**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Алгоритми та складність**

**Завдання № 3**

**Звіт**

**Виконав:**

студент групи К-29

Грищенко Юрій Анатолійович

**Київ-2019**

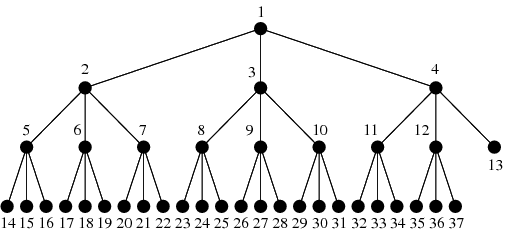
**Умова задачі**

d-арні піраміди схожі на бінарні, лише їх вузли, відмінні від листя, містять не по 2, а по d дочірних елементів. Представте d-арну піраміду у вигляді масиву (якою буде її висота для n елементів?) Розробіть ефективні процедури операцій Extract\_Max, Insert та Increase\_Key, призначених для роботи з d-арною незростаючою пірамідою. Проаналізуйте час роботи цих процедур і виразіть їх в термінах n та d.

**Опис алгоритму:**

Для d-арних пірамід можна використовувати алгоритми та спосіб подання, аналогічні до бінарних.[1]

Приклад представлення тернарної (3-арної) піраміди у вигляді масиву:



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... |

Бачимо, наприклад, що центральним дочірним елементом вузла 2 є 6, для вузла 3 — це 9 і т. д., тобто індекс збільшується в 3 рази.

Висота піраміди складає 1 для 1 елемента, 2 для 2-4 елементів, 3 для 5-13 елементів і т.д — піраміда висотою h може вмістити в собі 1 + 3 + 32 + 33 + … + 3h-1 елементів. Для великих h, якщо останній слой піраміди містить k елементів, то вся піраміда містить k + 1/3k + 1/9k + 1/27k + … = 3/2k, отже останній слой містить приблизно дві третини всіх елементів. Додавання ще одного слоя збільшує кількість елементів у три рази, звідси h ≈ log3n.

У загальному випадку, для d-арної піраміди матимемо h=logdn.

Процедури операцій також стандартні[2]:

//internal function, used for insert();

void increase\_key(int\* heap, int n, size\_t index, int new\_key)

{

heap[index] = new\_key;

while(index != 0)

{

size\_t parent\_index = get\_parent\_index(index, n);

//In a max-heap, the parent must be larger than its children.

if(heap[parent\_index] < heap[index])

swap(heap[parent\_index], heap[index]);

index = parent\_index;

}

}

void insert(int\* heap, size\_t& next\_index, size\_t length, int n, int element)

{

if(next\_index >= length)

throw invalid\_argument("Tried to insert into full array.");

//Insert new element at the bottom layer (so, last element in the array)

//And re-arrange until it becomes a max-heap again.

increase\_key(heap, n, next\_index, element);

next\_index++;

}

int extract\_max(int\* heap, size\_t& next\_index, int n)

{

if(next\_index == 0)

throw invalid\_argument("Tried to extract from empty array.");

//The largest element is at the top

int return\_element = heap[0];

//Replace the top element with the last element

heap[0] = heap[next\_index - 1];

size\_t current\_index = 0;

while(true)

{

//In a max-heap, the parent must be larger than all its children.

size\_t children\_index = get\_children\_index(current\_index, n);

size\_t largest\_index = current\_index;

for(size\_t i = children\_index; i < children\_index + n && i < next\_index; i++)

{

if(heap[i] > heap[largest\_index])

largest\_index = i;

}

if(largest\_index == current\_index)

break;

else

{

swap(heap[largest\_index], heap[current\_index]);

current\_index = largest\_index;

}

}

next\_index--;

return return\_element;

}

Процедури increase\_key та insert виконують по одній операції swap() на кожному рівні піраміди до тих пір, поки вона не відсортується — отже в кращому випадку вони займають O(1) часу, а в гіршому O(h)=O(logdn). Для більших d ефективність цієї функції зростає.

Процедура extract\_max також працює на кожному рівні, але окрім операцій swap() вона виконує порівняння зі всіма «дітьми» вузла, отже в гіршому випадку складність буде O(dh) = O(dlogdn). Для більших d ефективність цієї функції спадає.

**Інтерфейс користувача:**

Користувач вводить числа у консоль, програма їх виводить у відсортованому вигляді.

**int** **main**()

{

cout << "Enter array length:\n";

int length;

cin >> length;

int\* heap\_array = new int[length];

const int n = 3;

size\_t next\_index = 0;

cout << "Enter " << length << " elements.\n";

for (int i = 0; i < length; i++)

{

int new\_element;

cin >> new\_element;

insert(heap\_array, next\_index, length, n, new\_element);

}

cout << "Sorted elements:\n";

while (next\_index > 0)

{

cout << extract\_max(heap\_array, next\_index, n) << endl;

}

return 0;

}

Результат:

Enter array length:

7

Enter 7 elements:

7

4

6

8

9

-1

1

3

Sorted elements:

9

8

6

5

4

3

1

-1

**Список використаних джерел:**

1. https://en.wikipedia.org/wiki/D-ary\_heap

2. https://en.wikipedia.org/wiki/Binary\_heap