Politechnika Wrocławska

Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Projekt nr 1

Wariant: Zadanie na ocenę bdb (5.0)

Autor:	Aleksander Łyskawa 275462
Wydział i kierunek studiów:	W12N, Automatyka i Robotyka
Termin zajęć:	pon 15:15-16:55
Prowadzący:	dr inż. Witold Paluszyński
Data:	06.04.2024

Spis treści

1 Treść polecenia

Załóżmy, że Jan chce wysłać przez Internet wiadomość W do Anny. Z różnych powodów musi podzielić ją na n pakietów. Każdemu pakietowi nadaje kolejne numery i wysyła przez sieć. Komputer Anny po otrzy- maniu przesłanych pakietów musi poskładać je w całą wiadomość, ponieważ mogą one przychodzić w losowej kolejności. Państwa zadaniem jest zaprojektowanie i zaimplementowanie odpowiedniego rozwiązania radzą- cego sobie z tym problemem. Należy wybrać i zaimplementować zgodnie z danym dla wybranej struktury ADT oraz przeanalizować czas działania - złożoność obliczeniową proponowanego rozwiązania.

2 Opis wybranej struktury

Jako strukturę danych do rozwiązania zadanego problemu, wybrałem kolejkę priorytetową. Kolejkę tę oparłem na liście składającej się z węzłów, które reprezentują poszczególne elementy kolejki. Każdy węzeł zawiera informacje w postaci tekstu (typu std::string) oraz numer sekwencyjny (seqNum), który służy do określenia priorytetu danego elementu.
Struktura ta zawiera wskaźniki na głowę (head) i ogon (tail) kolejki, oraz zmienną przechowującą rozmiar kolejki (size). Klasa ta posiada metody umożliwiające dodawanie (insert), usuwanie minimalnego elementu (removeMin), oraz pobieranie rozmiaru kolejki (getSize). Dzięki tym metodom możliwe jest operowanie na kolejce z zachowaniem priorytetu elementów na podstawie ich numerów sekwencyjnych. Takie podejście pozwala na skuteczne odbieranie, przechowywanie, a następnie wyświetlanie pakietów z zachowaniem właściwej kolejności.

3 Implementacja wybranej struktury

3.1 Struktura Node

Struktura Node reprezentuje pojedynczy węzeł w kolejce priorytetowej. Składa się z następujących pól:

- std::string data - pole przechowujące wartość elementu kolejki
- int seqNum - pole przechowujące numer sekwencyjny elementu, który służy do określenia jego priorytetu w kolejce
- Node* prev - wskaźnik na poprzedni węzeł w kolejce
- Node* next - wskaźnik na następny węzeł w kolejce

3.2 Klasa Receiver

3.2.1 Zmienne prywatne

- Node* head - wskaźnik na pierwszy element kolejki
- Node* tail - wskaźnik na ostatni element kolejki
- int size - aktualny rozmiar kolejki

3.2.2 Metody prywatne

- void insert(const std::string& data, int seqNum) - metoda umożliwiająca wstawienie nowego elementu do kolejki zgodnie z jego numerem sekwencyjnym
- const std::string removeMin() - metoda prywatna usuwająca minimalny element z kolejki i zwracająca jego wartość
- int getSize() const - metoda zwracająca aktualny rozmiar kolejki

3.2.3 Metody publiczne

- receiver receive_message(const std::string filename, int packet_size, int message_size, int offset) - metoda publiczna umożliwiająca odbiór wiadomości z określonego pliku. Przyjmuje argumenty takie jak nazwa pliku, rozmiar paczki, rozmiar wiadomości i przesunięcie
- void display_received_message(receiver receivedPackets) - metoda wyświetlająca odebrane paczki. Przyjmuje jako argument obiekt typu receiver, który przechowuje odebrane paczki, składa je i wyświetla odebraną wiadomość

```
struct Node
{
    std::string data;
    int seqNum;
    Node* prev;
    Node* next;
    Node() {}
    Node(const std::string& data, const int seqNum, Node* prev, Node* next
   );
};

class receiver
{
private:
    Node* head;
    Node* tail;
    int size;
    void insert(const std::string& data, int seqNum);
    const std::string removeMin();
    int getSize() const;
    receiver();
public:
    receiver receive_message(const std::string filename,  int packet_size,
    int message_size, int offset);
    void display_received_message(receiver receivedPackets);
};
```

4 Rozwiązanie problemu

4.1 Argumenty programu

Program został skompilowany do pliku wykonywalnego następującymi komendami:
g++ receiver.cpp -c
g++ main.cpp receiver.o -o driver_zad3

Argumenty wywołania programu mają postać: driver filename offset message_size packet_size.
Oznaczją kolejno:

- nazwa pliku, z którego sczytana będzie wiadomość
- przesunięcie, które definiuje od którego wyrazu wiadomość będzię odbierana
- rozmiar wiadomości, który w przypadku mojej implementacji oznacza ile wyrazów zostanie odebranych. Jeśli rozmiar wiadomości będzie większy od rozmiaru pliku, odebrany zostanie cały plik tekstowy
- rozmiar pakietów, który definiuje liczbę wyrazów w każdym pakiecie. Pakiet nie może być większy niż rozmiar wiadomości

4.2 Wywołanie metody receive_packets

Po wywołaniu programu z odpowiednimi argumentami nastepuje wywołanie funckji receive_packets, do której przekazywane są argumenty wywołania programu. Funkcja zgodnie z zadanymi parametrami dzieli wiadomość na pakiety i zapisuje je w zaimplementownej strukturze, jaką w przypadku mojej implementacji jest kolejka priorytowa.

4.3 Złożenie i wyświetlenie wiadomości

Na koniec, za pomocą metody display_received_message, odebrane pakiety są składane w oryginalną wiadomość oraz wyświetlane w konsoli.

```
1  void receiver::display_received_message(receiver receivedPackets)
2  {
3      std::string receivedMessage;
4      for (int i = 1; receivedPackets.getSize(); i++)
5      {
6          receivedMessage = receivedMessage + receivedPackets.removeMin
       () + " ";
7      }
8      std::cout << std::endl;
9      std::cout << "Received message: " << std::endl;
10     std::cout << receivedMessage << std::endl;
11 }
```

5 Analiza złożoności obliczeniowej

Do przeprowadzenia analizy złożoności obliczeniowej wykorzytstałem napisany z pomocą ChatGPT program runtimes.cpp. Program w pętli wywołuje oryginalny program z argumentami, gdzie offset w kązdym wywołaniu jest ustawiony na 0, rozmiar pakietu równa się 1, natomiast rozmiar wiadomości przy kązdym kojelnym uruchomieniu programu zwiększany jest o 1000. Zatem program wywołuje kolejno:
./driver_zad3 message.txt 0 1000 1
./driver_zad3 message.txt 0 2000 1
...
./driver_zad3 message.txt 0 100000 1
Każde wywołanie programu ze zwiększonym argumentem mierzy czas działania programu i zapisuje wyniki do pliku tekstowego.

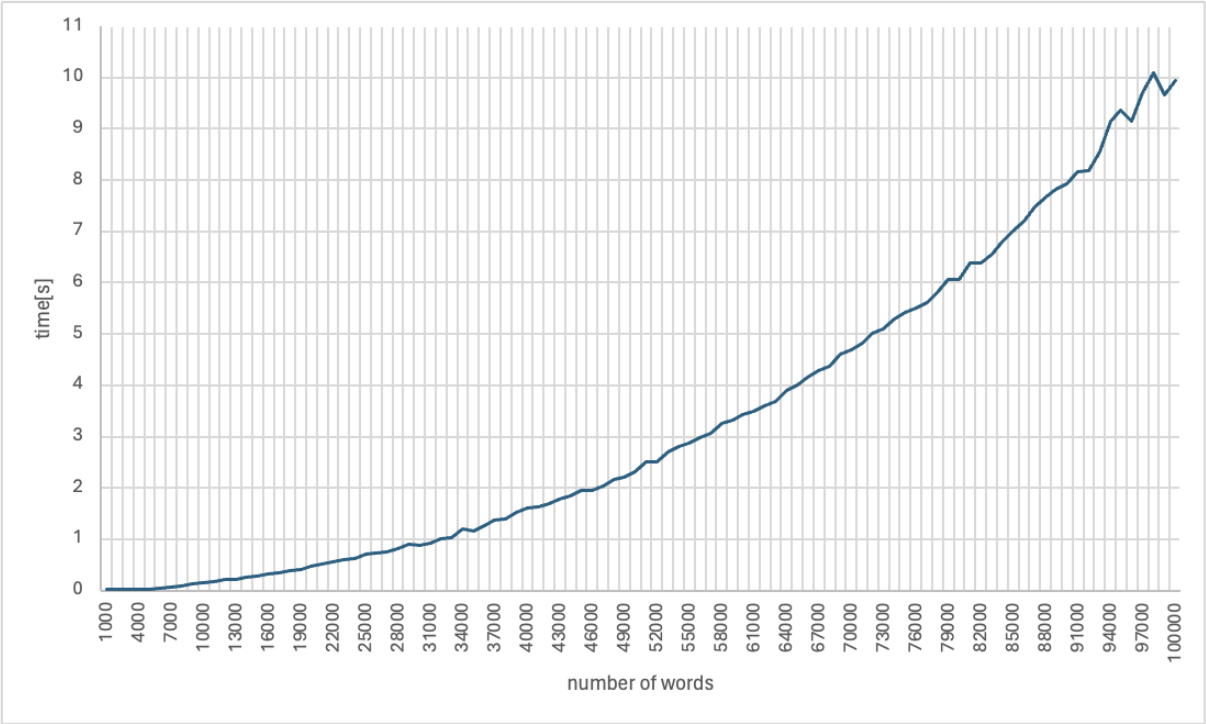
l.p	liczba słów	czas [s]
1	1000	0,018
2	2000	0,017
3	3000	0,022
4	4000	0,027
5	5000	0,033
6	6000	0,045
7	7000	0,066
8	8000	0,096
9	9000	0,125
10	10000	0,155
11	11000	0,177
12	12000	0,207
13	13000	0,227
14	14000	0,262
15	15000	0,270
16	16000	0,317
17	17000	0,353
18	18000	0,384
19	19000	0,401
20	20000	0,468
21	21000	0,510
22	22000	0,553
23	23000	0,600
24	24000	0,633
25	25000	0,708
26	26000	0,722
27	27000	0,749
28	28000	0,826
29	29000	0,904
30	30000	0,875
31	31000	0,916
32	32000	1,019
33	33000	1,040
34	34000	1,197

l.p	liczba słów	czas [s]
35	35000	1,166
36	36000	1,262
37	37000	1,365
38	38000	1,385
39	39000	1,522
40	40000	1,612
41	41000	1,624
42	42000	1,691
43	43000	1,779
44	44000	1,841
45	45000	1,953
46	46000	1,943
47	47000	2,045
48	48000	2,159
49	49000	2,211
50	50000	2,325
51	51000	2,513
52	52000	2,503
53	53000	2,693
54	54000	2,818
55	55000	2,867
56	56000	2,976
57	57000	3,074
58	58000	3,253
59	59000	3,330
60	60000	3,426
61	61000	3,489
62	62000	3,603
63	63000	3,681
64	64000	3,892
65	65000	4,016
66	66000	4,165
67	67000	4,283
68	68000	4,381

l.p	liczba słów	czas [s]
69	69000	4,604
70	70000	4,700
71	71000	4,830
72	72000	5,007
73	73000	5,108
74	74000	5,299
75	75000	5,414
76	76000	5,504
77	77000	5,606
78	78000	5,807
79	79000	6,055
80	80000	6,073
81	81000	6,386
82	82000	6,391
83	83000	6,566
84	84000	6,802
85	85000	7,007
86	86000	7,189
87	87000	7,485
88	88000	7,663
89	89000	7,813
90	90000	7,922
91	91000	8,161
92	92000	8,186
93	93000	8,551
94	94000	9,145
95	95000	9,353
96	96000	9,147
97	97000	9,694
98	98000	10,081
99	99000	9,656
100	100000	9,948

Rysunek 1: Tabela pomiarowa

Z uzyskanych pomiarów narysowałem wykres zależności czasu działania programu od zadanych danych.



Rysunek 2: Wykres czasu od ilości danych

Z wykresu odczytałem, że złożoność obliczeniowa zapisana w notacji dużego O jest równa $O(n^2)$, zatem jest to złożoność kwadratowa.

Wszystkie testy związane z badaniem złożoności obliczeniowej programu zostały przeprowadzone na laptopie MacBook Air M1. Laptop ten jest wyposażony w procesor Apple M1 z 8 rdzeniami CPU i 8 GB pamięci RAM. Testy zostały wykonane na systemie macOS przy użyciu domyślnego środowiska uruchomieniowego.

6 Źródła

Do pomocy przy implemnentacji kojeki priorytetowej:

- https://www.geeksforgeeks.org/priority-queue-set-1-introduction/
- https://www.sanfoundry.com/cpp-program-implements-priority-queue/
- ChatGPT

Do pomocy przy funkcji receive_message():

- ChatGPT
- https://www.tutorialspoint.com/cplusplus-program-to-read-file-word-by-word

Do pomocy przy badaniu złożoności obliczeniowej

- ChatGPT
- https://www.educative.io/answers/how-to-measure-time-intervals-in-cpp

Powyższe źródła nie zostały skopiowane bezpośrednio do programu. Zamiast tego, zostały wykorzystane jedynie jako wsparcie merytoryczne do samodzielnego rozwiązania zadania.