлів та n1 + n2 = n. Тоді Н1 має максимум log n1 + 1 корінь, а Н2 – log n2 + 1 корінь, тому в Н буде не більше log n1 + log n2 + 2 ≤ 2 log n + 2 = О(log n) коренів – час роботи BINOMIAL\_HEAP\_MERGE.

Кожна ітерація циклу виконується за константний час. Оскільки ми проходимо весь список, то ітерацій буде не більше log n1 + log n2 + 2.

Отже, загальний **час виконання BINOMIAL\_HEAP\_UNION складе О(log n).**

**Вставка вузла.**

Створюється біноміальна піраміда з одним вузлом (час Ο(1)) та зливається з початковою пірамідою (загальний час Ο(lg n)). **Отже, загальний час О(lg n).**

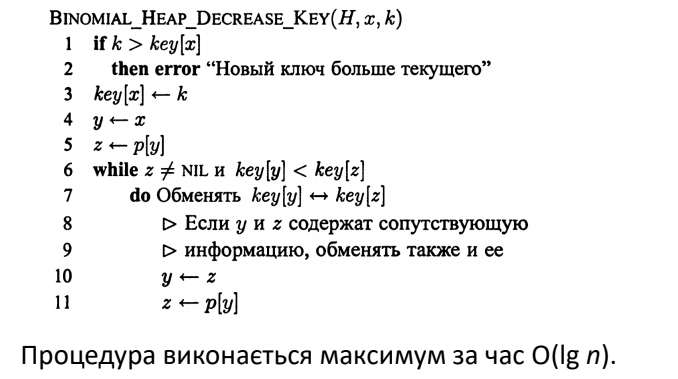
**Вилучення мінімального вузла.**

Видаляємо мінімальний корінь. Утворюємо з його синів нову біноміальну піраміду (перестановкою у зворотному порядку). Це виконується за час O(lg n), оскільки корінь дерева, в якому міститься n елементів, матиме lg n безпосередніх синів.

Зливаємо новоутворену піраміду з вихідною. **Час роботи процедури Ο(lg n).**

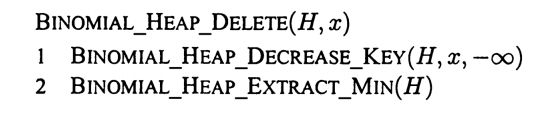
**Зменшення ключа.**

Значення ключа замінюється на менше. Після цього рухаємось у напрямку кореня, обмінюючи значення, якщо порушена умова неспадаючої піраміди.



**Час виконання циклу — О(lg n)**, оскільки дерево, що містить n елементів, матиме степінь lg n.

**Видалення ключа.**

****

Вважаємо, що жоден ключ не може містити ключ –. Ключ у вузлі для видалення робиться –. При цьому розглянута вершина стає одним з коренів, мінімальним, і може бути вилучена за допомогою процедури BINOMIAL\_HEAP\_EXTRACT\_MIN. **Сумарний час виконання складе Ο(lg n).**

**Основні модулі програми**

**Мова програмування: C++14.**

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

BinomialHeap<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>

BinomialHeap<T>::~BinomialHeap()

**- створення пустої піраміди.**

template<typename T>

BinomialHeap<T>::~BinomialHeap()

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

template<typename T>

T BinomialHeap<T>::peek() const

- **пошук найменшого елемента**

template<typename T>

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

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

template<typename T>

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

- **вставка новго ключа**

template<typename T>

T BinomialHeap<T>::pop()

- **вилучення найменшого елемента**

template<typename T>

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

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

template<typename T>

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

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

template<typename T>

void BinomialHeap<T>::decreaseKey(BinomialNode<T>\* node, T\* newKey)

- **зменшення значення ключа,** якщо є відомий вузол (node).

template<typename T>

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

- **видалення ключа з купи**

**Інтерфейс користувача, тестові приклади**

У програмі є інтерактивний режим, який дозволяє користувачеві створити біноміальну купу, 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: **Bob**

Inserted Bob.

> **insert**

Enter employee name: **Alice**

Inserted Alice.

> **print**

{ Name: Alice }

{ Name: Bob }

{ Name: Yurii }

> **insert**

Enter employee name: **David**

Inserted David.

> **insert**

Enter employee name: **Carl**

Inserted Carl.

> **search**

Enter employee name: **Yurii**

Heap contains Yurii? true

> **peek**

First employee is Alice

> **print**

{ Name: Carl }

{ Name: Alice }

{ Name: Bob }

{ Name: Yurii }

{ Name: David }

> **pop**

Popped Alice

> **search**

Enter employee name: **Alice**

Heap contains Alice? false

> **pop**

Popped Bob

> **pop**

Popped Carl

> **pop**

Popped David

> **pop**

Popped Yurii

> **pop**

Heap is empty

> **clear**

Deleting tree...

> **insert**

Enter employee name: **1**

Inserted 1.

> **insert**

Enter employee name: **2**

Inserted 2.

> **insert**

Enter employee name: **3**

Inserted 3.

> **insert**

Enter employee name: **4**

Inserted 4.

> **insert**

Enter employee name: **5**

Inserted 5.

> **insert**

Enter employee name: **6**

Inserted 6.

> **insert**

Enter employee name: **7**

Inserted 7.

> **print**

{ Name: 7 }

{ Name: 5 }

{ Name: 6 }

{ Name: 1 }

{ Name: 3 }

{ Name: 4 }

{ Name: 2 }

> **insert**

Enter employee name: **8**

Inserted 8.

> **print**

{ Name: 1 }

{ Name: 5 }

{ Name: 7 }

{ Name: 8 }

{ Name: 6 }

{ Name: 3 }

{ Name: 4 }

{ Name: 2 }

> **decrease**

Enter employee name: **8**

Enter new name (lower alphabetically): **0**

Decreased key

> **print**

{ Name: 0 }

{ Name: 1 }

{ Name: 5 }

{ Name: 7 }

{ Name: 6 }

{ Name: 3 }

{ Name: 4 }

{ Name: 2 }

> **exit**

**Висновки**

Ми дослідили стрктуру і властивості біноміальних пірамід і дійшли до висновку, що вони є ефективнішими за звичайні піраміди для деяких операцій (особливо злиття).

**Список використаних джерел**

1. https://en.wikipedia.org/wiki/Priority\_queue
2. https://uk.wikipedia.org/wiki/Купа\_(структура\_даних)
3. Лекція 5 з курсу “Алгоритми та складність” Шкільняк О.С. 2019-2020