Sprawozdanie z wczytywania do Stosu / Kolejki

Michał Bizoń

05 IV 2014

1 Założenia i implementacja metod

Naszym zadaniem była implementacja Stosu w formie tablicowej (w dwóch wersjach) oraz kolejki w formie listy. Następnie dla każdej z zaimplementowanych metod przechowywania danych mieliśmy zmierzyć czas przypisywania danych wczytywanych z pliku do stworzonych struktur.

1.1 Stos - wersja powiększająca tablicę o jeden element

Wersja tablicowa - tablica przechowująca Stos miała być powiększana o 1 element za każdym razem, gdy przekroczyła swoją wielkość.

W przypadku, gdy rozmiar tablicy jest początkowo nieznany - ponieważ rozmiar ten jest określony przez plik z danymi wczytanymi do programu - program powiększa tablicę w następujący sposób:

- Wczytuje element do tablicy głównej - jeżeli wartość zmiennej **aktualny_element** jest równa **aktualny_rozmiar**, to uruchomiona zostaje procedura **powiekszOne()** - zwiększająca rozmiar tablicy o 1 element
- W funkcji **powiekszOne()** utworzona zostaje nowa tablica pomocnicza - **tablica_pomocnicza**, której wielkość jest o 1 większa niż **aktualny_rozmiar**.
- Następuje przypisanie elementów z tablicy głównej - **tablica_elementow** do **tablica_pomocnicza** - od elementu 0 do elementu **aktualny_ rozmiar**.
- Zmienna **aktualny_rozmiar** jest następnie zwiększany o 1.
- Adres **tablica_pomocnicza** jest przypisywany do wskaźnika **tablica_elementow**, a następnie kasowana jest tablica pomocnicza
- Po powiększeniu tablicy głównej następuje przypisanie (wczytanie) danej z pliku do tablicy - na pozycję **aktualny_element**
- Zmienna **aktualny_element** jest zwiększana o 1
- Proces jest powtarzany, dopóki nie zostaną wczytane wszystkie elementy do tablicy (dopóki **ilosc_elementow** nie będzie równa ilości elementów pobranych z pliku

1.2 Stos - wersja powiększająca tablicę dwukrotnie

Wersja tablicowa - tablica przechowująca Stos miała być powiększana dwukrotnie za każdym razem, gdy ilość wczytanych elementów zrównała się z aktualną wielkością tablicy.

W przypadku, gdy rozmiar tablicy jest początkowo nieznany - ponieważ rozmiar ten jest określony przez plik z danymi wczytanymi do programu - program powiększa tablicę w następujący sposób:

- Wczytuje element do tablicy głównej - jeżeli wartość zmiennej **aktualny_element** jest równa **aktualny_rozmiar**, to uruchomiona zostaje procedura **powiekszDouble()** - zwiększająca rozmiar tablicy dwukrotnie
- W funkcji **powiekszDouble()** utworzona zostaje nowa tablica pomocnicza - **tablica_pomocnicza**, której wielkość jest dwukrotnie większa niż **aktualny_rozmiar**.
- Następuje przypisanie elementów z tablicy głównej - **tablica_elementow** do **tablica_pomocnicza** - od elementu **0** do elementu **aktualny_rozmiar**.
- Wartość zmiennej **aktualny_rozmiar** jest aktualizowana - dwukrotnie większa, niż poprzednia wartość (aktualny_rozmiar = 2*aktualny_rozmiar).
- Adres **tablica_pomocnicza** jest przypisywany do wskaźnika **tablica_elementow**
- Po powiększeniu / sprawdzeniu rozmiaru tablicy (czy jest jeszcze miejsce na umieszczenie kolejnego elementu) tablicy głównej następuje przypisanie (wczytanie) danej z pliku do tablicy - na pozycję **aktualny_element**
- Zmienna **aktualny_element** jest zwiększana o 1
- Proces jest powtarzany, dopóki nie zostaną wczytane wszystkie elementy do tablicy (dopóki **ilosc_elementow** nie będzie równa ilości elementów pobranych z pliku

1.3 Kolejka - lista jednokierunkowa

Kolejka została zaimplementowana jako lista jednokierunkowa. Każdy element tejże listy posiada zmienną typu **int** zawierającą wartość danego elementu (**wartosc** a także wskaźnik (typu **Element** na kolejny element.

Struktura (klasa) listy zawiera wskaźniki na **korzen** - czyli pierwszy wczytany element, a także **ostatni** - wskazujący na ostatni wczytany element. Posiada także zmienną typu **int** o nazwie **ilosc_elementow** - przechowującą ilość wczytanych już elementów

Wczytywane do programu dane z pliku i przypisywanie ich do listy odbywa się w następujący sposób:

- Inicjowany jest jeden element tymczasowy - typu element, do którego zostaje przypisana dana **wartosc**. Następnie element ten jest wysyłany do funkcji **push_back()** dodającej dany element na koniec istniejącej kolejki.
- Po wczytaniu elementu do funkcji **push_back()** funkcja na początku sprawdza, czy istnieje wskaźnik na ostatni element - jeżeli ma on inną wartość niż ***NULL***, to przypisuje się wskaźnik na dodawany element jako następny po ostatnim, a później ustawia wskaźnik **ostatni** jako dodawany element.

- Jeżeli wysłany do funkcji element jest pierwszym z kolei, automatycznie staje się on **korzen**iem, czyli pierwszym elementem listy. W przypadku, gdy **korzen** juz istnieje, nowo dodany element staje się **ostatni**m.
- Po każdym poprawnie dodanym elemencie zwiększany jest licznik (zmienna textbfilosc_elementow
- Proces jest powtarzany, dopóki nie zostaną wczytane wszystkie elementy (dopóki **ilosc_elementow** nie będzie równa ilości elementów pobranych z pliku

2 Sposób działania programu

Program - w skrócony sposób działa w następujący sposób:

```
Wyświetlenie powitania
  -> Wczytywanie pliku z danymi, dopóki nie zostanie wskazany poprawny plik
Wyświetlenie PARAMETRÓW symulacji i MENU:
  -> ILOSC POWTORZEN: - ile razy wykona sie przypisanie z wczytanym zestawem danych
  -> ROZMIAR TABLICY: - ilość elementów tablicy wczytanej z pliku
  -> Nazwa pliku wczytanego
  -> Nazwa pliku wynikowego - ogólna - modyfikowana przez funkcję zapisującą czasy

Rozpoczęcie Symulacji (Opcja S)
  -> Inicjacja obiektów typu STOS, Lista ; Inicjacja Tabel przechowujących czasy

  -> Wykonaj kolejno dla [x] algorytmów przypisania:
   |--> Dla n = 0 ; dopóki n <= ilości losowań ; zwiększaj n o 1 i wykonuj:
        --> Zapisz aktualny czas do zmiennej timeval &start
        ---> Wykonaj Algorytm Przypisania [x]
        --> Zapisz aktualny czas do zmiennej timeval &stop
        --> Oblicz czas wykonania wszystkich powtórzeń na zasadzie
                TABELA_CZAS_SUMA[n] += OBLICZ_CZAS(start, koniec);
        --> Oblicz czas danego przypisania na zasadzie:
                TABELA_CZASOW[n][x] = OBLICZ_CZAS(start, koniec);
        --> Zniszcz obiekt użyty do przypisania
   |--> Wyświetl czas wszystkich powtórzeń - 9 miejsc po przecinku
   |--> Wyświetl średni czas jednego przypisania - [Czas ogólny] / [ilość sortowań]

  -> Jeżeli Tryb DEBUG włączony - nie zapisuj wyników do plików, inaczej uruchom ZAPIS_C

  -> Oczyść pamięć obiektów typu STOS i Lista, wyzeruj Tabele czasów
  -> Zaczekaj na dowolny znak z klawiatury, wyczyść ekran i zakończ program
```

2.1 Dane wejściowe i wyjściowe

Program wczytuje dane z pliku i zapisuje w poniższy sposób:

Nr.	Dane Wejściowe	Dane Wyjściowe
1	[ilość elementów]	[czas wykonania przypisania nr. 1]
2	[element 1]	[czas wykonania przypisania nr. 2]
3	[element 2]	[czas wykonania przypisania nr. 3]
.	.	.
N	[element N]	[czas wykonania przypisania nr. N]

gdzie ***N*** - dla danych wczytywanych ilość elementów, a dla danych wyjściowych ilość przypisań. Dane wyjściowe są zapisywane do plików następująco:

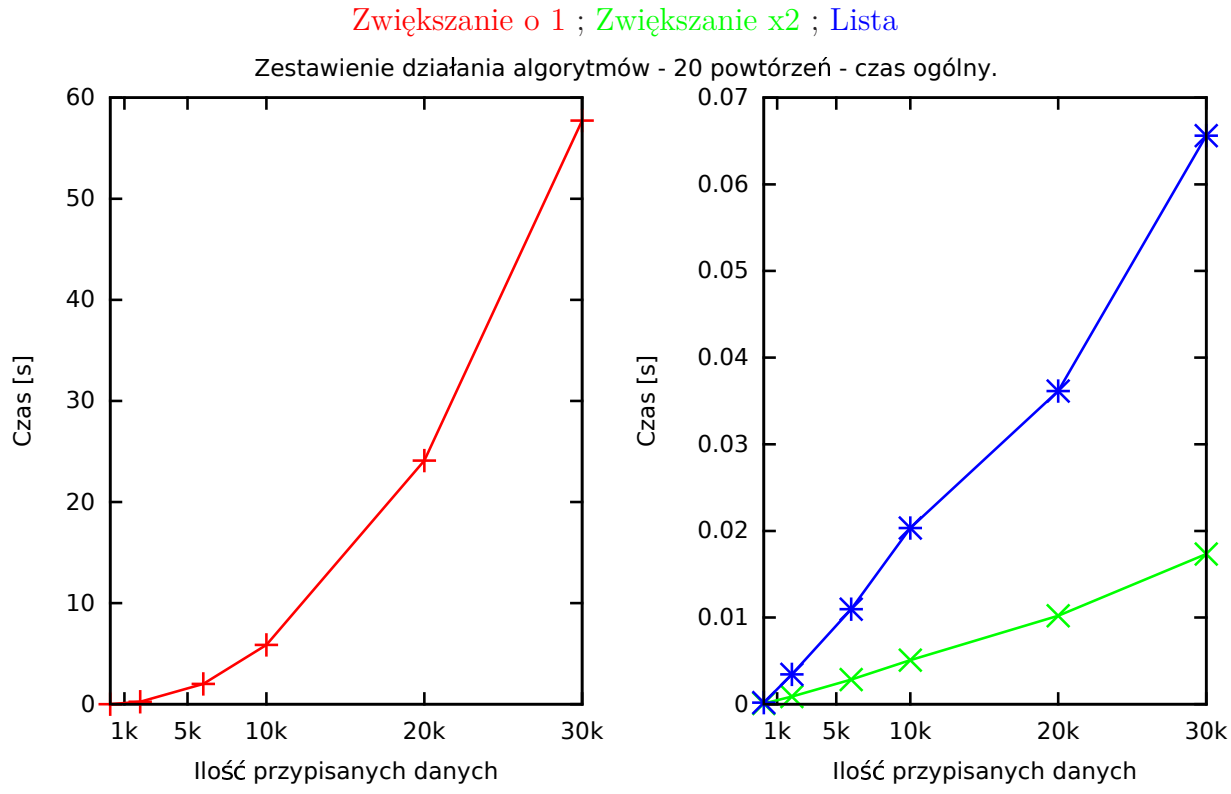
[ilość elementów]_WYNIKI_**[nr. algorytmu]**.csv

3 Wyniki

Wykres porównujący ogólne czasy przypisań tych algorytmów - dla zestawu danych kolejno:

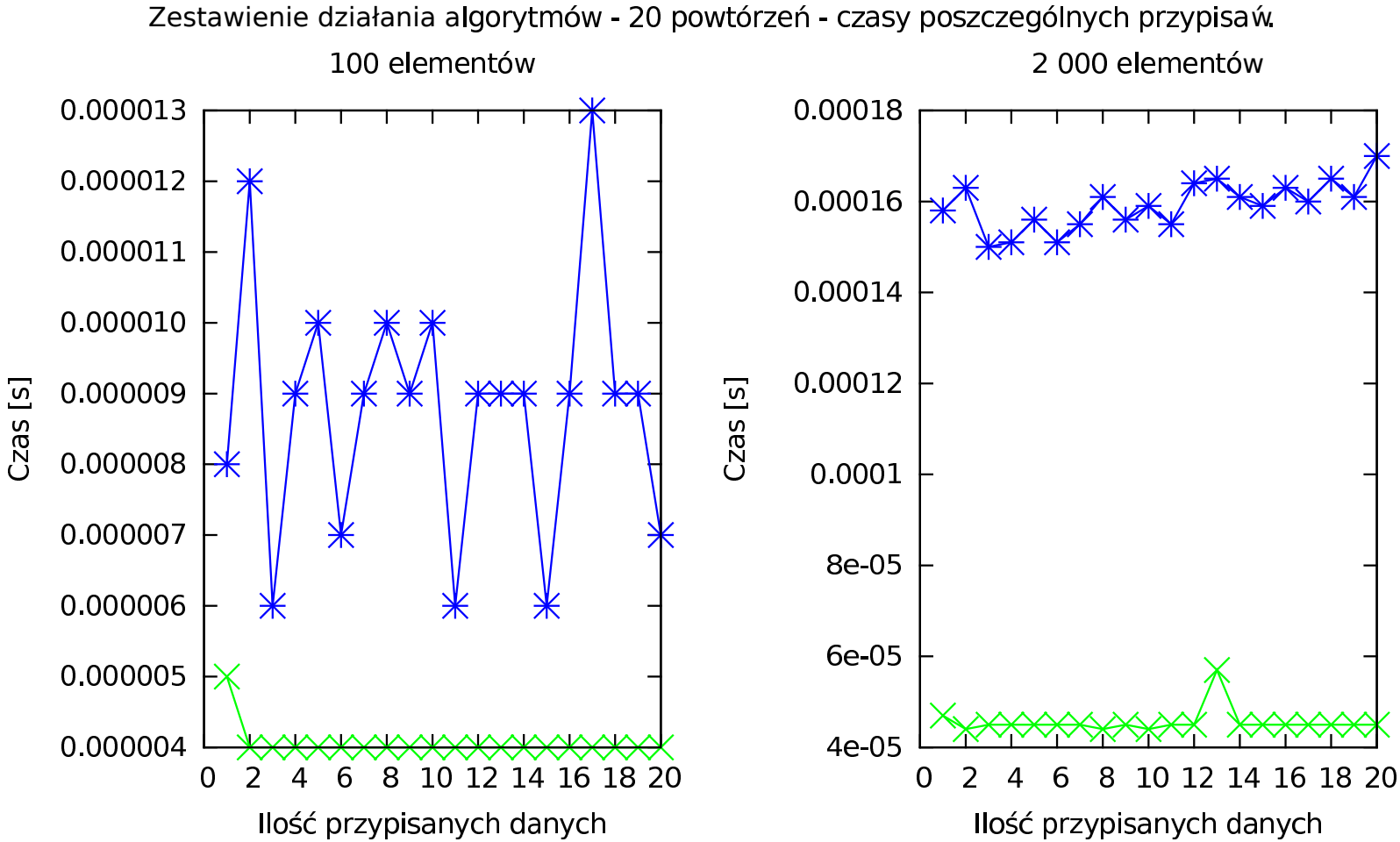
[100 ; 2000 ; 6000 ; 10000 ; 20000 ; 30000]

W związku z dużymi rozbierznościami w otrzymanych czasach, dane zostały rozbite na dwa wykresy

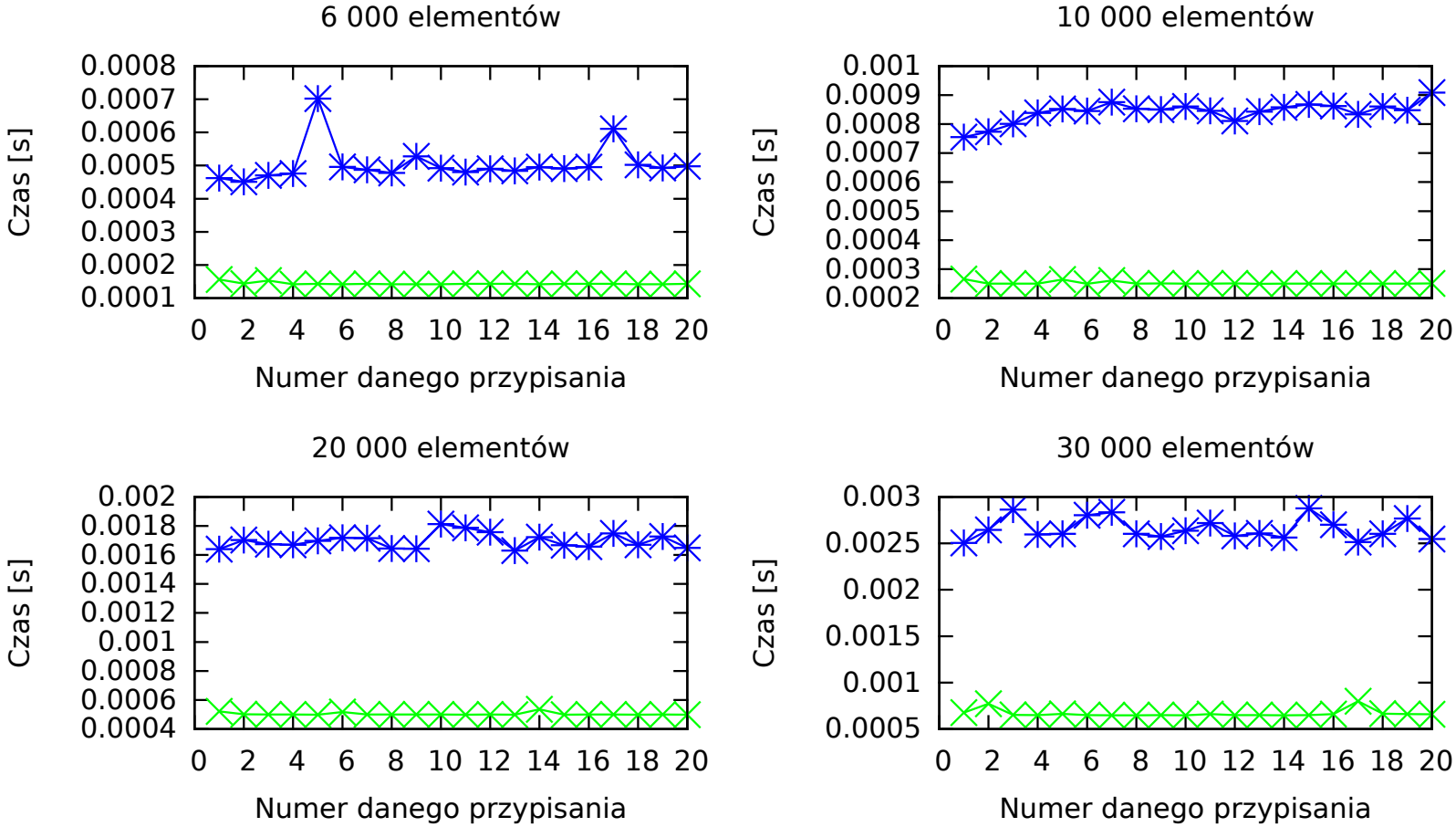


3.1 Tabele z porównaniem poszczególnych przypadków

Z racji bardzo dużych czasów przypisania dla Stosu powiększanego o 1, czasy przypisań zostały uzględnione na poniższych wykresach



Zestawienie działania algorytmów - 20 powtórzeń - czasy poszczególnych przypisań.



4 Wnioski

4.1 Złożoność obliczeniowa

Na podstawie przeprowadzonych pomiarów i sporządzonych wykresów można w przybliżeniu stwierdzić, iż złożoność obliczeniowa:

- Powiększania stosu o 1 element wynosi $\mathbf{O(n^2)}$
- Powiększania stosu x2 a także Listy jest w testowanym zakresie liniowa - wynosi $\mathbf{O(n)}$

Zaimplementowane przeze mnie algorytmy mają zakładaną z definicji złożoność obliczeniową.

4.2 Problemy implementacyjne

Główne problemy, z jakimi się zetknąłem polegały na alokowaniu (i zwalnianiu) pamięci tablicy dynamicznej tworzonej w klasie (jak początkowo sądziłem). Mimo wykorzystania Valgrinda (debuggera) oraz kompilowania programu pod Windowsem **(Visual C++ 2010)** oraz Linuxem **(Qt Creator 5.2.2 + GCC 4.8)** cały czas po poprawnym wykonaniu programu (aż do ostatniej linii symulacji program wykonywał się poprawnie) dostawałem artefakty (losowe liczby, znaki ASCII itp) wypisywane na wyjście - dlatego też po wykonaniu jednej pętli program zakańcza działanie. Prawdopodobnie spowodowane było to brakiem zwolnienia pamięci tablicy, do której wyczytywane są dane z pliku.