Programowanie Komputerów 3

Temat: Sprawozdanie

Seweryn Gładysz

Semestr: trzeci
Grupa dziekańska: III

1. Treść zadania

   Numer projektu 22

   Napisać w oparciu o szablon (template) bibliotekę do znajdywania w tablicy liczbowej wartości ekstremalnych oraz ich położenia.

   Zastosować algorytmy:

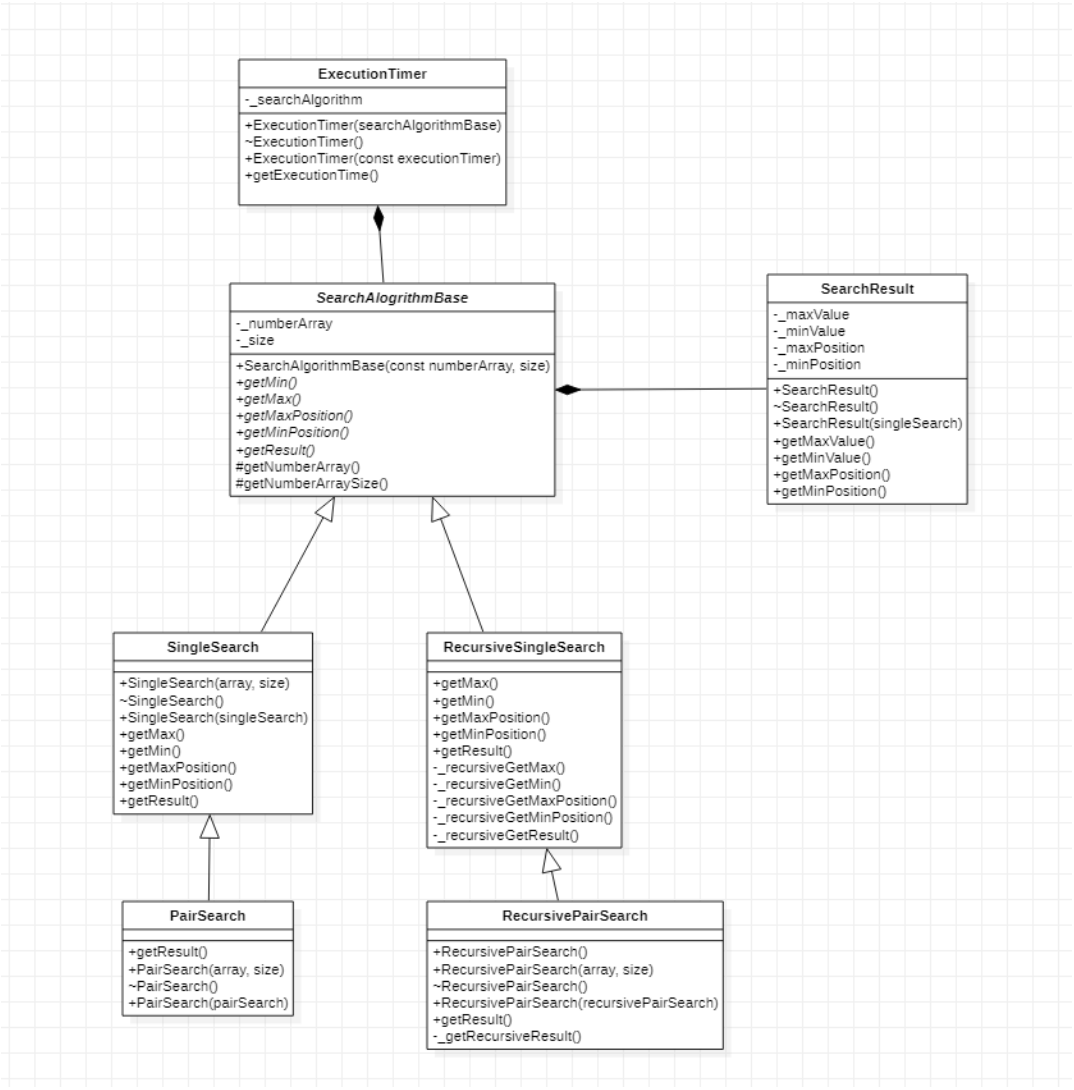
   a. pojedynczego przeglądania kolejnych elementów tablicy,
   b. przeglądania parami. Każdy z powyższych dwóch algorytmów zaimplementować w wersji:
      i. Iteracyjnej,
      ii. Rekurencyjnej.

Dodać metody dla mierzenia czasu wyszukiwania. Przeprowadzić porównanie dla min. jednego zestawu danych.

2. Analiza rozwiązania

Głównym zadaniem biblioteki jest szukanie na różne sposoby wartości ekstremalnych w podanej tablicy. Zdecydowałem się na wykorzystanie klas w celu późniejszego wykorzystania niektórych funkcji w metodach które będą wyszukiwać wartości ekstremalne parami.

Diagram klas:



Wyszukiwanie liniowe iteracyjne realizowane przez klasę *SingleSearch* polega na przeglądaniu kolejnych elementów tablicy i zapisywanie największej lub najmniejszej liczby oraz ich pozycji do zmiennej pomocniczej. Z wartości ekstremalnych oraz ich pozycji tworzony i zwracany jest obiekt klasy *SearchResult.* Drugą wersją szukania liniowego jest odmiana rekurencyjna. W odróżnieniu od wersji iteracyjnej, na każdym elemencie tablicy wywoływana jest rekurencyjnie funkcja sprawdzająca wartości ekstremalne oraz ich pozycje. Funkcja zwraca wynik gdy indeks przekazywany jako parametr zrówna się liczbie 0, która oznacza, że wszystkie elementy zostały sprawdzone.

Biblioteka powinna zawierać również możliwość szukania wartości ekstremalnych za pomocą wyszukiwania parami. Wersja iteracyjna jest realizowana przez klasę *PairSearch,* a rekurencyjna przez klasę *RescursivePairSearch*. W obu wersjach sprawdzane są dwa elementy naraz. Następnie są porównywane, większy może być elementem największym, a mniejszy może być elementem minimalnym. Następnie większa wartość jest sprawdzana z maksymalną wartością znalezioną w tablicy, a mniejsza wartość jest porównywana z minimalną wartością znalezioną w tablicy.

3. Specyfikacja zewnętrzna

Dlatego, że zadanie polega na utworzeniu bilbioteki do szukania wartości ekstremalnych to aby skorzystać z biblioteki wystarczy załączyć nagłówek biblioteki w następujący sposób: *#include "projekt_22*" oraz użycie interesującej nas klasy w zależności o tego w jaki sposób chcemy otrzymać wynik. Zamieszczam plik *main.cpp* w projekcie w którym znajdują się przykładowe wywołania dla wszystkich metod wyszukiwania oraz testy czasu wykonania zadania przez konkretne metody.

Biblioteka ma pewne założenia, które zostały zapożyczone z rozwiązań z innych języków programowania i bibliotek. Jeżeli funkcje nie będą mogły odnaleźć wartości ekstremalnych to w ich miejsce zostanie zwrócony *nullptr,* natomiast w przypadku pozycji tych wartości będzie to wartość *-1*.

4. Specyfikacja wewnętrzna
   a. *SearchAlgorithmBase<TNumber>*

Metoda	Opis
SearchAlgorithmBase(const TNumber, size)*	Konstruktor, który powinien być wykorzystywany domyślne. Jako argumenty przyjmuje wskaźnik na tablicę liczb oraz jej rozmiar.
virutal TNumber getMax()*	Wirtualna metoda, klasy dziedziczące po tej klasie będą za jej pomocą zwracać wartość maksymalną
virtual TNumber getMin()*	Wirtualna metoda będąca odpowiednikiem *getMax* tylko dla wartości minimalnych
virtual long long int getMaxPosition()	Również metoda czysto wirtualna (*pure virtual).* Za jej pomocą zwracana jest pozycja wartości maksymalnej.
virtual long long int getMinPosition()	Odpowiednik *getMaxPosition* dla wartości minimalnej.
virtual SearchResult<TNumber> getResult()	Zwraca wynik wyszukiwania wartości ekstremalnych i ich pozycji w postaci obiektu *SearchResult<TNumber>*

   b. *SingleSearch<TNumber> : public SearchAlgorithmBase<TNumber>*

Metoda	Opis

SearchAlgorithmBase(const TNumber, size)*	Konstruktor, który powinien być wykorzystywany domyślne. Jako argumenty przyjmuje wskaźnik na tablicę liczb oraz jej rozmiar.
virutal TNumber getMax()*	Nadpisuje metodę klasy bazowej *SearchAlgorithmBase<TNumber>* i realizuje to za pomocą wyszukiwania liniowego iteracyjnego.
virtual TNumber getMin()*	Wirtualna metoda będąca odpowiednikiem *getMax* tylko dla wartości minimalnych
virtual long long int getMaxPosition()	Również metoda czysto wirtualna (*pure virtual).* Za jej pomocą zwracana jest pozycja wartości maksymalnej.
virtual long long int getMinPosition()	Odpowiednik *getMaxPosition* dla wartości minimalnej.
virtual SearchResult<TNumber> getResult()	Zwraca wynik wyszukiwania wartości ekstremalnych i ich pozycji w postaci obiektu *SearchResult<TNumber>.* Realizuje zadanie za pomocą wyszukiwania iteracyjnego liniowego.

c. *PairSearch<TNumber> : public SingleSearch<TNumber>*

Metoda	Opis
SearchAlgorithmBase(const TNumber, size)*	Konstruktor, który powinien być wykorzystywany domyślne. Jako argumenty przyjmuje wskaźnik na tablicę liczb oraz jej rozmiar.
virtual SearchResult<TNumber> getResult()	Zwraca wynik wyszukiwania wartości ekstremalnych i ich pozycji w postaci obiektu *SearchResult<TNumber>.* Realizuje zadanie za pomocą wyszukiwania iteracyjnego parami.

d. RecursiveSingleSearch<TNumber> : public SearchAlgorithmBase<TNumber>

Metoda	Opis
SearchAlgorithmBase(const TNumber, size)*	Konstruktor, który powinien być wykorzystywany domyślne. Jako argumenty przyjmuje wskaźnik na tablicę liczb oraz jej rozmiar.
virutal TNumber getMax()*	Nadpisuje metodę klasy bazowej *SearchAlgorithmBase<TNumber>* i realizuje to za pomocą wyszukiwania rekurencyjnego liniowego.
virtual TNumber getMin()*	Wirtualna metoda będąca odpowiednikiem *getMax* tylko dla wartości minimalnych.

virtual long long int getMaxPosition()	Również metoda czysto wirtualna (*pure virtual).* Za jej pomocą zwracana jest pozycja wartości maksymalnej.
virtual long long int getMinPosition()	Odpowiednik *getMaxPosition* dla wartości minimalnej.
virtual SearchResult<TNumber> getResult()	Zwraca wynik wyszukiwania wartości ekstremalnych i ich pozycji w postaci obiektu *SearchResult<TNumber>.* Realizuje zadanie za pomocą wyszukiwania rekurencyjnego liniowego.

e. *RecursivePairSearch<TNumber> : public RecursiveSingleSearch<TNumber>*

Metoda	Opis
SearchAlgorithmBase(const TNumber, size)*	Konstruktor, który powinien być wykorzystywany domyślne. Jako argumenty przyjmuje wskaźnik na tablicę liczb oraz jej rozmiar.
virtual SearchResult<TNumber> getResult()	Zwraca wynik wyszukiwania wartości ekstremalnych i ich pozycji w postaci obiektu *SearchResult<TNumber>.* Realizuje zadanie za pomocą wyszukiwania rekurencyjnego parami.

f. *SearchResult<TNumber>*

Metoda	Opis
SearchResult()	Konstruktor inicjalizujący pusty obiekt
SearchResult(TNumber maxValue, TNumber *minValue, long long int maxPosition, long long int minPosition)*	Konstuktor inicjalizujący obiekt z wartościami.
const TNumber getMaxValue() const*	Metoda zwraca stałą wartość maksymalną znalezioną w tablicy.
const TNumber getMinValue() const*	Metoda zwraca stałą wartość minimalną znalezioną w tablicy.
long long int getMaxPosition() const	Metoda zwracająca indeks tablicy, gdzie odnaleziono wartość maksymalną.
long long int getMinPosition() const	Metoda zwracająca indeks tablicy, gdzie odnaleziono wartość minimalną.

g.

Dla wszystkich klasy zostały zaimplementowane odpowiednie przeładowania operatora = oraz konstruktory kopiujące. W przypadku klas dziedziczących po *SearchAlgorithmBase<TNumber>* konstruktor kopiujący i operator = zostało wykorzystane słowo kluczowe *default,* ponieważ domyślny konstruktor i operator zapewniają odpowiednie zachowanie klasy.

Implementacje destruktorów klas dziedziczących po klasie *SearchAlgorithmBase* również zostały zachowane domyślne, ponieważ klasa nie powinna czyścić tablicy, którą miała jedynie przeszukać.

5. Testowanie

Zadanie zakłada przypadki testowe o następującej treści:

Dodać metody dla mierzenia czasu wyszukiwania. Przeprowadzić porównanie dla min. jednego zestawu danych.

W tym celu został utworzony plik nagłówkowy *projekt_22_tests.h.* Jego zadaniem jest wypisanie czasu wykonania wyszukiwania. W celu uproszczenia testowania utworzona została klasa *ExecutionTimer,* której zadaniem jest właśnie sprawdzenie wcześniej wspomnianego czasu wykonania. Zadanie zostało zrealizowane za pomocą mechanizmu polimorfizmu.

Niżej zostaną zaprezentowane wyniki działania programu dla różnej wielkości tablic oraz systemów operacyjnych:

a. Windows (minGW) 8000 elementowa tablica typu *double*.

```
****** Mierzenie czas├│w wykonania ******
Pojedynczo iteracyjnie: 4.6e-05
Wynik: Max: 999 at: 50, min: -5 at: 137

Pojedynczo rekurencyjnie: 0.0030335
Wynik: Max: 999 at: 5774, min: -5 at: 7752

Parami iteracyjnie: 4.88e-05
Wynik: Max: 999 at: 50, min: -5 at: 137

Parami rekurencyjnie: 0.0050761
Wynik: Max: 422 at: 1, min: -5 at: 7752
```

b. Linux – Debian - 8000 elementowa tablica typu *double.*

```
****** Mierzenie czasów wykonania ******
Pojedynczo iteracyjnie: 3.87e-05
Wynik: Max: 999 at: 731, min: -5 at: 2870

Pojedynczo rekurencyjnie: 0.0009182
Wynik: Max: 999 at: 6288, min: -5 at: 7923

Parami iteracyjnie: 6.49e-05
Wynik: Max: 999 at: 2908, min: -5 at: 7191

Parami rekurencyjnie: 0.001292
Wynik: Max: 42 at: 0, min: -5 at: 7923
```

c. Windows (minGW) 25000 elementów typu *double*

```
Process finished with exit code -1073741571 (0xC00000FD)


Process finished with exit code -1073741571 (0xC00000FD)
```

W przypadku tego testu dochodzi do błędu spowodowanego z przeładowaniem stosu, którego limit w systemie Windows wynosi zaledwie 1MB. Ograniczenie to nie pozwala na poprawne wykorzystanie klas związanych z wyszukiwaniem rekurencyjnym. Nastomiast wyszukiwanie iteracyjnie działa poprawnie:

```
****** Mierzenie czas├┤w wykonania ******
Pojedynczo iteracyjnie: 0.0001052
Wynik: Max: 999 at: 720, min: -5 at: 46

Parami iteracyjnie: 0.0001631
Wynik: Max: 999 at: 720, min: -5 at: 705
```

d. Linux – Debian – 25000 elementów typu *double*

```
****** Mierzenie czasów wykonania ******
Pojedynczo iteracyjnie: 2.48e-05
Wynik: Max: 999 at: 990, min: -5 at: 639

Pojedynczo rekurencyjnie: 0.0007777
Wynik: Max: 999 at: 7871, min: -5 at: 7919

Parami iteracyjnie: 5.68e-05
Wynik: Max: 999 at: 990, min: -5 at: 639

Parami rekurencyjnie: 0.0010923
Wynik: Max: 508 at: 0, min: -5 at: 7919

****** Mierzenie czasów wykonania ******
Pojedynczo iteracyjnie: 2.48e-05
Wynik: Max: 999 at: 990, min: -5 at: 639

Pojedynczo rekurencyjnie: 0.0007777
Wynik: Max: 999 at: 7871, min: -5 at: 7919

Parami iteracyjnie: 5.68e-05
Wynik: Max: 999 at: 990, min: -5 at: 639

Parami rekurencyjnie: 0.0010923
Wynik: Max: 508 at: 0, min: -5 at: 7919
```

W przeciwieństwie do

systemu *Windows, Linux* ma możliwość zmiany wielkości jaką można zmagazynować na stosie. Domyślną wartością jest 8MB, które pozwalają na poprawne wyszukiwanie rekurencyjne.