P O L I T E C H N I K A R Z E S Z O W S K A
im. Ignacego Łukasiewicza
WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

Tomasz Nowak

Grupa P05

Projekt Algorytmy i struktury danych nr. 1

Rzeszów 2020

1.Wstęp

Celem projektu było napisanie algorytmu obsługującego ciąg liczb całkowitych w postaci tablicy. Algorytm ten rozwiązałem w środowisku Python 3.9, ponieważ w nim się czuję najlepiej gdy przychodzi mi rozwiązać problemy algorytmiczne. Istnieją do tego może i lepsze oraz szybsze języki programowania lecz ja postawiłem na swoje doświadczenie w środowisku Python'a.

2. Opis problemu

Dla zadanego ciągu liczb całkowitych (w postaci tablicy) znajdź liczbę wszystkich podciągów malejących (Podciąg musi składać się z przynajmniej dwóch wartości).

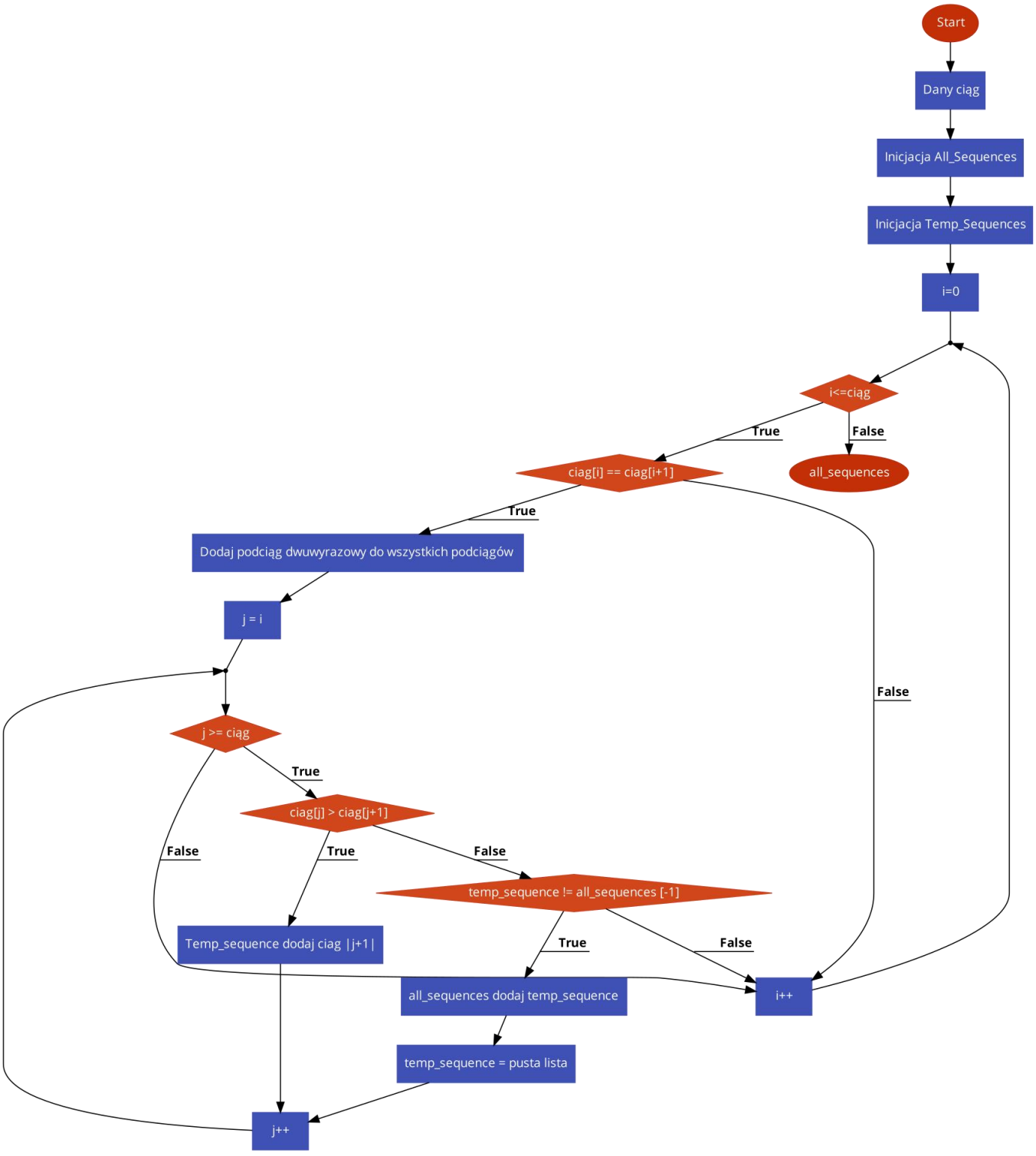
Przykład.
Wejście: A[] = [5, 4, 2, 2, 1]
Wyjście: Liczba wszystkich podciągów malejących to 4.
[5, 4], [5, 4, 2], [4, 2], [2, 1]

Wejście: A[] = [1, 2, 4, 6, 7]
Wyjście: Liczba wszystkich podciągów malejących to 0.

3. Obserwacje

Gdy wchodzimy w pierwszą pętle 'for' w naszej tablicy musimy sprawdzić czy A[P] == A[P+1], jeśli tak to postępujemy wedle założenia i dodajemy do wszystkich podciągów ten podciąg składający się z dwóch porównywanych wyrazów.
Następnie sprawdzamy kolejne wyrazy i jeśli są one prawidłowe względem założeń to dodajemy je do chwilowego pomocniczego podciągu.

4. Schemat blokowy



5. Pseudokod

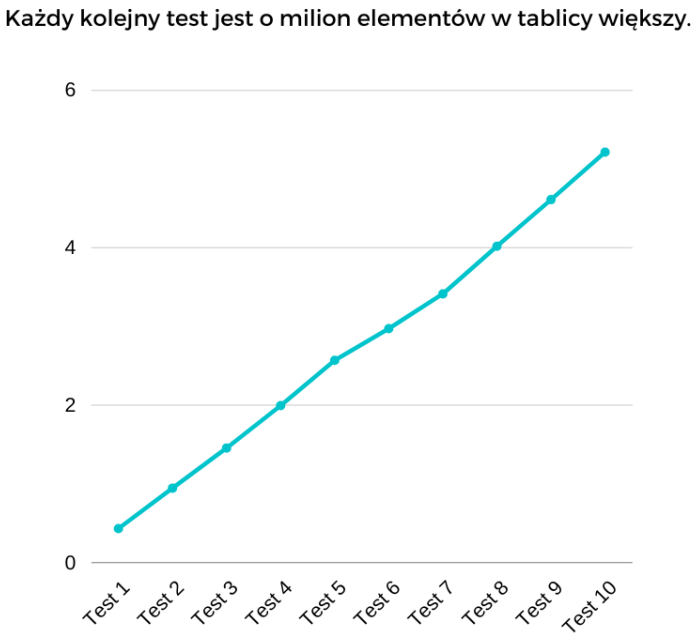
```
Lista = []  -> Wprowadzamy
All_sequences = []
Temp_sequence = []
Length -> długość listy
Dla i = 0 do Length
Jeśli Lista[i] == Lista [i+1]
All_sequences[] Dodaj podciąg Lista[i], Lista[i+1]
Temp_sequence = Lista[i], Lista[i+1]
Dla j = i do Length
                Jeśli Lista[j] > Lista[j + 1]
                    Temp_sequence + Lista[j + 1]
                Jeśli Nie
                    Jeśli Temp_sequence != All_sequences[-1]
                        All_sequences[] += Temp_sequence[]
                    Temp_sequence = 0
                    Zakończ
```

6. Złożoność obliczeniowa

W zawartym kodzie w repozytorium wykonałem funkcję testującą oraz mierzącą czas. Przyjąłem ,że każdy kolejny test będzie się odbywał z milionem więcej elementów w liście a oto rezultat pomiarów:

```
Czas dla 0 elementów w liście: 0.0
Czas dla 1000000 elementów w liście: 0.4352903366088867
Czas dla 2000000 elementów w liście: 0.9492707252502441
Czas dla 3000000 elementów w liście: 1.4587228298187256
Czas dla 4000000 elementów w liście: 1.9974606037139893
Czas dla 5000000 elementów w liście: 2.571315288543701
Czas dla 6000000 elementów w liście: 2.9743423461914062
Czas dla 7000000 elementów w liście: 3.415929079055786
Czas dla 8000000 elementów w liście: 4.019750356674194
Czas dla 9000000 elementów w liście: 4.611892223358154
Czas dla 10000000 elementów w liście: 5.214914798736572
```

A teraz przenieśmy nasze wyniki na wykres liniowy aby zilustrować sobie czas działania naszego algorytmu dla każdego osobnego testu.



Co możemy zauważyć to to, że wykres jest liniowy i z każdym kolejnym milionem elementów czas wykonywania algorytmu wydłuża się o pół sekundy. W związku z czym złożoność obliczeniową możemy opisać jako $O(n)$.

7. Podsumowanie

Mam wielką nadzieję, że rozwiązanie oraz dogłębne zbadanie powyższego problemu przypadnie do gustu, każdemu czytającemu moje sprawozdanie. Jeszcze nigdy nie podchodziłem do wykonanych przez siebie algorytmów w tak wnikliwy sposób czyt.: tworzenie kodu blokowego oraz pseudokodu. Największa dla mnie osobiście trudnością była "współpraca" ze środowiskiem Office. A w szczególności na największy aplauz zasługuje Word a zaraz po nim Excel którego finalnie "odłożyłem na bok".