Piotr Adrian Brzeziński
Bartłomiej Patryk Rasztabiga
Laboratorium AISDI - grupa 104

Kopce - porównanie wydajności

Opis implementacji

Zaimplementowano 3 rodzaje kopców minimalnych: 2-arny, 3-arny oraz 4-arny. Dodatkowo umożliwiono stworzenie kopca d-arnego.

Kod kopca zawiera metodę *push*, która wstawia kolejny element, przywracając właściwość kopca za pomocą metody *shiftup* (heapify).

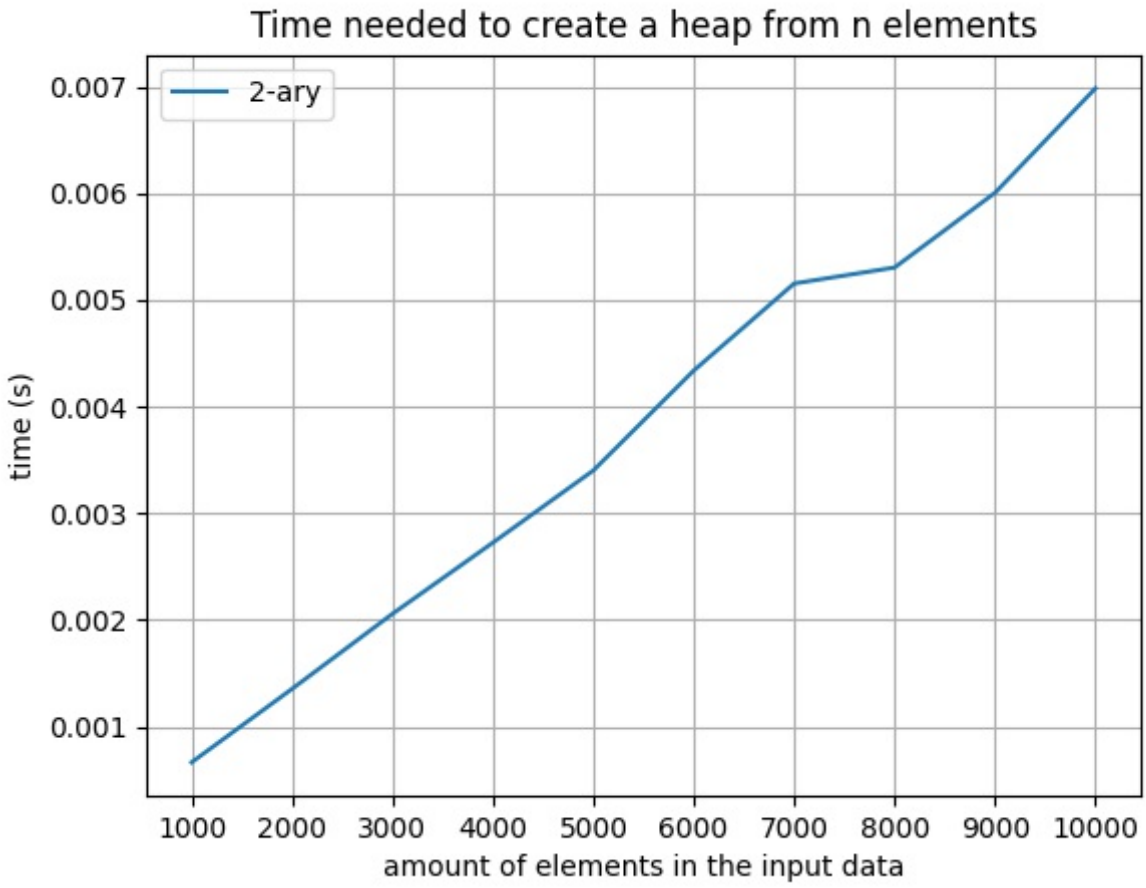
Zaimplementowano również metodę *print* umożliwiającą wyświetlenie kopca na ekranie.

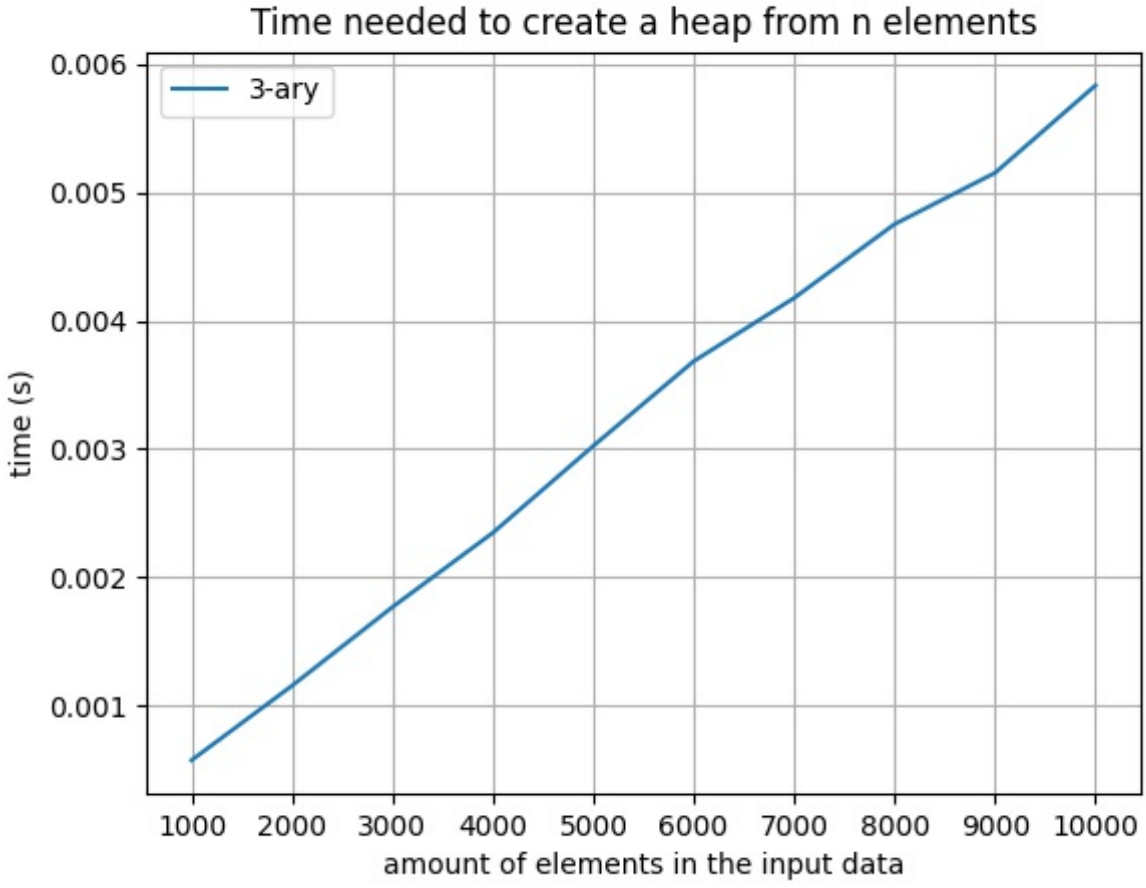
Sprawdzenie poprawności implementacji

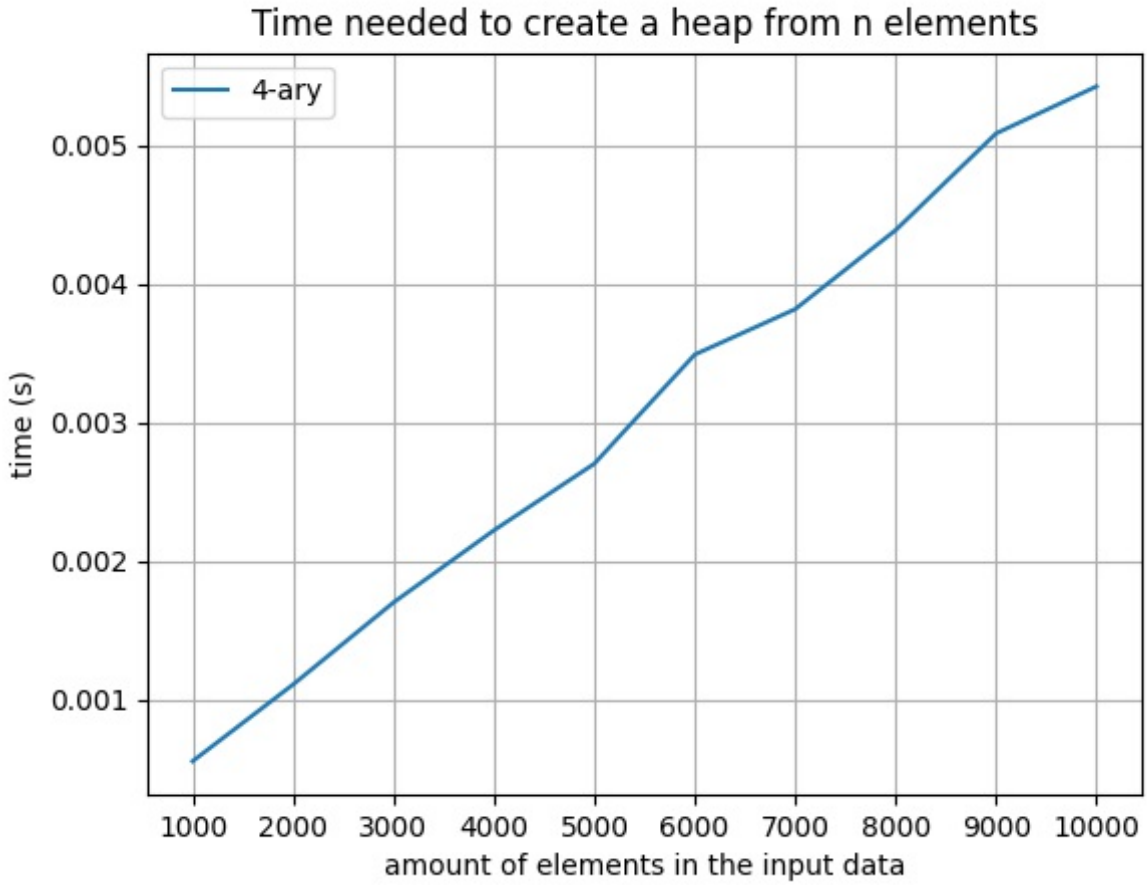
Poprawność implementacji została zagwarantowana przez testy jednostkowe w pliku *test_main.py*.

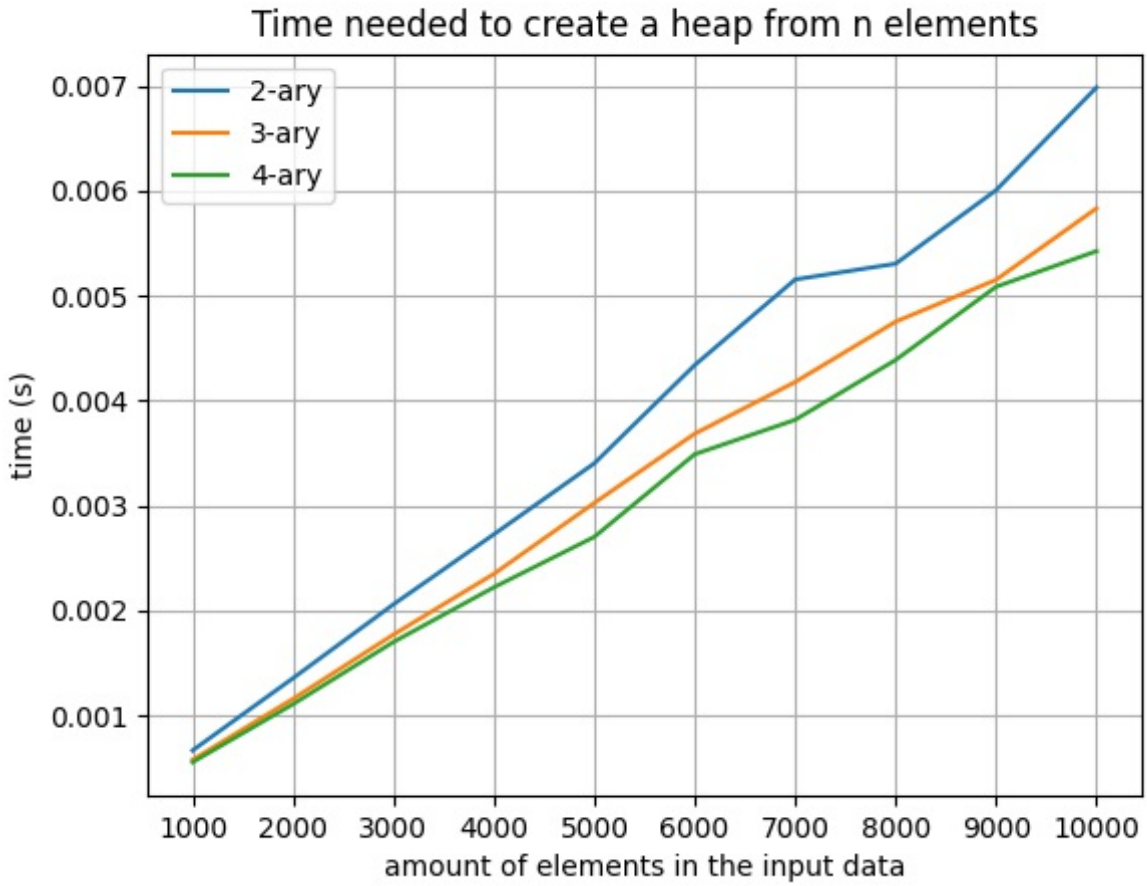
Dodatkowo zweryfikowano zgodność z wizualizacją dostępną na stronie https://dheap.blackpinguin.de/ (https://dheap.blackpinguin.de/).

Tworzenie kopców


Time needed to create a heap from n elements
2-ary
0.007
0.006
0.005
0.004
0.003
0.002
0.001
time (s)
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
amount of elements in the input data


Time needed to create a heap from n elements
3-ary
0.006
0.005
0.004
0.003
0.002
0.001
time (s)
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
amount of elements in the input data


Time needed to create a heap from n elements
4-ary
0.005
0.004
0.003
0.002
0.001
time (s)
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
amount of elements in the input data


Time needed to create a heap from n elements
2-ary
3-ary
4-ary
0.007
0.006
0.005
0.004
0.003
0.002
0.001
time (s)
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
amount of elements in the input data

Dane do wykresów:

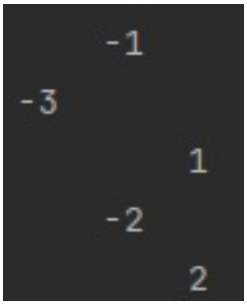
```
Tworzenie kopców z n elementów:
1000:   2-ary: 0.000667s        3-ary: 0.00057791s      4-ary: 0.00055483s
2000:   2-ary: 0.0013552s       3-ary: 0.00115773s      4-ary: 0.00110783s
3000:   2-ary: 0.0020584s       3-ary: 0.00177007s      4-ary: 0.00169978s
4000:   2-ary: 0.0027256s       3-ary: 0.00234852s      4-ary: 0.00222062s
5000:   2-ary: 0.0034018s       3-ary: 0.00302471s      4-ary: 0.0027013s
6000:   2-ary: 0.004338s        3-ary: 0.00368407s      4-ary: 0.00348874s
7000:   2-ary: 0.0051538s       3-ary: 0.00417639s      4-ary: 0.00381633s
8000:   2-ary: 0.0053037s       3-ary: 0.00475141s      4-ary: 0.00438649s
9000:   2-ary: 0.0060041s       3-ary: 0.00515022s      4-ary: 0.00508412s
10000:  2-ary: 0.0069832s       3-ary: 0.0058309s       4-ary: 0.00542357s
```

Platforma, na której testowano wydajność:

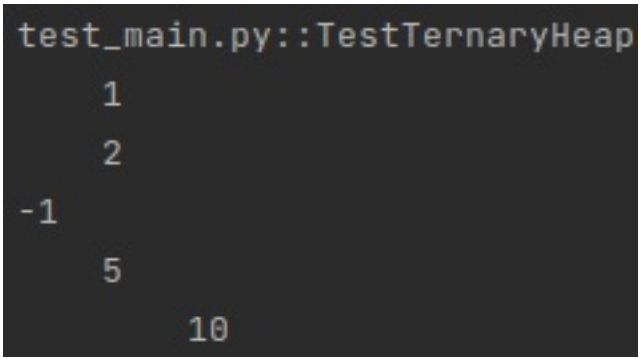
```
Kernel: 5.12.1-2-MANJARO x86_64
Distro: Manjaro Linux
CPU: 6-Core Intel Core i7-10750H
Mem: 12830.9/31853.0 MiB (40.3%)
Python 3.9.4
```

Wyświetlanie kopców

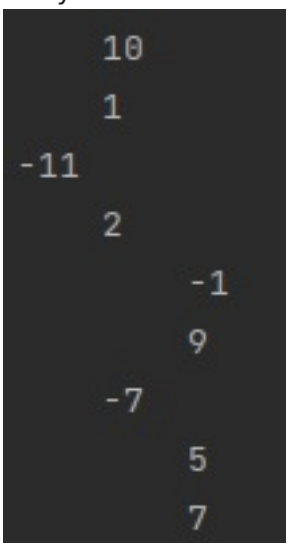
2-ary:



3-ary:



4-ary:



Wnioski

Dodawanie elementów do kopca 4-arnego jest najszybsze, ponieważ posiada on najmniejszą wysokość ze wszystkich testowanych kopców. Przez to metoda przywracająca właściwość kopca ma do wykonania najmniej przeniesień.

Złożoność metody push to O(wysokość) = O(log(d,n)).