Data : 12. 11 . 2021 r.

Politechnika Rzeszowska im. Ignacego Łukasiewicza
Wydział Matematyki i Fizyki Stosowanej
Kierunek Inżynieria i Analiza Danych
Gr. Laboratoryjna nr 4
Nazwa przedmiotu :
Projekt – Algorytmy i struktury Danych

Temat projektu:

Program do znajdowania największej możliwej liczby

Anna Turek

Nr albumu: 169856

1. Realizacja projektu

Program polega na znajdowaniu największej możliwej liczby, skonstruowanej poprzez złączenie zadanych przez użytkownika elementów.

2. Analiza kodu użytego do napisania programu:

2.1 Biblioteki

- #include <iostream>
- #include <string>
- #include <algorithm>
- #include <vector>
- #include <ctime>
- #include <fstream>
- #include <chrono>
- #include <iomanip>

  - ❖ Biblioteka <iostream> posłuży nam do deklaracji cout, cin, endl, back_inserter.
  - ❖ Biblioteka < string > posłuży nam do deklaracji ciągu znaków, czyli stringów
  - ❖ Biblioteka < algorithm > posłuży nam do funkcji sort oraz transform
  - ❖ Biblioteka < vector > posłuży nam do deklaracji tablic dynamicznych, czyli takich do których na bieżąco można dopisywać wartości.
  - ❖ Pozostałe biblioteki posłużą nam do obliczenia złożoności czasowej

2.2 Funkcja int main ()

2.1.1 Deklaracja zmiennych

- int input; // deklaracja zmiennej
- std::vector<int> inputVector;
- std::cout << " Press \n t - test \n c - continue" << std::endl;
- char option;
  std::cin >> option;
- switch (option){
- case 't': {
  int NumberOfTests;
- std::cout << "input number of test" << std::endl;
- std::cin >> NumberOfTests;
- tests (NumberOfTests);
- break;

  - ❖ Deklarujemy zmienną “input” za pomocą int.
  - ❖ Następnie deklarujemy wektora, do którego dodamy nasze wcześniej zadeklarowane zmienne “input” (int)[np. 1,2,3,4] za pomocą vector <int> inputVector. InputVector będzie nazwą tego wektora.
  - ❖ Następnie deklaracje mówią nam o tym, że użytkownik za pomocą zmiennej char (“t” lub “c”) może wybrać czy komputer będzie losował za niego daną liczbę wartości czy sam je wpisze.

2.1.2 Polecenia dla użytkownika

- std::cout << "Fill vector with numbers." << std::endl;
- std::cout << "Write 'x' if you want to end." << std::endl;

  - ❖ W pierszym cout mówimy użytkownikowi programu aby wypełniłl wektor liczbami.
  - ❖ W drugim cout informujemy go, że musi wpisać “x” jeśli chce zakończyć wypisywanie liczb. Może jednak być to dowolna litera jaką wybierze.

2.1.3 Pętla

- while (std::cin >> input) {
- inputVector.push_back(input);

  - ❖ Pętla while mówi nam, że będzie się powtarzać dopóki użytkownik będzie wpisywał wartości liczbowe (input)[int]. Gdy wpisze literę, pętla przerwie swoje działanie.
  - ❖ Funkcja inputVector.push_back(input) zapełnia wartościami wektora dodając każdą nowo wpisaną na koniec.

2.1.4 Przeniesienie logiki

- std::string biggestNumber = GetBiggestNumber(inputVector); //przeniesienie głównej logiki do funkcji GetBiggestNumber()

- std::cout << "Your biggest number is: " + biggestNumber << std::endl;

  - ❖ Przenosimy główną logikę funkcji do funkcji GetBiggestNumber.
  - ❖ Wprowadzamy do konsoli informację o wyniku dla użytwonika oraz sam wynik w postaci biggestNumber.

2.3 Funkcja GetBiggestNumber

2.3.1 Deklaracja zmiennych

- std::string GetBiggestNumber(std::vector<int> inputVector) {
- std::vector<std::string> transformedVector; //deklaracja wektora który później będzie miał te same wartości co inputVector tylko w stringach ["1", "2", "3", "4"]

- ❖ Deklarujemy GetBiggestNumber
- ❖ Następnie deklarujemy wektora, do którego dodamy nasze wcześniej zadeklarowane zmienne “input” (int)[np. 1,2,3,4] za pomocą vector <int> inputVector. InputVector będzie nazwą tego wektora. TransformedVector to nazwa wektora, który będzie miał te same wartości co inputVector, lecz będzie zapisany w funkcji to_string (string)[np. “1”, “2”, ”3”, “4”]. Funkcja vector<string> transformedVector odpowiada za deklarację tego wektora.

2.3.2 Transformacja

- std::transform(
- inputVector.begin(),
- inputVector.end(),
- std::back_inserter(transformedVector),
- [](const int& number) { return std::to_string(number); }

- ❖ Najpierw transformujemy wektor int na wektor string. Jest to konieczne, ponieważ nie chcemy dodawać do siebie liczb, tylko je łączyć tak aby z ich połączenia powstała jak największa liczba.
- ❖ O wprowadzonej przez użytkownika początkowej wartości mówi nam inputVector.begin()
- ❖ Za to o wprowadzonej przez użytkownika końcowej wartości mówi nam inputVector.end()
- ❖ funkcja back_inserter zapełni wartościami wektor transformedVector który będzie naszym wynikiem. Przekazywane elementy idą od końca.
- ❖ Ostatni parametr przekazuje każdy element do ciała funkcji. Dzięki tej funkcji wiemy jak przetransformować wektor. Jest to sposób w jaki go wypełniamy. W tej funkcji przekazujemy jako referencję (oryginalną wartość) każdy parametr z input i konwertujemy go to_string. Przekonwertowany wynik wnosimy na transformedVector

2.3.3 Sortowanie

```
std::sort(
● transformedVector.begin(),
●  transformedVector.end(),
●  [](const std::string& lhs, const std::string& rhs) { return rhs + lhs < lhs + rhs;}
);
```

//lhs, rhs - parametry, które przekazujemy do ciała funkcji, łączymy je na dwa sposoby i porównujemy

- ❖ Pierwszy parametrem jest transformedVector.begin() , który jest początkową wartością inputVector
- ❖ O Wartości końcowej inputVector mówi transformedVector.end()
- ❖ Ostatnia funkcja pobiera dwa parametry, łączy je ze sobą w różnej kolejności i porównuje na zasadzie większy-mniejszy. Następnie ustala w jakiej kolejności posortować dane w wektorze.

Zasada na jakiej działa sortowanie pokazana na przykładzie:

rhs + lhs < lhs + rhs

czyli np:

“34544” < “44354”

z danych: [“12”, “65”,“**44**”, “**345**”, “1”]

Jest to tak zwana metoda “sortowania bąbelkowego”. Sortowanie bąbelkowe polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności jeśli zaburzają one porządek, w jakim sortuje się tablicę.

2.3.4 Uzyskanie największej liczby

- std:: string biggestNumber = GetConcatenatedStringFromVector(transformedVector); //uzyskanie biggestNumber poprzez konkatenację elementów wektora w osobnej fukncji
- return biggestNumber;

  - ❖ Deklarujemy biggestNumber jako GetConcatenatedStringFromVector(transformedVector). Uzyskujemy wynik poprzez złączenie elementów wektora. Nastąpi to w następnej funkcji.
  - ❖ Zwracamy wartość biggestNumber

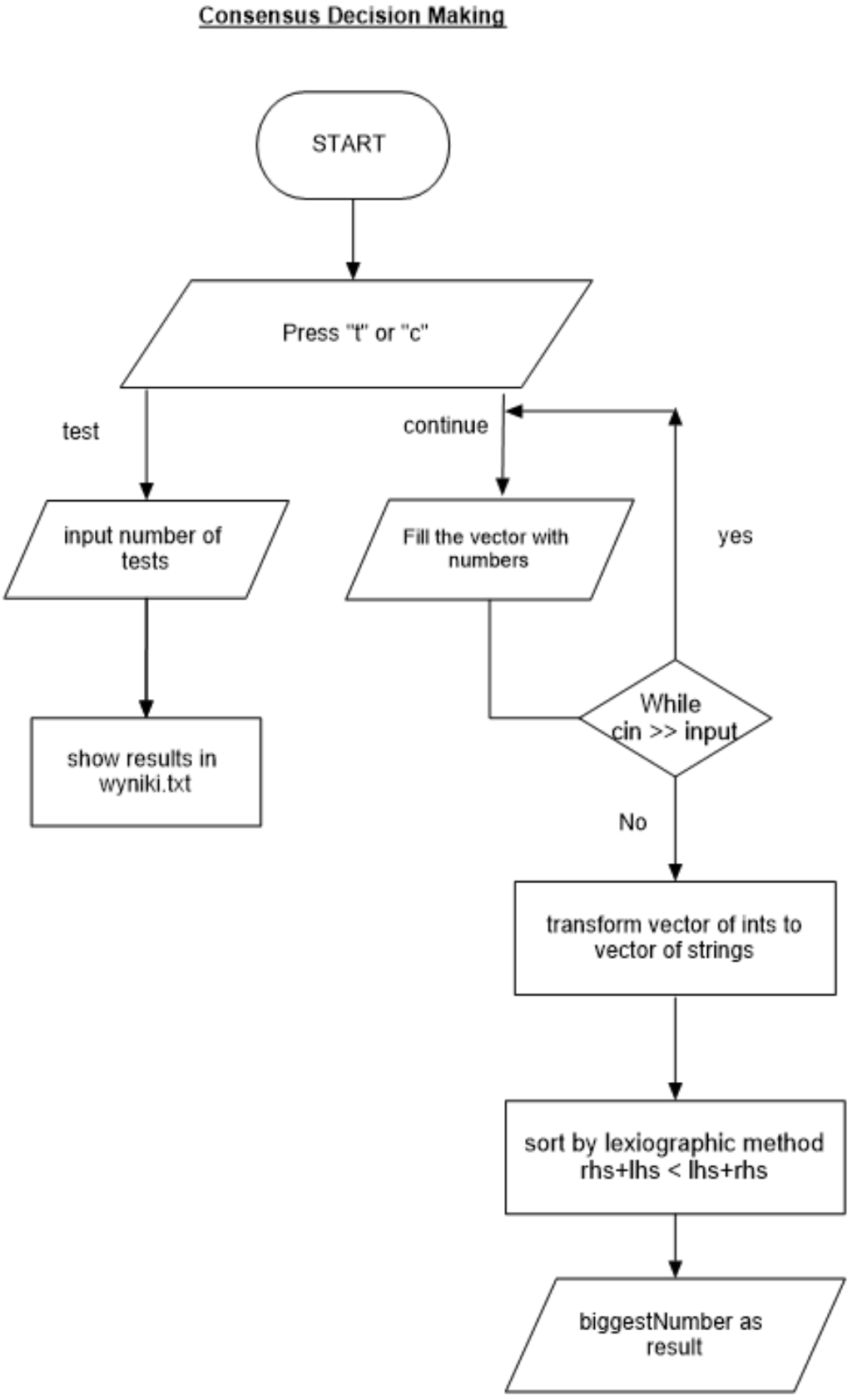
2.4 Funkcja GetConcatenatedStringFromVector

2.4.1 Wybór największej liczby

- std::string concatenatedString = "";
- for (size_t i = 0; i < vector.size(); ++i) {
  concatenatedString += vector[i];
  }
- return concatenatedString;

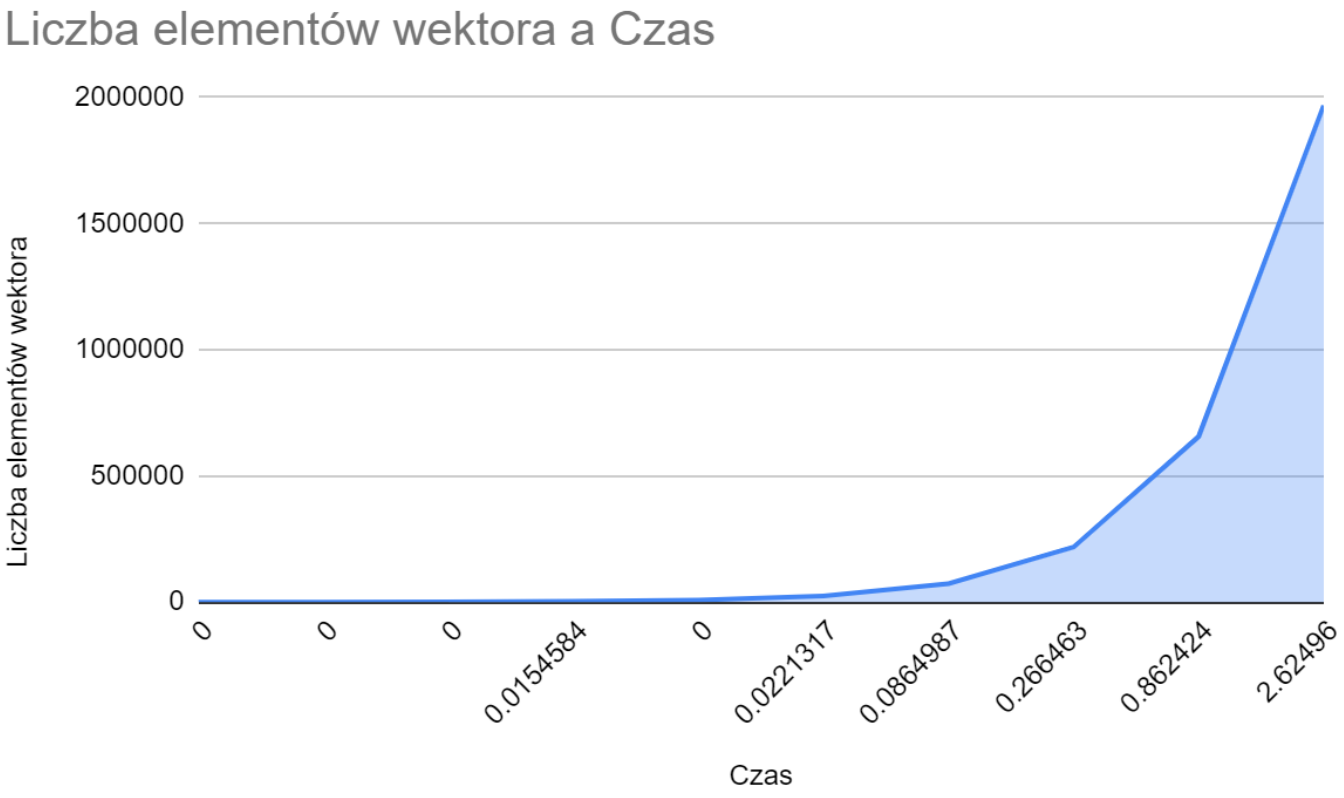
  - ❖ Najpierw deklarujemy pustą funkcję to_string.
  - ❖ Następnie w pętli “for” iterujemy od 0 do tylu razy, ile wynosi wielkość wektora vector. Następnie łączymy wszystkie wartości wektora aby nie mieć wyniku wypisanego z przerwami, tylko jako ciąg znaków. Ostateczny wynik będziemy mieć w funkcji concatenatedString += vector[i]. Jest to to samo co: concatenatedString = concatenatedString + vector[i].
  - ❖ Zwracamy wartość concatenatedString

3. Schemat blokowy algorytmu



4. Złożoność czasowa na podstawie funkcji testującej

- Na wykresie widzimy tendencję rosnącą. Po wpisaniu liczby elementów wektora komputer oblicza dla nich czas a potem czas dla ich trzykrotności. Początek jest przy 100 elementach, a kończymy na 1968300.



5. Pseudokod

```
    wczytaj (input, option)
        w przypadku "t"
            wczytaj liczbę testów
        default
            dopóki (fill vector with numbers)
                break
        transformuj int for string
    sortuj transformedVectors
string biggestNumber
```

6. Wnioski

- Ułożenie kodu przy pomocy funkcji jest bardzo praktyczne. Płynne przejście z deklaracji, przez transformację i sortowanie bardzo pomaga w znalezieniu największej możliwej liczby oraz w osiągnięciu przejrzystości kodu. Złożoność czasowa jest pokazuje że im większą liczbę elementów w wektorze komputer ma obliczyć, tym więcej czasu potrzebuje on na to działanie. Widać to dopiero przy naprawdę dużych liczbach, których użytkownik nie jest w stanie prawdopodobnie sam wpisać.