AISDI – labolatorium z kopców

Jakub Stępniak
Przemysław Wyziński
Lab 102, grupa 4

Cel ćwiczenia

Celem ćwiczenia była implementacja 3 kopców:
- 2 arnego
- 3 arnego
- 4 arnego

Zostały zaimplementowane zgodnie z interfacem:
- push(wartość) – dodanie wartości do kopca
- pop - wyjęcie wartości ze szczytu kopca
- get_raw_data() - otrzymanie wszystkich wartości, które znajdują się w kopcu
- display() - wyswietlenie kopca

Kopiec jest zaimplementowany jako tablica, którą interpretujemy jako drzewo o własnościach.
- Każdy rodzic ma wartość mniejszą od wszystkich swoich dzieci
- jest lewostronnie pełne, co znaczy, że kolejny poziom drzewa jest warzony dopiero gdy cały poziom zostanie wypełniony od lewej strony.

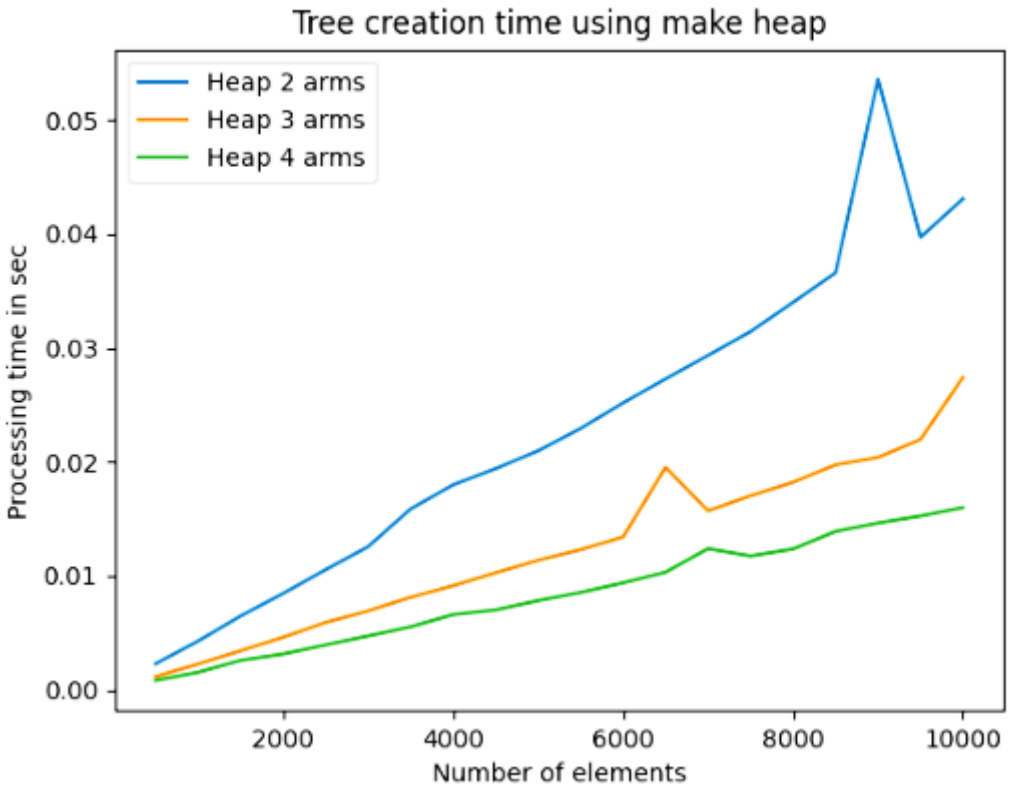
Ilość dzieci wychodzących z rodzica definiujemy jako parametr w konstruktorze. Przewidziana ilość dzieci w ćwiczeniu to 2, 3, 4. W zależności od tego mamy do czynienia z kopcem dwu, trzy, cztero-arnym.

Wyjęcie wartości z kopca następuje przez wyjęcie wartości z tablicy o indeksie 0 (korzeń drzewa), a następnie przeprowadzenie operacji down_heap czyli naprawy drzewa od góry do dołu (od korzenia do liści) aby naprawić kopiec po to by spełniał wyżej wspomniane własności.
Dodanie nowego elementu do kopca odbywa się w taki sposób, że jest on dodawany na koniec tablicy (po prawej stronie ostatniego liście w drzewie lub tworzy nowy poziom jeśli poprzedni jest już zapełniony). Następnie przeprowadza się operację up_heap czyli naprawę kopca przeprowadzaną od dołu do góry (od liści do korzenia).

Badania

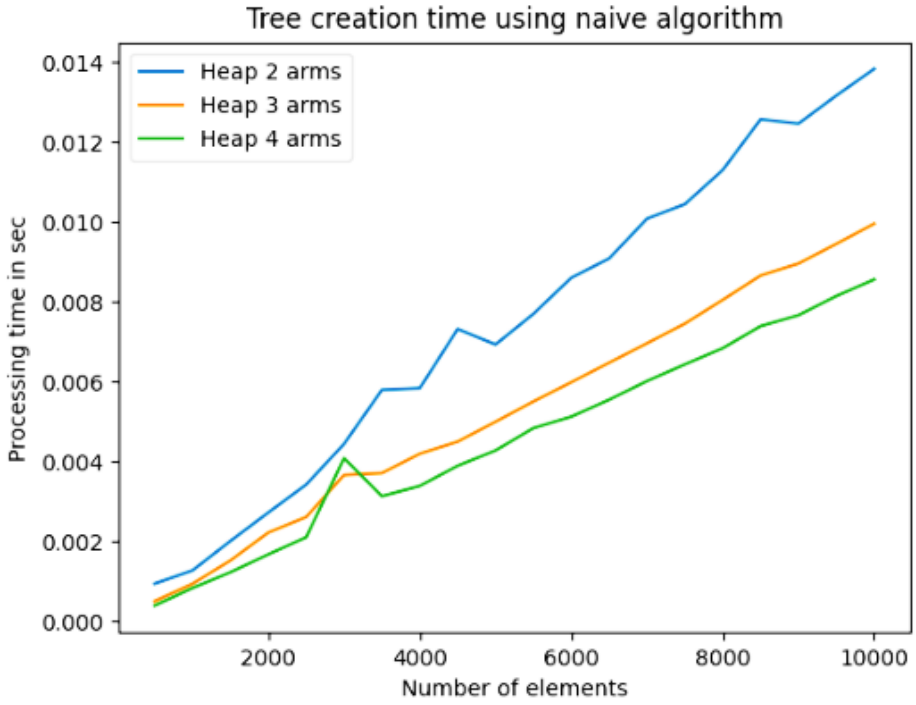
Badanie przeprowadzono dla 10 000 niepowtarzających się losowych danych.

Tworzenie drzewa metodą make_heap czyli odtwarzania od przedostatniego elementu w górę:



Najszybciej tworzyło się drzewo 4arne ze względu na to, że ma mniejsza wysokość co znacznie przyspiesza znalezienie odpowiedniego miejsca dla nowych wartości.

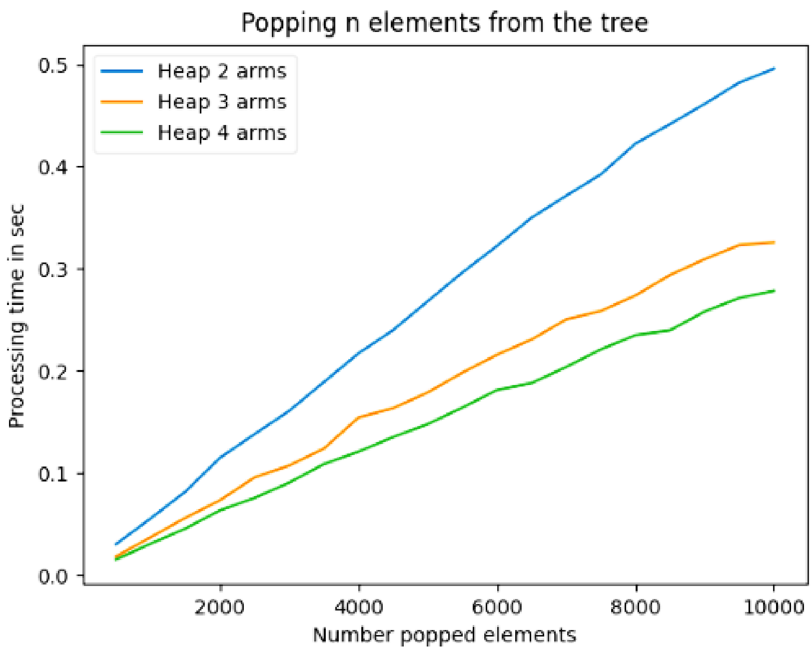
Tworzenie drzewa algorytmem naiwnym:



Wyniki były bardzo zaskakujące ze względu na to, że kopiec tworzony algorytmem naiwnym (metoda push dla każdego nowego elementu) działała szybciej. Wniosek jaki wysnuliśmy był taki że dla tego rodzaju danych oraz ilości danych złożoność nlog(n) sprawdza się lepiej niż złożoność liniowa.

Podobnie jak w poprzednim przypadku najszybciej tworzył się kopiec 4arny.

<u>Usunięcie szczytu kopca:</u>



Podobnie możemy zauważyć że wraz ze wzrostem liczby dzieci (liczby n) czasy zarówno tworzenia kopca jak i usuwania z niego wartości są mniejsze. Taka sytuacja ma prawdopodobnie pewien próg liczby dzieci kiedy przestanie się opłacać dodawać liczbę dzieci przez to, że zwiększy się koszt znajdowania najmniejszej wartości wśród dzieci rodzica w przypadku operacji down_heap.

Kopce przedstawiają się następująco:

```
------ Heap Two Arms diplay ------
                                        1
                    11                                        5
          22                    12                    8                   15
     26          33          38          34          46          35          19          28
  70    32    52    42    64    83    48    53    60    104    105    97    77    75    87    50
80    71    61    91    100    68    56    82    94
```

```
------ Heap Three Arms diplay ------
                                    1
             8                      5                      22
    12       11       15       26       33       38       34       46       35
 19 28 70 32 52 42 64 83 48 53 60 104 105 97 77 75 87 50 80 71 61 91 100 68 56 82 94
```

```
------ Heap Four Arms diplay ------
                                    1
         8                  5                  19                  12
  11   15   26   33   38   34   46   35   22   28   70   32   52   42   64   83
48  53  60  104  105  97  77  75  87  50  80  71  61  91  100  68
------ end diplay ------
```