2022

Merge-sort verslag

IN DEZE DOCUMENT LEG IK UIT HOE IK DE MERGE SORT HEB GEIMPLEMENTEERD IN C++

CEYHUN ÇAKIR | 1784480

Inhoud

Inleiding

In deze opdracht moest ik een merge sort algoritme maken gebaseerd op de pseudo code die op canvas is gegeven. Ik had een keuze moeten maken tussen de iteratieve versie en de recursieve versie. Ik zelf heb voor de recursieve manier gekozen aangezien ik de vorige opdracht al op de iteratieve manier heb gemaakt. Verder was dit opdracht bedoelt als een opzet aangezien we later gebruik gaan maken van multithreading

Wat heb ik gedaan?

Ik ben eerst begonnen met het analyseren van de pseudo code. Na dat ik een begrip van het pseudo code kreeg begon ik met het opzetten van de environment. Na dat ik mijn (.cpp) bestanden had opgezet ging ik beginnen met het schrijven van het merge sort algoritme. Hieronder in de afbeeldingen zien je hoe ik sommige delen van het algoritme heb gemaakt.

```
if(array_01.size() == 0) {

  // head van array_02
  std::vector<T> array_02_head = {array_02.begin(), array_02.end() - (array_02.size() - 1)};
  // tail van array_02
  std::vector<T> array_02_tail = {array_02.begin() + 1, array_02.end()};
  // sorted lijst van array_01 en array_02_tail
  std::vector<T> sorted = merge_sort(array_01, array_02_tail);

  // output is array_02_head + sorted
  std::vector<T> output = {};
  // we reserve voor de output vector de size van array_02_head en de sorted list size
  output.reserve(array_02_head.size() + sorted.size());

  output.emplace_back(array_02_head[0]);

  for(auto e : sorted) {
    output.emplace_back(e);
  }

  // returning output
  return output;
```

We zien bijvoorbeeld dat in het pseudo code een head en een tail van een array nodig hebben om dit algoritme compleet te maken. Ik heb hier voor een subvector aangemaakt die de head en de tail van een vector pakt.

Ook maak ik ruimte aan in het output vector. Dit doe ik voornamelijk zodat de vector precies de grootte heeft om de elementen te accepteren.

Verder naast het merge sort algoritme heb ik nog een functie aangemaakt die op een recursieve manier de merge sort functie gebruikt. Voor de code heb ik pseudo code gevolgd die op canvas geleverd was.

```
template <typename T>
std::vector<T> recursive_merge_sort(std::vector<T> data) {
  if(data.size() == 1) {
    return data;
  } else {
    std::size_t const middle = data.size() / 2;
    std::vector<T> first(data.begin(), data.begin() + middle);
    std::vector<T> second(data.begin() + middle, data.end());
    return merge_sort(recursive_merge_sort(first), recursive_merge_sort(second));
  }
}
```

Wat liep ik tegen aan?

Ik liep vooral in het begin aan hoe ik de head en de tail kon pakken van een vector. Door wat research te doen en aan de docent te vragen, kreeg ik te horen dat een subvector voor dit probleem handig zou kunnen zijn. Verder ben ik gaan uitzoeken hoe ik de indexering het beste kan doen om dus de head en de tail te krijgen.

```
// head van array_02
std::vector<T> array_02_head = {array_02.begin(), array_02.end() - (array_02.size() - 1)};
// tail van array_02
std::vector<T> array_02_tail = {array_02.begin() + 1, array_02.end()};
```

Wat kan ik verbeteren?

Eventueel zou ik de head en de tail die ik aanmaak een keer kunnen aanmaken dan het steeds dubbel aanmaak bij andere stukjes in het code. Ook kan ik multithreading gebruiken om het proces sneller te maken, maar dat komt later in het opdracht terug. Verder zou ook meerdere tests kunnen maken dan nu dat ik alleen de snelheid van het algoritme bereken.

Uitkomst

In de afbeeldingen hieronder zie je de gegeven input en de uitkomst van de merge sort algoritme.

Gegeven input:

```
std::vector<int> data = {75, 83, 40, 24, 42, 63, 23, 22, 34, 58, 65, 100, 30, 57, 90, 14, 46};
```

De output:

```
14 | 22 | 23 | 24 | 30 | 34 | 40 | 42 | 46 | 57 | 58 | 63 | 65 | 75 | 83 | 90 | 100 | Time taken by merge sort algoritme: 140386 microseconds
```