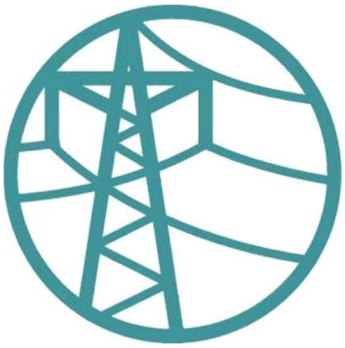


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Temat pracy:

Dla zadanej tablicy dwuwymiarowej o rozmiarze MxN przesuń wszystkie jej elementy w kierunku „spiralnym”

Dominik Maciąg

nr. indeksu: 15776

Spis treści

1. Wstęp

Celem ćwiczenia było zaprojektowanie i zaimplementowanie algorytmu służącego do przesuwania wszystkich elementów dowolnej zadanej tablicy w kierunku spiralnym. Algorytm zaimplementowano w języku C++.

2. Ogólny opis programu

Użytkownik ma możliwość wyboru pomiędzy ręcznym tworzeniem tablicy, a zaimportowaniem jej z pliku txt. W pierwszym przypadku z klawiatury deklarowane są wymiary tablicy, a następnie jej elementy mogą być uzupełnione manualnie lub poprzez wypełnienie losowymi liczbami całkowitymi. W drugiej opcji program korzysta z wcześniej stworzonej przez niego tablicy utworzonej przy pomocy funkcji „save_table_to_txt()". Tablica jest tworzona przy pomocy szablonu klasy vector. Umożliwia to podanie jej wymiarów po uruchomieniu programu.

3. Algorytm

3.1. Opis implementacji

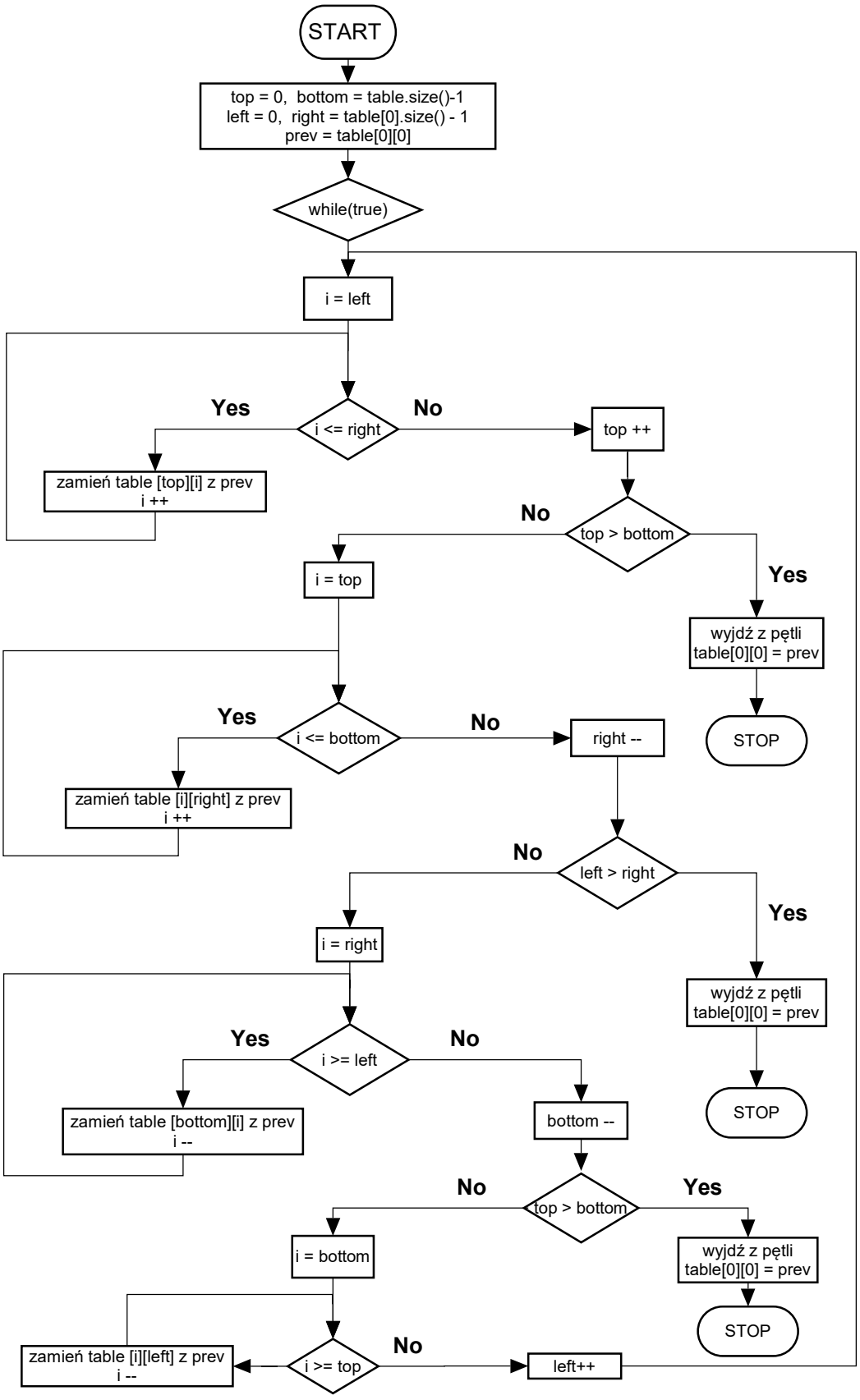
Funkcja „spiral_spin" odpowiada za przesunięcie elementów tablicy w kierunku spiralnym. Przyjmuje ona za argument adres pamięci stworzonej lub zaimportowanej tablicy.

Początkowo tworzone są zmienne przechowujące numer pierwszego i ostatniego wiersza oraz pierwszej i ostatniej kolumny oznaczone odpowiednio przez „top", „bottom", „left" i „right". Ponadto tworzona jest zmienna przechowująca pierwszy element tablicy oznaczona jako „prev".

Po inicjalizacji zmiennych utworzona została nieskończona pętla while, a w niej 4 niezależne występujące kolejno pętle for. Zadaniem każdej z nich jest przesuwanie elementów odpowiednio wiersza o najmniejszym numerze, kolumny o największym numerze, wiersza o najmniejszym numerze oraz kolumny o najmniejszym numerze.

Rozważmy pierwszą pętlę. Zmiennej iterowanej „i" przypisano wartość numeru skrajnej lewej kolumny, czyli 0. Warunkiem kończącym pętlę jest przypadek kiedy zmienna „i" wskazuje numer większy niż indeks ostatniej kolumny. Innymi słowy, kiedy to przesunięte zostały wszystkie elementy górnego wiersza. W ciele pętli skorzystano z funkcji swap, zamieniającej wartość zmiennej „prev" z odpowiednim elementem tablicy. Na przykład chcą przesunąć element z pozycji nr 3 na pozycję nr 4 do zmiennej „prev" trafia wartość elementu na miejscu nr 4, a element nr 3 znajduje się teraz na pozycji nr 4. Dzięki temu możliwe jest przypisanie kolejnym elementom wartości elementów poprzednich. Jak łatwo zauważyć, po wyjściu z pętli zmienna „prev" posiada teraz wartość ostatniego elementu górnego wiersza. Do zmiennej „top" dodawana jest teraz jedynka, ponieważ pierwszym wierszem, w którym elementy nie są przestawione jest wiersz o indeksie 1. W przypadku, kiedy to numer górnego wiersza będzie większy niż numer ostatnio zamienianego dolnego wiersza pętla zostaje przerwana, ponieważ sytuacja taka świadczy o tym, że wszystkie wiersze zostały ze sobą zamienione. Analogiczne przejścia zachodzą dla kolejnych kolumn i wierszy. Najpierw jest to prawa skrajna kolumna, dolny wiersz oraz lewa kolumna. W przypadku zamiany wszystkich elementów, pętla while zostaje przerwana i wartość ostatniego zamienionego argumentu zostaje przypisana pierwszemu elementowi tablicy.

3.2. Schemat blokowy algorytmu



Rysunek 3.1 Schemat blokowy algorytmu

3.3. Algorytm zapisany w pseudokodzie

Krok01: top ← 0
bottom ← ostatni indeks wiersza
left ← 0
right ← indeks ostatniej kolumny
prev ← pierwszy element tablicy

Krok02: Wykonuj pętlę nieskończoną:
i ← left
Dopóki i ≤ right:
Zamień element tabeli o indeksie [top][i] z poprzednim elementem
i ← i+1
Następnie:
top ← top +1

Krok03: Jeśli top > bottom:
Wyjdź z pętli
table[0][0] = prev
STOP algorytmu
Jeśli nie:
i ← top

Krok04: Dopóki: i ≤ bottom
Zamień element tabeli o indeksie [i][right] z poprzednim elementem
i ← i+1;
Następnie:
right ← right − 1

Krok05: Jeśli left > right:
Wyjdź z pętli
table[0][0] = prev
Stop algorytmu
Jeśli nie:
right ← right

Krok06: Dopóki i ≥ left:
Zamień element tabeli o indeksie [bottom][i] z poprzednim elementem
i ← i – 1;
Następnie:
bottom ← bottom – 1

Krok07: Jeśli top > bottom:
Wyjdź z pętli
table[0][0] = prev
STOP algorytmu
Jeśli nie:
i ← bottom

Krok08: Dopóki i ≥ top
Zamień element tabeli o indeksie [i][left] z poprzednim elementem
i ← i –1;
Następnie:
left ← left + 1

Krok09: Powtórz pętlę nieskończoną

3.4. Testy algorytmu

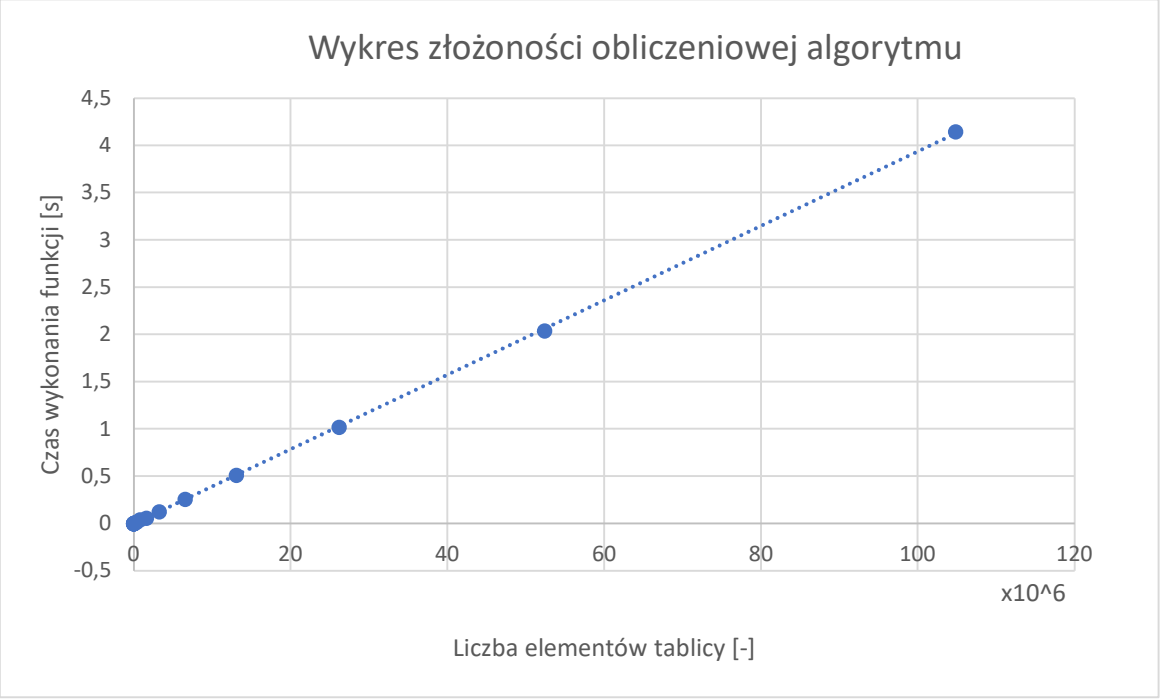
Program testowano na tablicach o liczbie wierszy wynoszącej 20. Liczbę kolumn zwiększano od 20 do $20 \cdot 2^{18}$ za każdym razem mnożąc ich liczbę przez 2. Za każdym razem wywołania funkcji testowano czas jej wykonania. Rezultaty zebrano w tabeli 3.1.

Czas wykonania [s]	Liczba elementów tablicy [-]
0	400
0	800
0	1600
0	3200
0	6400
0	12800
0	25600
0.0020503	51200
0.0019131	102400
0.0080417	204800
0.0157199	409600
0.039009	819200
0.055743	1638400
0.124943	3276800
0.254445	6553600
0.508801	13107200
1.01685	26214400
2.03768	52428800
4.14463	104857600

Tabela 3.1 Tabela zbiorcza wyników testów programu

3.5. Złożoność czasowa algorytmu

Na podstawie wykresu nr 3.1, można stwierdzić, że czas wykonania operacji jest zależny liniowo od liczby elementów tablicy.



Wykres 3.1 Złożoność obliczeniowa algorytmu

4. Wnioski

Problem poddany analizie charakteryzuje złożoność czasowa O(n). Wynika to bezpośrednio z wykresu 3.1. Mimo, wydawać by się mogło, skomplikowanego schematu blokowego przedstawionego na rysunku 3.1 zaprojektowano algorytm, który można realnie stosować dla dużych tablic. Największa analizowana tablica składała się z 104857600 zmiennych typu int, więc łącznie zajmowała niecałe 0,5GB. Czas wykonania algorytmu dla takiej tablicy wynosił nieco ponad 4s. Biorąc pod uwagę, iż obliczenia przeprowadzane były na średniej klasy komputerze osobistym, algorytm można uznać za optymalny.

5. Podsumowanie

Cele ćwiczenia zostały zrealizowane.