Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

В рамках лабораторной работы необходимо реализовать бинарную кучу(мин или макс), а так же биноминальную кучу
Для реализованных куч выполнить следующие действия:

- Наполнить кучу N кол-ва элементов (где N = 10 ^ i, i от 3 до 7).
- После заполнения кучи необходимо провести следующие тесты:
- 1000 раз найти минимум/максимум
- 1000 раз удалить минимум/максимум
- 1000 раз добавить новый элемент в кучу
- Для всех операция требуется замерить время на выполнения всей 1000 операций и рассчитать время на одну операцию, а так же запомнить максимальное время которое требуется на выполнение одной операции если язык позволяет его зафиксировать, если не позволяет воспользоваться хитростью и расчитывать усредненное время на каждые 10,25,50,100 операций, и выбирать максимальное из полученных результатов, что бы поймать момент деградации структуры и ее перестройку.
- По полученным в задании 2 данным построить графики времени выполнения операций для усреднения по 1000 операций, и для максимального времени на 1 операцию..
-

Описание метода/модели.

Биноминальная и бинарная куча

Бинарная куча — это полное бинарное дерево, в котором выполняется свойство кучи:

- Min-Heap: Родитель меньше или равен дочерним узлам.
- Max-Heap: Родитель больше или равен дочерним узлам.

Операции и сложность:

- Вставка: Добавление в конец (последний уровень), затем просеивание вверх (O(log N)).
- Удаление минимума (или максимума):
- Удаление корня, замена последним элементом.
- Просеивание вниз (O(log N)).
- Поиск минимума/максимума: O(1).
- Построение кучи: O(N).

Биномиальная куча — это набор биномиальных деревьев, где:

- Каждое дерево упорядочено как куча.

Деревья удовлетворяют биномиальному свойству:

- Дерево степени k содержит 2^k элементов.
- Дочерние деревья образуются объединением двух деревьев одинаковой степени.

Операции и сложность:

- Вставка: Создание одноузлового дерева и слияние (O(log N)).
- Объединение двух куч: O(log N).
- Удаление минимума: Найти минимум и перестроить кучу (O(log N)).
- Поиск минимума: O(log N).

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык C, а для анализа данных Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

Блок кода отвечающий за реализацию куч:

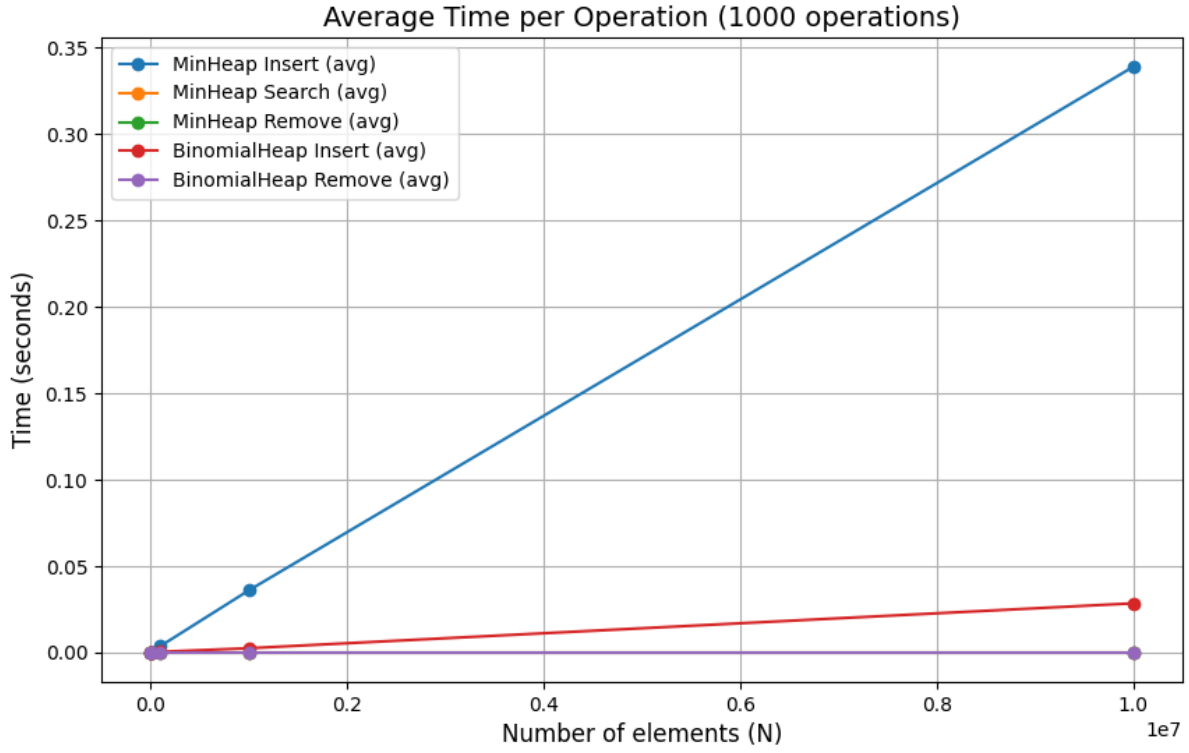
```
typedef struct {
    int *heap;
    int size;
    int capacity;
} MinHeap;
MinHeap* create_minheap(int capacity) {
    MinHeap* heap = (MinHeap*)malloc(sizeof(MinHeap));
    heap->size = 0;
    heap->capacity = capacity;
    heap->heap = (int*)malloc(capacity * sizeof(int));
    return heap;
}
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void heapify(MinHeap *heap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < heap->size && heap->heap[left] < heap->heap[smallest])
        smallest = left;
    if (right < heap->size && heap->heap[right] < heap->heap[smallest])
        smallest = right;
    if (smallest != idx) {
        swap(&heap->heap[idx], &heap->heap[smallest]);
        heapify(heap, smallest);
    }
}
void insert_minheap(MinHeap *heap, int key) {
    if (heap->size == heap->capacity) {
        printf("Overflow: Heap is full!\n");
        return;
    }
    int i = heap->size++;
    heap->heap[i] = key;
    while (i != 0 && heap->heap[(i - 1) / 2] > heap->heap[i]) {
        swap(&heap->heap[i], &heap->heap[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
```

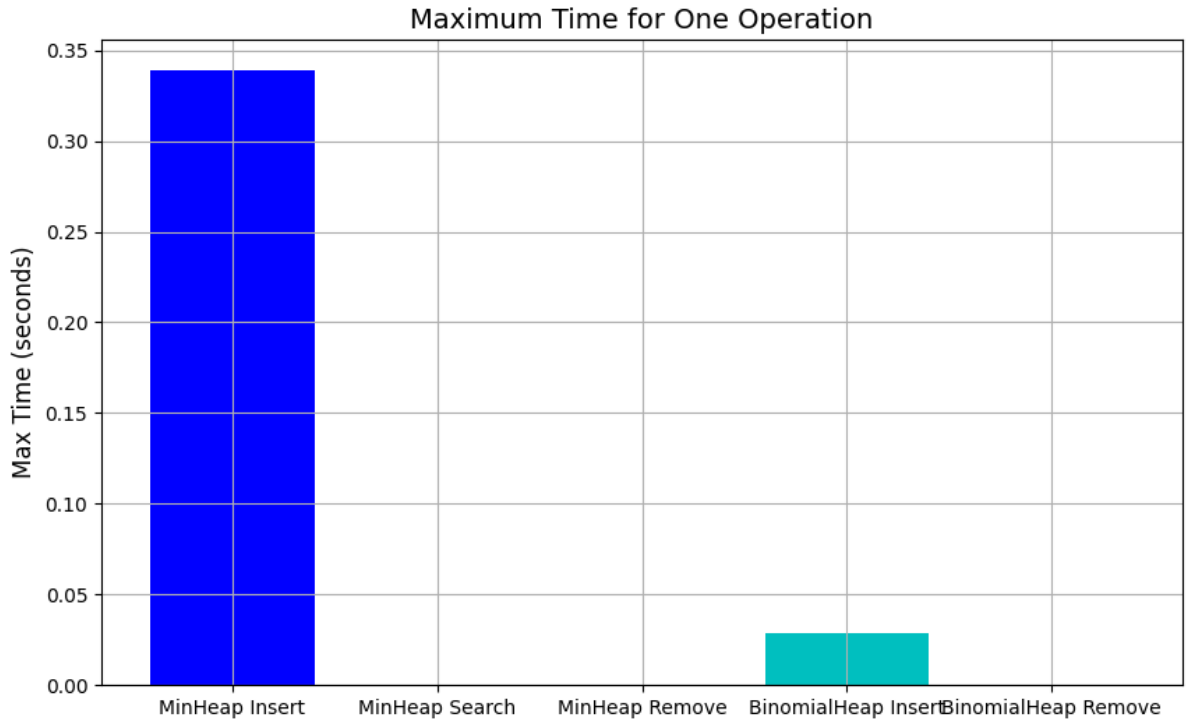
Блоки кода отвечающие за тестирование:

```
    void run_tests() {
        int sizes[] = {1000, 10000, 100000, 1000000, 10000000}; // Для N = 10^i, где i от 3 до 7
        int num_sizes = 5;
        srand(time(NULL));
        for (int i = 0; i < num_sizes; i++) {
            int size = sizes[i];
            printf("Running tests for N=%d\n", size);
            // Генерация случайных данных
            int *data = (int*)malloc(size * sizeof(int));
            for (int j = 0; j < size; j++) {
                data[j] = rand() % 1000000;
            }
            // Тестирование MinHeap
            MinHeap *min_heap = create_minheap(size);
            clock_t start_time = clock();
            for (int j = 0; j < size; j++) {
                insert_minheap(min_heap, data[j]);
            }
            double minheap_insert_time = (double)(clock() - start_time) / CLOCKS_PER_SEC;
            start_time = clock();
            for (int j = 0; j < 1000; j++) {
                get_min(min_heap);
            }
            double minheap_search_time = (double)(clock() - start_time) / CLOCKS_PER_SEC / 1000;
            start_time = clock();
            for (int j = 0; j < 1000; j++) {
                extract_min(min_heap);
            }
            double minheap_remove_time = (double)(clock() - start_time) / CLOCKS_PER_SEC / 1000;
            printf("MinHeap - Insert: %f, Search: %f, Remove: %f\n", minheap_insert_time, minheap_search_time, minheap_remove_time);
            // Тестирование BinomialHeap (аналогично)
            BinomialHeap *binomial_heap = create_binomial_heap();
            start_time = clock();
            for (int j = 0; j < size; j++) {
                insert_binomial_heap(binomial_heap, data[j]);
```

Результаты работы программы выводится в виде графика зависимости и приведены ниже. Так же результаты тестирования в виде строк данных хранятся в data.txt и приведены ниже.

```
  Running                 tests                  for                  N=1000
MinHeap    -    Insert:    0.000045,    Search:    0.000000,    Remove:    0.000000
BinomialHeap       -       Insert:       0.000010,       Remove:       0.000000
Running                  tests                   for                  N=10000
MinHeap    -    Insert:    0.000395,    Search:    0.000000,    Remove:    0.000000
BinomialHeap       -       Insert:       0.000041,       Remove:       0.000000
Running                  tests                   for                  N=100000
MinHeap    -    Insert:    0.003726,    Search:    0.000000,    Remove:    0.000000
BinomialHeap       -       Insert:       0.000449,       Remove:       0.000000
Running                 tests                   for                 N=1000000
MinHeap    -    Insert:    0.035988,    Search:    0.000000,    Remove:    0.000001
BinomialHeap       -       Insert:       0.002546,       Remove:       0.000000
Running                 tests                   for                 N=10000000
MinHeap    -    Insert:    0.339064,    Search:    0.000000,    Remove:    0.000001
BinomialHeap - Insert: 0.028537, Remove: 0.000000
```





Заключение.

Бинарная куча быстрее в поиске минимума (O(1)), но медленнее при объединении (O(N)). Подходит для приоритетных очередей, где объединение редкость.

Биномиальная куча эффективнее для частых слияний (O(log N)) и динамического изменения кучи. Используется в сетевых алгоритмах (Дейкстра, Прим), параллельных системах.

Если важна скорость извлечения минимума — бинарная куча.

Если нужно часто объединять кучи — биномиальная куча предпочтительнее.

Так же в данном случае используется язык C в качестве основного для реализации куч и их тестирования, так как на языке Python перегружается стек и работа программы происходит в разы дольше, чем на C.

Ещё по графикам видно, что скорость работы функций Search и Remove происходят довольно быстро (доли микросекунды), поэтому они выводятся как равные 0, что само собой не так.