Politechnika Wrocławska

Raport
Mikroprojekt 2
Projektowanie i Analiza Algorytmów

7 maja 2023

Filip Mirski nr indeksu 264267

1 Wprowadznie

Celem ćwiczenia było przegotowanie danych do badania, a następnie przeprowadzenie sortowania trzema algorytmami. W moim przypadku były to Quick Sort, Bucket Sort i Merge Sort. Badanie zostało przeprowadzone na zbiorze będącym bazą danych filmów wraz z ich ocenami z serwisu IMDb. Przygotowanie danych polegało na usunięciu zbędnych wierszy, które nie posiadają oceny, a następnie zamieszczenie danych w dowolnej strukturze (w moim przypadku był to wektor o typie Ranking - napisanej strukturze). Po przygotowaniu przystąpiłem do badania czasów algorytmów na zbiorach danych 10,000; 100,000; 500,000; 962,472 (mniej niż milion ponieważ liczba wyników była powyżej 47,000).

```
        while (std::getline(file, tmpString)) {

            if (size != 0)
                if (ranking.size() >= size) break;

            std::stringstream tmp(tmpString);
            std::getline(tmp, tmpIndex, ',');
            std::getline(tmp, title, ',');
            while (title.front() == '"')
            {
                std::string tmpTmpName;
                if (title.back() == '"') break;
                std::getline(tmp, tmpTmpName, ',');
                title = title + ',' + tmpTmpName;
            }
            std::getline(tmp, pos, ',');
            if (!pos.empty()) {
                ranking.push_back({ stoi(tmpIndex), title, stoi(pos) });
            }
```

Rysunek 1: Fragment funkcji pobierającej dane z pliku przy okazji usuwająca puste miejsca

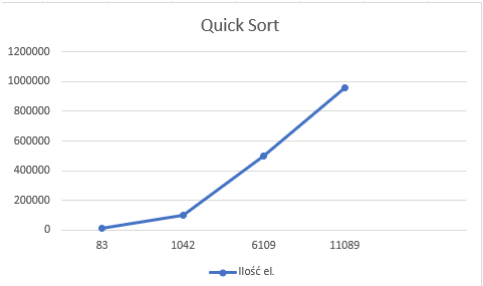
2 Quick Sort

Quick sort jest to algorytm polegający na metodzie "dziel i rządź" (algorytm rekurencyjny), jego charakterystyka polega na tym, że dane są dzielone na mniejsze części przy pomocy zmiennej pivot, natomiast te mniejsze elementy są już sortowane przed końcowym połączeniem. Poniżej fragment kodu:

```
{
    int pivot = Tab.at((begin + end) / 2).pos;
    int beginIx = begin - 1;
    int i = begin;
    while (i < end)
    {
        if (Tab.at(i).pos < pivot) {
            beginIx++;
            if(beginIx != i)
                swap(&beginIx, &i);
        }
        i++;
    }
    if (beginIx != end) {
        int incBeginIx = beginIx + 1;
        swap(&incBeginIx, &end);
    }
    return beginIx + 1;
}

void QuickSort::quickSort(std::vector<Ranking>& Tab, int begin, int end)
{
    if (begin < end) {
        int beginIx = partition(Tab, begin, end);
        quickSort(Tab, begin, beginIx - 1);
        quickSort(Tab, beginIx + 1, end);
```

Rysunek 2: Fragment funkcji z implementacją algorytmu quick sort

Algorytm w średnim przypadku powinien mieć złożoność obliczeniową O(nlogn), a w pesymistycznym $O(n^2)$, za to złożoność realna jest gdzieś pomiędzy tymi dwoma, nawet bardziej skłaniająca się do liniowej, ale już złożoność pamięciowa została utrzymana na poziomie O(1) ze względu na niewykorzystywanie dodatkowych struktur danych.



Rysunek 3: Wykres z wynikami czasowymi działania na określonych zbiorach algorytmem Quick Sort

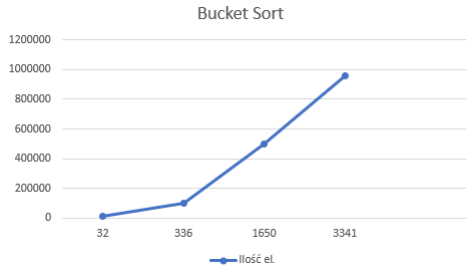
3 Bucket Sort

Algorytm bucket sort polega na stworzeniu mniejszych segregatorów na dane gdzie będą zamieszczane poszczególne dane, bo pudełka mają "etykiete" z zakresem jaki się w nim mieści i tym sposobem w średnim przypadku można osiągnąć O(n+k), gdzie k to ilość znanych elementów dodanich w pesymistycznym natomiast może dojść aż do $O(n^2)$.

```
std::vector<Ranking>* bucket = new std::vector<Ranking>[length]; // creating buckets (10)

for (int i = 0; i < length; i++)
{
    bucket[i] = std::vector<Ranking>();
}

for (int i = 0; i < Tab.size(); i++)
{
    bucket[Tab.at(i).pos - min].push_back(Tab.at(i));
}

int a = 0;

for (int i = 0; i < length; i++)
{
    int tmpSize = bucket[i].size();
    if (tmpSize > 0)
    {
        for (int j = 0; j < tmpSize; j++)
        {
            Tab.at(a) = bucket[i][j];
            a++;
        }
```

Rysunek 4: Fragment kodu implementacji Bucket Sort

W trakcie badań jak można zauważyć po zmieżonych czasach udało się uzyskać prawie złożoność liniowąm z lekkimi odchyleniami. Algorytm również posiada złożoność pamięciową na poziomie O(1) ze względu na zastasowanie wsakźników w celu "wypełniania pojemników", a nie tworzenia ich kopii.



Rysunek 5: Wykres przedstawiający wyniki

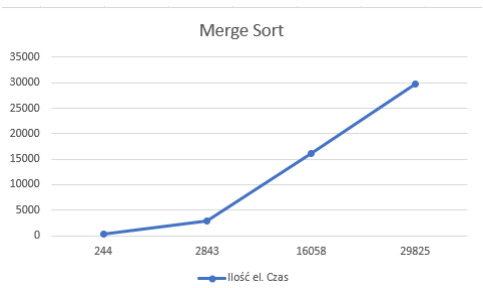
4 Merge Sort

Merge sort można nazwać gorszą wersją algorytmu quick sort ze względu na złożoność pamięciową O(n) i w pesymistycznych przypadku złoność obliczeniową jest taka sama jak w średnim, czyli O(nlogn). Sama operacja sortowania ponownie polega na dzieleniu tablicy na mniejsze problemy, ale tutaj elementy są sortowane w momencie ponownego scalania co w teorii znacznie spowalnia działanie algorytmu. W mojej implementacji ze

```
void MergeSort::merge(std::vector<Ranking>& Tab, int begin, int mid, int end) {
    int n1 = mid - begin + 1;
    int n2 = end - mid;
    std::vector<Ranking> L(n1), R(n2);
    for (int i = 0; i < n1; i++) {
        L[i] = Tab[begin + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = Tab[mid + 1 + j];
    }
    int i = 0, j = 0, k = begin;
    while (i < n1 && j < n2) {
        if (L[i].pos <= R[j].pos) {
            Tab[k] = L[i];
            i++;
        }
        else {
            Tab[k] = R[j];
            j++;
        }
        k++;
    }
```

Rysunek 6: Fragment kodu implementacji Merge Sort

względu na tworzenie podtablicy lewej i prawej jest złożoność pamięciowa na poziomie O(n) i złożoność czasowa osiągnieta O(nlogn)



Rysunek 7: Wykres przedstawiający wyniki

5 Wnioski

Jak można było się spodziewać najgorszym algorytmem do sortowania takiego zbioru danych był merge sort ze względu na sortowanie dopiero przy końcu działania rekurecji, drugi wypadł quick sort, a najlepiej spisał się bucket sort ze względu na zbiór danych i tym jaki mieliśmy sortować, czyli oceny od 1 do 10, co pozwliło na wykorzystanie wszystkich 10 pojemników.

6 Bibliografia

- Wprowadzenie do algorytmów Cormen Thomas H., Leiserson Charles E., Rivest Ronald L, Clifford Stein ISBN 978-83-01-16911-4
- `https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie`
- `https://pl.wikipedia.org/wiki/Sortowanie_szybkie`
- `https://en.wikipedia.org/wiki/Bucket_sort`
- Data Structures and Algorithms in C++, 2nd Edition Michael T. Goodrich, Roberto Tamassia, David M. Mount ISBN: 978-0-470-38327-8