Piotr Klimkowski
nr indeksu. 226252 05.04.17

Sprawozdanie z laboratorium nr3.

1.Cel

Stworzenie struktur danych: kolejka, stos i lista.
Stworzenie interfejsów do każdej struktury.
Stworzenie interfejsu obsługującego każdą powyższą strukturę.

2.Realizacja

Każda ze struktur została oparta na węzłach (Ang. Node).

```
class Node
{
    Node *prev;   //wskaznik na poprzednie ogniwo
    Node *next;   //wskaznik na nastepne ogniwo
    int value;    //zawartosc ogniwa
…
}
```

Węzły dzięki wskaźnikom pozwalają na bardzo proste przechodzenie po elementach. W odróżnieniu od tablic nie jesteśmy w stanie dotrzeć do konkretnego elementu od razu, konieczne jest przechodzenie po kolei zaczynając od pierwszego lub ostatniego węzła w strukturze.

Struktury w zależności od swoich potrzeb zawierają wskaźniki tail oraz head, które wskazują na ostatnio dodany i pierwszy element. Ogólnie patrząc każda ze struktur jest listą różnią się one jedynie metodami wyszukiwania oraz „wyrzucania" elementów. W stosie metoda *pop();* wyrzuci ostatnio dodany element oraz przypisze *ogonowi* adres przed ostatniego elementu, w kolejce zaś wyrzucony zostanie pierwszy element a adres drugiego zostanie przypisany *głowie.*

3. Pomiary

Wykorzystując główny interfejs *iRunable*, stworzona została funkcja przyjmująca referencje do obiektu głównego interfejsu. W środku funkcji zostaje dodana zadana ilość elementów, element poszukiwany (w tym wypadku „2") zostaje dodany w losowo wybranym miejscu z przedziału 25% wszystkich elementów do 75% wszystkich elementów (prostszymi slowami, jeśli zadamy ze funkcja ma dodac 100 elementow do struktury, to szukana wartosc losowo pojawi się w przedziale od 25 do 75 elementu). Następnie startuje stoper i zostaje wykonana metoda *find_value(int);*. Zliczony czas zostaje wrzucony do dynamicznej tablicy a struktura zostaje wyczyszczona *flush();*. Na koniec ze wszystkich pomiarów zostaje wyliczona średnia, i na ekran terminalu wyrzucony zostaje komunikat informujacy o strukturze jaka badalismy, dla jakiej ilosci elementów oraz ile razy został powtórzony pomiar.

```
---------------------------------------------
Stack
100 elementow-4.4 mikrosekund
zrobilem to: 10 razy
---------------------------------------------
//Przykładowy komunikat dla wywołania programu - ./program -100 10 s
```

3.1 Uruchamianie programu

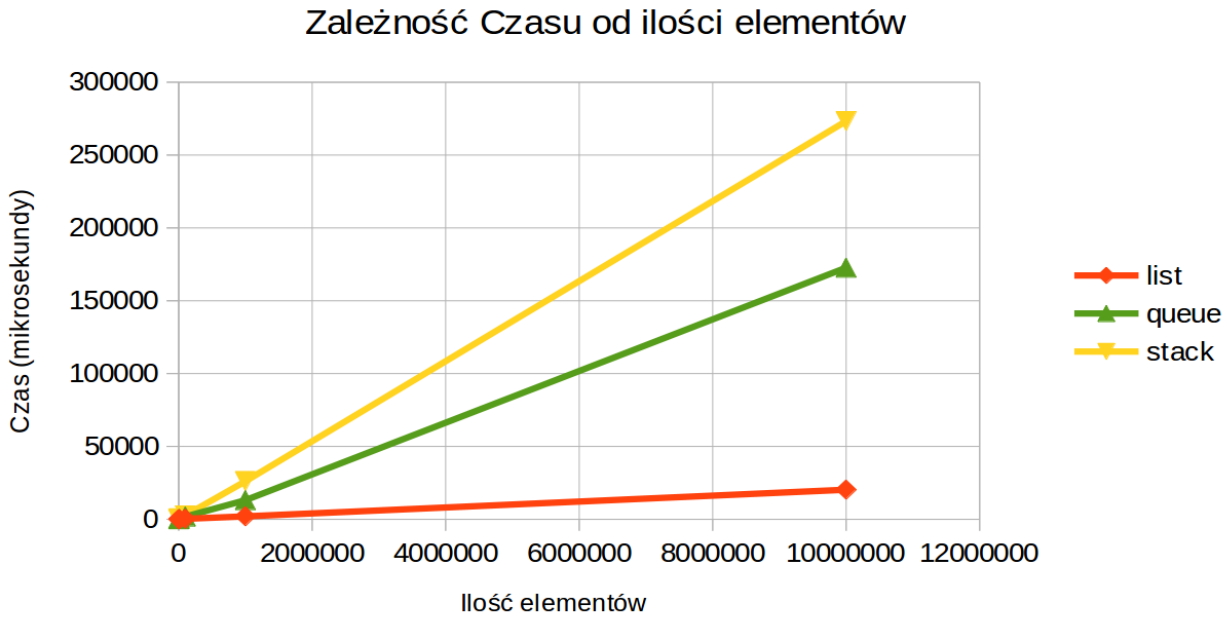
./program -a b c

a- ilosc elementów jakie mają być dodane do struktury

b- ilość powtórzeń pomiaru

c- wybór struktury „s”tack, „q”ueue, „l”ist.

4.Wyniki



wykres nr1.

5.Wnioski

Na podstawie wkyresu nr1. można zauważyć, że najszybciej wyszukiwanie działa dla listy. Dzieje się tak ponieważ lista jako jedyny algorytm wyszukuje wartość przechodząc po kolejnych węzłach i sprawdzając czy ich *wartość==szukana*, dla stosu oraz kolejki z powodu innej specyfikacji struktury, wyszukiwanie polega na wysuwaniu po kolei elementów *pop();* i sprawdzaniu czy dany wartość danego elementu odpowiada tej szukanej.

6.Spostrzeżenia

Podczas pisania programu wielokrotnie pojawiał się błąd naruszenia ochrony pamięci, który występował przy próbach usunięcia ostatniego elementu struktury.

Metoda

```
remove(Node* a){
        if(a→next==a→prev){
                head=nullptr;
                tail=nullptr;
        }
        …
        delete a;
        size--;
}
```

w przypadku próby usunięcia ostatniego elementu (oba jego wskaźniki next i prev wskazuja wtedy na nullptr) head oraz tail przyjmowały wskaźnik na puste miejsca. Powodowało to wyżej wspomniany błąd, ale tylko w jednej strukturze. Z racji że stos posiada jedynie wskaźnik na tail, problem nie występował, problem nie występował też w liście za to w strukturze queue powodowało to błąd naruszenia. Ostatecznie usunąłem powyższą część metody ponieważ nie puste wskaźniki tail i head nie powodowały problemów przy ponownym wypełnianiu struktury.