Obliczanie złożoności obliczeniowej algorytmu Bubble Sort

Mateusz Kałwa

Problematyka:

Obliczyć złożoność obliczeniowa pesymistyczna, średnia i optymistyczna oraz w miare możliwości pesymistycznej. Sprawdzić dla losowych danych i tablicy posortowanej, czy liczba wykonanych operacji dominujacych zgadza sie oszacowaniami.

Algorytm sortowania Bubble sort wraz z zliczaniem operacji porównania i swap:

```cpp
// Bubble sort
template <class T>
std::map<std::string, int> sort(std::vector<T>& v)
{
    int number_of_swaps = 0;
    int number_of_ifs = 0;

    for (int it = 0; it < v.size(); ++it)
    {
        for (int j = 0; j < v.size() - 1; ++j)
        {
            ++number_of_ifs;
            if (v[j] > v[j + 1])
            {
                std::swap(v[j], v[j + 1]);
                ++number_of_swaps;
            }
        }
    }
    return { {"swap", number_of_swaps}, {"if", number_of_ifs} };
}
```

Obliczanie złożoności obliczeniowej:

Dominujaca operacja naszego algorytmu jest operacja porównania `if( v[j] > v[j+1] )`. Posiadamy w algorytmie dwie petle, jedna wykonuje sie n razy, natomiast druga $n-1$ razy. Zatem liczba użyć operacji porównania bedzie wynosiła $n \times (n-1)$. Złożoność obliczeniowa operacji porównania bedzie wynosiła $O(n^2)$. Bedzie sie to odnosiło do każdej możliwej sytuacji, dlatego weźmiemy pod uwage druga operacje dominujaca, która jest operacja `swap`.

1. Sytuacja optymistyczna:

   W tej sytuacji nasz wektor jest już posortowany. Dla takiej sytuacji operacja `swap` nigdy sie nie wykona. Przyjmujac poprzednie założenia, złożoność obliczeniowa porównań wyniesie $O(n^2)$, lecz `swapów` $O(1)$.

2. Sytuacja średnia:

   Niestety, średniej złożoności czasowej Bubble Sort nie można wyjaśnić w prosty sposób.

   Można z grubsza powiedzieć, że w średnim przypadku mamy około połowe mniej operacji wymiany niż w najgorszym przypadku, ponieważ około połowa elementów znajduje sie w prawidłowej pozycji w porównaniu do sasiedniego elementu.

   Oczekiwana liczba porównań dla losowo wybranej permutacji listy $(1, 2, \ldots, n)$ jest:

   $1.5 \cdot (n^2 - n \cdot \ln n - (\gamma + \ln(2) - 1) \cdot n) + O(\sqrt{n})$,

   Gdzie γ oznacza stała Eulera-Mascheroniego; to oczekiwana liczba swapów $\frac{1}{4}(n^2 - n)$.

   Źródło: German Wikipedia

   Zatem złożoność obliczeniowa obu operacji bedzie wynosiła $O(n^2)$.

3. Sytuacja pesymistyczna:

   W tej sytuacji nasz wektor jest odwrotnie posortowany. W takiej sytuacji operator `swap` bedzie wykonany $n-1$ razy dla pierwszej iteracji, dla drugiej $n-2$ itd. Korzystajac ze wzoru na sume ciagu arytmetycznego możemy obliczyć ile razy wykona sie operacja `swap`: $(n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$. Przyjmujac poprzednie założenia, złożoność obliczeniowa obu operacji bedzie wynosiła $O(n^2)$.

Obliczenia dla wszystkich przypadków dla n = 1000:

1. Sytuacja optymistyczna:

   Liczba operacji:
   if: $n \times (n-1) \Rightarrow 1000 \times 999 = 999000$
   swap: 0

2. Sytuacja średnia:

   Liczba operacji:
   if: $n \times (n-1) \Rightarrow 1000 \times 999 = 999000$
   swap: $\frac{n^2-n}{4} \Rightarrow \frac{1000000-1000}{4} = 249750$

3. Sytuacja pesymistyczna:

   Liczba operacji:
   if: $n \times (n-1) \Rightarrow 1000 \times 999 = 999000$
   swap: $\frac{n\times(n-1)}{2} \Rightarrow \frac{1000\times 999}{2} = 499500$

Program który wypisuje zliczenia dla wszystkich przypadków:

```cpp
auto main() -> int
{
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<int> r_num(1, 1000000);

    std::vector<int> vec;
    int n = 1000;

    std::cout << "Bubble Sort: [ n = " << n << " ]" << std::endl << std::endl <<
        std::endl;

    // Optimistic situation
    for (int i = 0; i < n; ++i)
        vec.push_back(i);

    std::map<std::string, int> dominants = sort<int>(vec);

    std::cout << "Optimistic situation:" << std::endl << std::endl;
    std::cout << "\tDominant operations:\n\t\tIf:\t" << dominants["if"] << "\n\t\
        tSwap:\t" << dominants["swap"] << std::endl << std::endl;
    vec.clear();

    // Avrage situation
    for (int i = 0; i < n; ++i)
        vec.push_back(r_num(gen));

    dominants = sort<int>(vec);

    std::cout << "Avrage situation:" << std::endl << std::endl;
    std::cout << "\tDominant operations:\n\t\tIf:\t" << dominants["if"] << "\n\t\
        tSwap:\t" << dominants["swap"] << std::endl << std::endl;
    vec.clear();

    // Pessimistic situation
    for (int i = n - 1; i >= 0; --i)
        vec.push_back(i);

    dominants = sort<int>(vec);

    std::cout << "Pessimistic situation:" << std::endl << std::endl;
    std::cout << "\tDominant operations:\n\t\tIf:\t" << dominants["if"] << "\n\t\
        tSwap:\t" << dominants["swap"] << std::endl << std::endl;
    vec.clear();
}
```

Wyniki programu:

Bubble Sort: [**n = 1000**]

Optimistic situation:

```
Dominant operations:
        If:     999000
        Swap:   0
```

Average situation:

```
Dominant operations:
        If:     999000
        Swap:   245515
```

Pessimistic situation:

```
Dominant operations:
        If:     999000
        Swap:   499500
```

Podsumowanie:

Przeprowadzone obliczenia wykazuja zgodność z wynikami, które uzyskaliśmy poprzez nasze wcześniejsze wyliczenia.