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

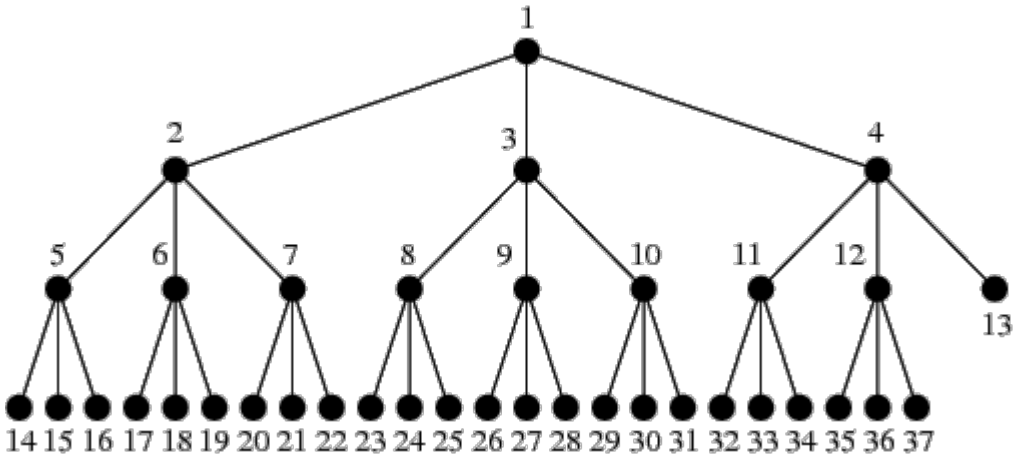
Умова задачі

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 $1 + 3 + 3^2 + 3^3 + \ldots + 3^{h-1}$ елементів. Для великих h, якщо останній слой піраміди містить k елементів, то вся піраміда містить $k + 1/3k + 1/9k + 1/27k + \ldots = 3/2k$, отже останній слой містить приблизно дві третини всіх елементів. Додавання ще одного слоя збільшує кількість елементів у три рази, звідси $h \approx \log_3 n$.

У загальному випадку, для d-арної піраміди матимемо $h=\log_d n$.

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 $O(h)=O(\log_d n)$. 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 $O(dh) = O(d\log_d n)$. Для більших d ефективність цієї функції спадає.

Інтерфейс користувача:

Програма виконується в режимі демонстрації, тобто всі значення та операції задаються в коді, і в консоль виводяться лише результати:

```
int main()
{
	int* heap_array = new int[10];
	const int n = 3;
	size_t next_index = 0;
	insert(heap_array, next_index, 10, n, 9);
	insert(heap_array, next_index, 10, n, 3);
	insert(heap_array, next_index, 10, n, 1);
	insert(heap_array, next_index, 10, n, 8);
	insert(heap_array, next_index, 10, n, -1);
	insert(heap_array, next_index, 10, n, 4);
	insert(heap_array, next_index, 10, n, 5);
	insert(heap_array, next_index, 10, n, 6);
	while(next_index > 0)
	{
		cout << extract_max(heap_array, next_index, n) << endl;
	}
	return 0;
}
```

Результат:

```
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