Penjelasan Kode dan Analisis Kompleksitas

Intersect

Berikut kode yang terdapat pada slide saya implementasikan ulang dalam bahasa pemrograman C++, dengan struktur data vektor. Kompleksitas waktu untuk algoritma tersebut adalah $N + M$, dengan N adalah panjang dari *posting lists* pertama dan M adalah panjang dari *posting lists* kedua.

```
// kompleksitas waktu length(v1) + length(v2)
vector <int> Intersect(vector<int> v1, vector<int> v2) {
  auto p1 = v1.begin();
  auto p2 = v2.begin();
  vector <int> answer;
  while(p1 != v1.end() && p2 != v2.end()){ // Pada slide mengecek apakah null, dalam
kasus ini mengecek apakah
                                           // sudah berada di ujung container
    if(docID(p1) == docID(p2)){
      add(answer, docID(p1));
      p1 = next(p1);
      p2 = next(p2);
    }else if(docID(p1) < docID(p2)){
      p1 = next(p1);
    }else{
      p2 = next(p2);
    }
  }
  return answer;
}
```

Terdapat juga algoritma lain, dengan kompleksitas waktu $\min(N, M) \times \log(\max(N, M))$.

```
// intersection binary search
// Kompleksitas waktu: min(length(v1), length(v2)) * log(max(length(v1), length(v2)))
vector <int> IntersectBinarySearch(vector <int> v1, vector <int> v2){
  if(v1.size() > v2.size()) swap(v1, v2); // Memastikan ukuran v1 lebih kecil
  vector <int> answer;
  for(auto &value : v1){
    // Binary search apakah ada "value" pada v2
    int lo = 0;
    int hi = (int) v2.size() - 1;
    while(lo < hi){
      int mid = (lo + hi) / 2;
      if(v2[mid] < value) lo = mid + 1;
      else hi = mid;
    }
    if(v2[lo] == value) add(answer, value);
  }
```

```
  return answer;
}
```

Algoritma ini bekerja lebih cepat ketika $\max(N, M)$ nilainya cukup besar.

Pada intinya akan dilakukan iterasi pada v_1 dengan mengecek apakah elemen yang dilakukan iterasi saat ini terdapat pada v_2. Karena v_2 memiliki invarian nilainya yang terurut menaik, maka *binary search dapat dilakukan*.

Union dan Difference

```
// returns v1 or v2
// kompleksitas waktu length(v1) + length(v2)
vector <int> Union(vector<int> v1, vector<int> v2) {
  auto p1 = v1.begin();
  auto p2 = v2.begin();
  vector <int> answer;
  while(p1 != v1.end() || p2 != v2.end()){
    if(p1 == v1.end()){
      // Kasus p1 di ujung, maka langsung tambahkan p2
      add(answer, docID(p2));
      p2 = next(p2);
    } else if(p2 == v2.end()){
      // Kasus p2 di ujung, maka langsung tambahkan p1
      add(answer, docID(p1));
      p1 = next(p1);
    } else if(docID(p1) == docID(p2)){
      // Jika sama, maka tambahkan salah satu saja
      add(answer, docID(p1));
      p1 = next(p1);
      p2 = next(p2);
    }else if(docID(p1) < docID(p2)){
      // Tambahkan yang lebih kecil
      add(answer, docID(p1));
      p1 = next(p1);
    }else{
      add(answer, docID(p2));
      p2 = next(p2);
    }
  }
  return answer;
}

// Returns v1 and not v2
// kompleksitas waktu length(v1) + length(v2)
vector <int> Difference(vector<int> v1, vector<int> v2) {
  auto p1 = v1.begin();
  auto p2 = v2.begin();
  vector <int> answer;
```

```
  while(p1 != v1.end() || p2 != v2.end()){
    if(p1 == v1.end()){
      // Jika p1 sudah habis, bisa berhenti
      break;
    } else if(p2 == v2.end()){
      // Jika p2 sudah habis, tambahkan p1 sampai habis
      add(answer, docID(p1));
      p1 = next(p1);
    } else if(docID(p1) == docID(p2)){
      // Jika sama, maka lewatkan
      p1 = next(p1);
      p2 = next(p2);
    }else if(docID(p1) < docID(p2)){
      // Jika p1 lebih kecil, tambahkan dan lanjut
      add(answer, docID(p1));
      p1 = next(p1);
    }else{
      // Selain itu, lanjut saja p2nya
      p2 = next(p2);
    }
  }
  return answer;
}
```