01.12.2020

Algorytmy i Struktury Danych

Sprawozdanie z projektu

Grupa Projektowa nr. 5

Jakub Matuszko

Numer indeksu: 166673

Treść projektu: Uporządkuj zadaną tablicę wielu powtarzających się elementów, tak aby zgromadzić te same elementy zachowując kolejność ich pierwszego wystąpienia.

# Treść projektu:

Uporządkuj zadaną tablicę wielu powtarzających się elementów, tak aby zgromadzić te same elementy zachowując kolejność ich pierwszego wystąpienia.

# Wymagane cechy programu:

a) główny kod programu powinien być zaimplementowany w oddzielnej funkcji, którą powinna być wywoływana wewnątrz programu

b) w głównym programie powinno zostać wykonane kilka testów sprawdzających działanie funkcji

c) program powinien mieć możliwość odczytywania danych wejściowych i zapisu wyników do plików tekstowych

d) kod powinien być opatrzony stosownymi komentarzami

# Przykłady:

Wejście: [ 1, 2, 3, 1, 2, 1 ]

Wyjście: [ 1, 1, 1, 2, 2, 3 ]

Wejście: [ 5, 4, 5, 5, 3, 1, 2, 2, 4 ]

Wyjście: [ 5, 5, 5, 4, 4, 3, 1, 2, 2 ]

# Wybrany język programowania: C++

# Wybrane środowisko: Code::Blocks

# Wejście programu

Plik input.txt z liczbami całkowitymi mniejszymi od 2’147’483’647 i większymi od – 2’147’483’647. Liczby tę są oddzielone spacją lub enterem.

# Wyjście programu

Plik output.txt z liczbami całkowitymi mniejszymi od 2’147’483’647 i większymi od – 2’147’483’647. Oddzielone średnikiem i spacją.

# Problematyka algorytmu:

Głównym założeniem algorytmu jest sortowanie wprowadzonych elementów względem ich wystąpienia. W przypadku tego algorytmu ważnym zadaniem jest zachowanie ich odpowiedniej kolejności, do tego została użyta funkcja *swap* z malejącą pętlą *k –* zastosowanie tych dwóch funkcji pozwala przestawiać elementy obok siebie z zachowaniem poprawnej kolejności ich występowania. Na przykład wprowadzając liczby *[1;4;5;1;3]* sama funkcja *swap* zamieniłaby kolejnością liczbę *1* z *4 –* dałoby to wynik *[1;1;5;4;3]* taka operacja spowodowałaby że początkowa kolejność występowania elementów została zmieniona. Gdy użyjemy funkcji *swap* z pętlą *k* dostaniemy wynik *[1;1;4;5;3]*, ten wynik zachowuje prawidłową kolejność.

# Opis programu:

## Funkcja główna (main)

Program wczytuje liczby z pliku *input.txt*, zapisuje je do wektora *numbers* dopisując każdą na końcu. Program jest zabezpieczony przed błędnym wczytaniem pliku, w przypadku gdy *input.good()* zwróci wartość fałsz - program przekaże do konsoli komunikat o błędzie pliku i zakończy program. W przeciwnym wypadku zostanie rozpoczęte wczytywanie liczb do zmiennej tymczasowej *n* która odpowiada za przechowywanie wartości liczby z pliku. Następnie program sprawdzi czy wczytywana liczba jest poprawna, gdy okaże się że wejście wskazuje błąd to funkcja *input.clear()* usunie flagę z błędem, a następnie funkcja *input.ignore()* zignoruje jeden znak. Czynność ta będzie się powtarzać do napotkania poprawnych danych wejściowych. W przypadku gdy liczby są połączone z nie właściwym znakiem to program pominie złe znaki np. gdy jest wprowadzona liczba *121ab15* to algorytm wskaże że są to dwie liczby 12 i 15, pomijając przy tym nie poprawne znaki. Po poprawnym wczytaniu liczby zostaje ona dopisana na koniec wektora za pomocą funkcji *numbers.push\_back(n)*.

## Funkcja swap

Założeniem tej funkcji jest przestawiać dwie liczby miejscami. Wczytywane są adresy dwóch zmiennych zawierające liczby, następnie liczba pierwsza jest zapisywana do zmiennej *temp*. Kolejnym krokiem jest przypisanie wartości drugiej liczby do zmiennej pierwszej liczby. Po wykonaniu tych operacji do drugiej zmiennej jest przypisywana wartość zmiennej *temp*.

## Funkcja sort

Funkcja wczytuje adres wektora liczb *numbers*. Zostają utworzone dwie pętle o indeksach *i* i *j*, pierwszy z nich rozpoczyna się od początku wektora, a drugi zaczyna od liczby o jeden większej od indeksu *i.* Zadaniem indeksu *i* jest wybranie pierwszej liczby, a indeksu *j* poszukiwanie liczb równych do pierwszej liczby. W przypadku gdy wartości liczby z miejsca *i* i *j* są równe sobie,zostanie rozpoczęta kolejna pętla której zdaniem jest przesunięcie liczby z pozycji *j* do pozycji *i+1* przy użyciu funkcji *swap*. Dzięki temu wszystkie liczby równe sobie znajdują się obok siebie. Po każdym takim odnalezieniu, indeks *i* zostaje przesunięty o jedną pozycję w prawo – taki zabieg pozwala przejść od razu do liczby innej niż ta pierwsza i pomija zbędne operacje.

# Użyte zmienne:

## Funkcja main:

*input* – zmienna pliku wejściowego,

*output* – zmienna pliku wyjściowego,

*n* – zmienna liczb całkowitych (Tymczasowo do niej zapisywane są liczby z pliku),

*numbers* – wektor liczb całkowitych,

## Funkcja swap:

*temp* – zmienna liczb całkowitych (do niej jest przypisywana wartość pierwszej liczby),

*a ­*– zmienna liczb całkowitych,

*b* – zmienna liczb całkowitych,

## Funkcja sort:

*i* – zmienna liczb całkowitych, używana w pętli jako indeks *i,*

*j* – zmienna liczb całkowitych, używana w drugiej pętli jako indeks *j,*

*k* – zmienna liczb całkowitych, używana do pętli przesuwającej elementy.

Wszystkie zmienne liczb całkowitych ograniczone są zakresem od -2’147’483’647 do 2’147’483’647.

# Pseudokod funkcji sort

1. Przekazanie adresu wektora do funkcji sort,
2. przypisanie wartości zmiennej: i = 0,
3. jeżeli i nie jest mniejsze od rozmiaru wektora przejdź do kroku 10, w przeciwnym wypadku przejdź dalej,
4. przypisz wartość zmiennej: j = i + 1,
5. jeżeli j nie jest mniejsze od rozmiaru wektora zwiększ wartość zmiennej i o 1 (i++) i przejdź do kroku 03, w przeciwnym wypadku przejdź dalej,
6. sprawdź czy liczba z pozycji i jest równa liczbie z pozycji j: jeżeli tak to przejdź dalej, w przeciwnym wypadku zwiększ wartość j o 1 (j++) i przejdź do kroku 05,
7. przypisz wartość zmiennej: k = j
8. dopóki k jest większe od i + 1, wykonuj: Zamień liczbę z miejsca o indeksie k z liczbą z miejsca o indeksie k – 1, następnie zmniejsz wartość zmiennej k o 1 (k--), po zakończeniu przejdź dalej,
9. zwiększ wartość zmiennej: i++; j++ i przejdź do kroku 05,
10. Zwróć posortowaną tablicę

# Schemat blokowy funkcji sort

# 

Rysunek Schemat blokowy algorytmu funkcji sort

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Wektor: | | 1 | | 2 | | 3 | | 1 | | 2 | | 1 |
| Indeks | | [0] | | [1] | | [2] | | [3] | | [4] | | [5] |
| Operacja 1 | | i = 0; | | j = 1; | | j = 2; | | j = 3; i=j; | |  | |  |
| Operacja 2 | |  | |  | |  | | Swap(3,2) | |  | |  |
| Operacja 3 | |  | |  | | Swap(2,1) | |  | |  | |  |
| Wektor: | | 1 | | 1 | | 2 | | 3 | | 2 | | 1 |
| Operacja 4 | |  | | i = 1; | | j = 2; | | j = 3; | | j = 4; | | j = 5; i=j; |
| Operacja 5 | |  | |  | |  | |  | |  | | Swap(5,4) |
| Operacja 6 | |  | |  | |  | |  | | Swap(4,3) | |  |
| Operacja 7 | |  | |  | |  | | Swap(3,2) | |  | |  |
| Wektor: | | 1 | | 1 | | 1 | | 2 | | 3 | | 2 |
| Zakończenie pętli 1; i++ | | | | | | | | | | | | |
| Wektor: | | 1 | | 1 | | 1 | | 2 | | 3 | | 2 |
| Operacja 8 | |  | |  | |  | | i = 3; | | j = 4; | | j = 5; i=j; |
| Operacja 9 | |  | |  | |  | |  | |  | | Swap(5,4) |
| Wektor: | | 1 | | 1 | | 1 | | 2 | | 2 | | 3 |
| Operacja 10 | |  | |  | |  | |  | | i = 4; | | j = 5; |
| Zakończenie pętli 2; i++ | | | | | | | | | | | | |
| Operacja 11 | |  | |  | |  | |  | |  | | i = 5; |
| Pętla j się nie rozpocznie ponieważ warunek j = i + 1 przekroczy wielkość wektora | | | | | | | | | | | | |
| Wektor: | 1 | | 1 | | 1 | | 2 | | 2 | | 3 | |

# Przykładowy sposób działania algorytmu sortującego

Tabela Sposób działania algorytmu sortującego

# Testy programu

## Test 1.0

Wprowadzone są do programu liczby wraz z literami:

Wejście:



Wyjście:



Wniosek:

Program wczytał dane z pliku, algorytm wykrył błąd wejścia i pominął znaki które nie są cyframi.

## Test 2.0

Wprowadzone są do programu liczby połączone z literami:

Wejście:



Wyjście:



Wniosek:

Program wczytał dane z pliku, w przypadkach gdy liczby były połączone z literami program usunął litery i oddzielił liczby. Gdy liczba jest połączona z literą bądź znakiem specjalnym i znowu z liczbą program interpretuje to jako dwie różne czyli *1ac3* program wczyta jako liczby 1 i 3.

## Test 3.0

Wprowadzone są do programu tylko litery:

Wejście:



Wyjście:



Wniosek:

Program wczytał litery z pliku, w każdym przypadku wykrył błąd. Do wektora nie zostały dodane żadne liczby więc program zakończył z pustym wynikiem.

## Test 4.0

Wprowadzone są tylko liczby:

Wejście:



Wyjście:



Wniosek:

Program wczytał liczby z pliku, wynik został wyświetlony poprawny.

## Test 5.0

Wprowadzone są liczby o zakresie większym niż int (2’147’483’647):

Wejście:



Wyjście:



Wniosek:

Program wczytał liczby z pliku, wszystkie które przekroczyły wartość 2’147’483’647 zostały odrzucone. Liczby które były mniejsze od graniczącej wartości zostały posortowane.