Politechnika Wrocławska

Raport
Mikroprojekt 1
Projektowanie i Analiza Algorytmów

2 kwietnia 2023

Filip Mirski nr indeksu 264267

1 Zadanie 1

Zadanie pierwsze polegało na zaimplementowaniu dynamicznej tablicy dwuwymiarowej, a następnie napisać na podstawie tego trzy funkcję: wypełnienie tej tablicy losowymi liczbami od 0 do liczby podanej przez użytkownika, wyświetlić tablicę oraz znaleźć jej maksymalny element. To wszystko należało opakować w menu konsolowe. Zadanie zostało wykonane obiektowo.

```
Zad1::Zad1(int n, int m)
{
    columnSize = m;
    rowSize = n;

    tab = new int* [columnSize];
    for (int i = 0; i < columnSize; i++)
    {
        tab[i] = new int[rowSize];
    }
}
```

Rysunek 1: Inicjalizacja dwuwymiarowej dynamicznej tablicy

1.1 Podpunkt A

Wypełnienie dynamicznej tablicy dwuwymiarowej ma złożoność obliczeniową O($n * m$) - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową O($n * m$).

```
void Zad1::numberGenerator(int rangeSize)
{
    srand(time(0));

    for (int i = 0; i < columnSize; i++)
    {
        for (int j = 0; j < rowSize; j++)
        {
            tab[i][j] = rand() % (rangeSize + 1);
        }
    }
}
```

Rysunek 2: Funckja odpowiadająca za uzupełnianie tablicy

Funkcja "rand()" losuje liczbe z zakresu od 0 do wybranego przez użytkownika zakresu, a funkcja srand zmienia ziarno losowania za każdym uruchomieniem programu.

1.2 Podpunkt B

Wyświetlenie dynamicznej tablicy dwuwymiarowej ma złożoność obliczeniową O($n * m$) - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową O($n * m$).

```
void Zad1::displayTab()
{
    for (int i = 0; i < columnSize; i++)
    {
        for (int j = 0; j < rowSize; j++)
        {
            cout << tab[i][j] << " ";
        }
        cout << endl;
    }
}
```

Rysunek 3: Funkcja odpowiadająca za wyświetlenie tablicy

1.3 Podpunkt C

Wyszukiwanie wartości maksymalnej w dynamicznej tablicy dwuwymiarowej ma złożoność obliczeniową O($n*m$) - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową O($n * m$).

```
void Zad1::maxValue()
{
    int max = 0;
    for (int i = 0; i < columnSize; i++)
    {
        for (int j = 0; j < rowSize; j++)
        {
            if (max < tab[i][j])
            {
                max = tab[i][j];
            }
        }
    }
    cout << "Najwiszka liczba w tablicy to: " << max << endl;
}
```

Rysunek 4: Funkcja odpowiadająca za wyszukiwanie wartości maksymalnej w tablicy

1.4 Wynik

```
Podaj wymiary tablicy (n x m):
5
4

--- TABLICA ---
1. Wygeneruj liczby z zakresu:
2. Wyswietl tablice
3. Znajdz najwieksza wartosc
4. Wczytaj liczby z pliku tekstowego
5. Zapisz tablice w pliku tekstowego
6. Wczytaj liczby z pliku binarnego
7. Zapisz tablice w pliku binarnym
8. Wyjscie
Podaj numer opcji:
1
Podaj zakres liczb do wylosowania:
81

--- TABLICA ---
1. Wygeneruj liczby z zakresu:
2. Wyswietl tablice
3. Znajdz najwieksza wartosc
4. Wczytaj liczby z pliku tekstowego
5. Zapisz tablice w pliku tekstowego
6. Wczytaj liczby z pliku binarnego
7. Zapisz tablice w pliku binarnym
8. Wyjscie
Podaj numer opcji:
2
61 35 67 16 43
34 4 26 17 37
60 81 13 30 43
40 31 12 13 24

--- TABLICA ---
1. Wygeneruj liczby z zakresu:
2. Wyswietl tablice
3. Znajdz najwieksza wartosc
4. Wczytaj liczby z pliku tekstowego
5. Zapisz tablice w pliku tekstowego
6. Wczytaj liczby z pliku binarnego
7. Zapisz tablice w pliku binarnym
8. Wyjscie
Podaj numer opcji:
3
Najwiszka liczba w tablicy to: 81
```

Rysunek 5: Rezultat wykonania zadania pierwszego

2 Zadanie 2

W zadaniu drugim należało zczytać i wpisać wartości pliku tekstowego i binarnego, program został wykonany wykorzystując wcześniejsze zadanie. Wszystko również zostało "opakowane" w menu konsolowe.

2.1 Podpunkt A

Zapisywanie dynamicznej tablicy dwuwymiarowej w pliku tekstowym ma złożoność obliczeniową $O(n * m)$ - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową $O(n * m)$. Do wykonania zadania została

```
void Zad1::loadToTextFile()
{
    ofstream file;
    file.open("array.txt");

    if (file.is_open()) {
        int num = 0;
        for (int i = 0; i < this->columnSize; i++) {
            for (int j = 0; j < this->rowSize; j++) {
                num = tab[i][j];
                file << num << " ";
            }
            file << "\n";
        }
    }

    file.close();
}
```

Rysunek 6: Funkcja odpowiadająca za zapisanie tablicy w pliku tekstowym

wykorzystana biblioteka "fstream", ofstream - pokazuje, że zmienna file będzie działała na wyjście do pliku tekstowego, metody "open" i "close" odpowiednio otweirają i zamykają plik - konieczne do funkcjonowania programu. Na koniec zostało zmienione standardowe wyjście console out na file co powoduje przekazanie danych do pliku tekstowego.

2.2 Podpunkt B

Wczytywanie dynamicznej tablicy dwuwymiarowej z pliku tekstowym ma złożoność obliczeniową $O(n * m)$ - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową $O(n * m)$. Do wykonania zadania została

```
void Zad1::loadFromTextFile()
{
    string line;
    ifstream file;
    file.open("data.txt");

    if (file.is_open()) {
        for (int i = 0; i < this->columnSize; i++) {
            for (int j = 0; j < this->rowSize; j++) {
                file >> tab[i][j];
            }
        }
    }

    file.close();
}
```

Rysunek 7: Funkcja odpowiadająca za wczytywanie tablicy z pliku tekstowym

wykorzystana biblioteka "fstream", ifstream - pokazuje, że zmienna file będzie działała na wejściu do pliku tekstowego, metody "open" i "close" odpowiednio otweirają i zamykają plik - konieczne do funkcjonowania programu. Na koniec zostało zmienione standardowe wyjście console in na file co powoduje przekazanie danych z pliku tekstowego do programu.

2.3 Podpunkt C

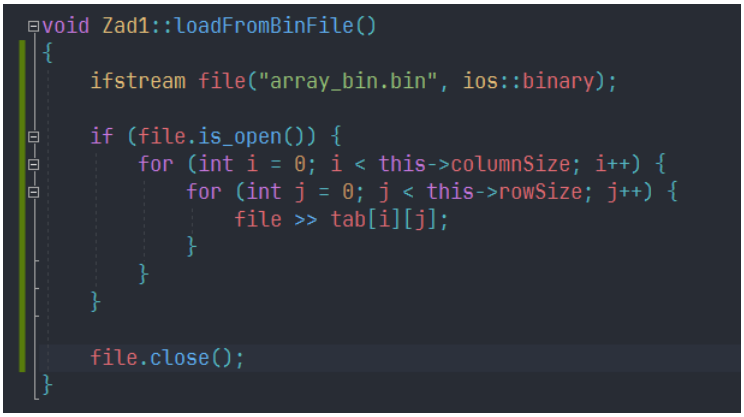
Zapisywanie dynamicznej tablicy dwuwymiarowej w pliku binarnym ma złożoność obliczeniową O($n * m$) - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową O($n * m$). Do wykonania zadania została

Rysunek 8: Funkcja odpowiadająca za zapisanie tablicy w pliku tekstowym

wykorzystana biblioteka "fstream", ofstream - pokazuje, że zmienna file będzie działała na wyjście do pliku tekstowego, dodatkowo jest podany argument, który mówi o czytaniu pliku w rozszerzeniu "bin". Metody "open" i "close" odpowiednio otweirają i zamykają plik - konieczne do funkcjonowania programu. Na koniec zostało zmienione standardowe wyjście console out na file co powoduje przekazanie danych do pliku tekstowego.

2.4 Podpunkt D

Wczytywanie dynamicznej tablicy dwuwymiarowej z pliku binarnego ma złożoność obliczeniową O($n * m$) - gdzie "n" to liczba kolumn i "m" to liczba wierszy, a pamięciową O($n * m$). Do wykonania zadania została



Rysunek 9: Funkcja odpowiadająca za wczytywanie tablicy z pliku tekstowym

wykorzystana biblioteka "fstream", ifstream - pokazuje, że zmienna file będzie działała na wejściu do pliku tekstowego, dodatkowo jest podany argument, który mówi o czytaniu pliku w rozszerzeniu "bin". Metody "open" i "close" odpowiednio otweirają i zamykają plik - konieczne do funkcjonowania programu. Na koniec zostało zmienione standardowe wyjście console in na file co powoduje przekazanie danych z pliku tekstowego do programu.

3 Zadanie 3

Celem tego zadania było napisanie kodu, który ma dwie funkcje rekurencyjne służące do obliczania silni i potęgi. To zadanie również zostało wykonane wykorzystując obiektowe paradygmaty programowania.

3.1 Podpunkt A

Rekurencyjne wykonanie potęgi ma złożoność obliczeniową O(n), a pamięciową O(n) - gdzie "n" to wartość argumentu "exp".

```
int Zad3::power(int base, int exp)
{
    if (exp > 0)
        return base * power(base, exp - 1);
    else
        return 1;
}
```

Rysunek 10: Funkcja odpowiedzialna za obliczenie potęgi

3.2 Podpunkt B

Rekurencyjne wykonanie silni ma złożoność obliczeniową O(n), a pamięciową O(n) - gdzie "n" to wartość argumentu "factor".

```
int Zad3::factorial(int factor)
{
    if (factor == 0 || factor == 1)
        return 1;
    else
        return factor * factorial(factor - 1);
}
```

Rysunek 11: Funkcja odpowiedzialna za obliczenie silni

4 Zadanie 4

W zadaniu czwartym należało napisać funkcję rekurencyjną sprawdzającą czy string jest palindromem. Złożo-

```
bool Zad4::isPal(std::string str, int begin, int end)
{
    if(begin >= end) // większy w momencie gdy wyraz je
        return true;

    if (str[begin] != str[end])
        return false;

    return isPal(str, begin + 1, end - 1);
}
```

Rysunek 12: Funkcja odpowiedzialna za sprawdzanie czy string jest palindromem

ność pamięciowa i obliczeniowa powyższej funkcji to odpowiednio O(n) i O(n) - gdzie n to długość stringa.

5 Zadanie 5

To zadanie jest rozszrzeniem zadania czwartego, bo należało stworzyć funkcję rekurencyjną generującą permutacje zadanego stringa i w tej funkcji mamy sprawdzać, które permutacje są palindromami i zapisujemy jest w tablicy, następnie należało usunąć duplikaty. Do tego celu została wykorzystana tablica statyczna o rozmiarze n!, gdzie n to dlugość stringa.

```cpp
void Zad4::genPerm(std::string prefix, std::string rest)
{
    if (!rest.length()) {
        if (isPal(prefix, 0, prefix.size() - 1)) {
            tab[place] = prefix;
            place++;
        }
        return;
    }

    for (int i = 0; i < rest.size(); i++) {
        std::string newPrefix = prefix + rest[i];
        std::string newRest = rest.substr(0, i) + rest.substr(i + 1);
        genPerm(newPrefix, newRest);
    }
}
```

Rysunek 13: Funkcja generująca palindromy z permutacji

Złożoność pamięciowa i obliczeniowa powyższej funkcji to odpowiednio O(n!) i O(n!).

```cpp
void Zad4::deleteDup()
{
    for (int i = 0; i < 719; i++) {
        for (int j = i + 1; j < 720; j++) {
            if (tab[i] == tab[j]) {
                tab[j] = "";
            }
        }
    }
}
```

Rysunek 14: Funkcja usuwająca duplikaty

Złożoność pamięciowa tej funkcji to O(n^2), a obliczeniowa to O(n^2).

6 Zadanie 8

Zadanie ósme polegało na napisaniu obiektowo dwóch struktur danych: listy jednokierunkowej i kolejki, a następnie dodanie do nich metod: dodawanie/usuwanie elementy, wyświetlanie struktury i usunięcie wszystkich elementów ze struktury.

6.1 Lista

6.1.1 Dodawanie elementu

Dodawanie elementu do listy ma złożoność obliczeniową i pamięciową równą O(1).

```
void List::addElement(int data)
{
    Node* newNode = new Node(data); //
    if (head == nullptr) {
        head = newNode;
        // jeżeli jest to pierwszy node
    }
    else {
        newNode->next = head;
        head = newNode;
    }
}
```

Rysunek 15: Dodawanie elementu do listy

6.1.2 Usuwanie elementu

Usuwanie elementu z listy ma złożoność obliczeniową i pamięciową równą O(n).

```
void List::removeElement(int data)
{
    // jeżeli lista jest pusta
    if (head == nullptr) {
        std::cout << "Lista jest pusta\n";
    }
    // jeżeli element, który chcemy usunąć jest pierwszy
    if (head->data == data) {
        head = head->next;
    }
    Node* toDelete = head;
    // iterowanie po całej liście w poszukiwaniu element
    while (toDelete->next != nullptr) {
        // jeżeli wartość z noda następnego od tego, w k
          next naszego aktualnego elementy na ptr ptra n
        if (toDelete->next->data == data) {
            toDelete->next = toDelete->next->next;
        }
        // jak nie spełni warunktu to przekaż wartość na
        toDelete = toDelete->next;
    }
}
```

Rysunek 16: Usuwanie elementu z listy

6.1.3 Wyświetlanie listy

Wyświetlanie listy ma złożoność obliczeniową i pamięciową równą O(n).

```
void List::displayList()
{
    Node* show = head; // przypisanie do zmienn
    if (head == nullptr)
        std::cout << "Lista jest pusta!\n";
    while (show != nullptr) {
        std::cout << show->data << " "; // poka
        show = show->next; // zmienienie wskaźn
    }
    std::cout << std::endl;
}
```

Rysunek 17: Wyświetlanie listy

6.1.4 Usuwanie wszystkich elementów

Usuwanie wszystkich elementów z listy ma złożoność obliczeniową i pamięciową równą O(n).

```
void List::removeAll()
{
    while (head != nullptr) {
        Node* toDelete = head; // pr
        head = head->next; // przeka
        delete toDelete; // usunieci
    }
}
```

Rysunek 18: Usuwanie wszystkich elementów z listy

6.2 Kolejka

6.2.1 Dodawanie elementu

Dodawanie elementu do kolejki ma złożoność obliczeniową i pamięciową równą O(1).

```
void QueDs::add(int data)
{
    Node* newNode = new Node(data);
    // jeżeli kolejka jest pusta to tail i
    if (tail == nullptr) {
        tail = newNode;
        head = newNode;
    }
    else { // jeśli już istnieją jakieś mi
      był tail
        tail->next = newNode;
        tail = newNode;
    }

}
```

Rysunek 19: Dodawanie elementu do kolejki

6.2.2 Usuwanie elementu

Usuwanie elementu z kolejki ma złożoność obliczeniową i pamięciową równą O(1).

```
// usuwanie elementu, który pi
void QueDs::remove()
{
    Node* toDelete = head;
    head = head->next;
    if (head == nullptr) {
        tail = nullptr;
    }
    delete toDelete;
}
```

Rysunek 20: Usuwanie elementu z kolejki

6.2.3 Wyświetlanie kolejki

Wyświetlanie kolejki ma złożoność obliczeniową i pamięciową równą O(n).

```
void QueDs::display()
{
    Node* show = head;
    if (head == nullptr)
        std::cout << "Kolejka jest pusta!\n";
    while (show != nullptr) {
        std::cout << show->data << " ";
        show = show->next;
    }
    std::cout << std::endl;
}
```

Rysunek 21: Wyświetlanie listy

6.2.4 Usuwanie wszystkich elementów

Usuwanie wszystkich elementów z kolejki ma złożoność obliczeniową i pamięciową równą O(n).

```
void QueDs::removeAll()
{
    Node* toDelete = head;
    while (head != nullptr) {
        Node* toDelete = head;
        head = head->next;
        delete toDelete;
    }
    tail = nullptr;
}
```

Rysunek 22: Usuwanie wszystkich elementów z kolejki