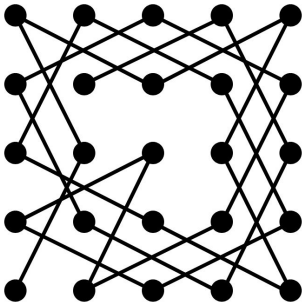
Naive Trie Implementation

COMTRAN (Competitive Traveling Nerd)

Mengenai COMTRAN



COMTRAN atau Competitive Traveling Nerd adalah komunitas pecinta pemrograman kompetitif di Universitas Telkom yang mempelajari berbagai macam algoritma dan struktur data dengan fokus kompetisi pemrograman kompetitif. Kami merasa hal ini menyenangkan dan menantang.

Kata “Competitive” menunjukkan fokus utama kami. Kata “Traveling” merujuk kepada target kami yaitu pergi ke final dan memenangkan piala. Kata “Nerd” adalah bagaimana orang-orang menyebut kami. Logo COMTRAN secara garis besar merupakan salah satu solusi dari Knight’s Tour Puzzle pada 5x5 papan catur.

Mengenai Implementation Note Ini

Seringkali kita kesulitan untuk melakukan implementasi walaupun sebenarnya sudah mengerti konsep dasarnya. Untuk itu, COMTRAN hadir dengan Implementation Note ini sebagai media belajar yang harapannya dapat membantu teman-teman belajar algoritma dan struktur data.

Kegunaan Naive Trie

Naive Trie merupakan struktur data yang digunakan untuk menyimpan kumpulan kata secara efisien karena nanti kumpulan kata ini akan dicari dan dihitung jumlahnya. Pencarian yang dilakukan biasanya menggunakan substring. Misal, terdapat nama “Wisnu” dan “Winda”. Ketika dicari kata “Wi”, maka yang muncul keduanya. Namun, ketika dicari “Wis”, maka yang muncul hanya satu. Sementara itu, ketika dicari “Wa”, maka tidak ada yang muncul.

Naive Trie dapat digunakan untuk masalah yang berhubungan dengan *prefix substring*. Biasanya digunakan ketika masalahnya mencari jumlah elemennya saja. Ketika data yang ada sedikit, kita bisa menggunakan *naive search*, melakukan pencarian secara *linear*. Namun, ketika data yang ada sangat banyak, hal tersebut akan memakan waktu sehingga terjadi TLE.

Representasi Naive Trie

Implementasi yang akan dilakukan pada *naive Trie* ini akan memanfaatkan struktur data *hash map*. Dikatakan *naive* karena implementasinya tidak benar-benar Trie namun bekerja seperti Trie. Implementasi yang akan dilakukan ini akan sangat memakan *memory*. Misal, kita ingin menyimpan kata "Wisnu" dan kata "Winda", maka representasi dari *hash map* tersebut seperti berikut.

```
Wisnu = 1, Wisn = 1, Wis = 1, Wi = 2, W = 2
Winda = 1, Wind = 1, Win = 1, Wi = 2, W = 2
```

Maka dari itu, ketika kita mencari "Wi" hasilnya adalah 2. Dengan kata lain, ada dua elemen yang memiliki substring "Wi." Naive Trie dapat dikategorikan menjadi *variant* dari Trie yaitu Hash Trie.

Implementasi Naive Trie

Untuk implementasi Naive Trie, kita dapat memanfaatkan struktur data *hash map*. Sebelum membangun Trie, kita definisikan terlebih dahulu tipe data *unsigned integer* karena jumlah suatu elemen tidak mungkin negatif.

```
#define uint unsigned int
```

Kemudian, bangun struktur data baru bernama Trie yang memanfaatkan *hash map* dengan *key* berupa String dan *value* berupa Uint.

Implementasi Naive Trie

```
struct Trie {
  map<string, uint> records;
}
```

Selanjutnya, bangun *method* di dalam *struct* tersebut sesuai dengan kebutuhan. Dalam catatan ini kita akan membuat *method* untuk menambahkan elemen baru (*insert*), mencari suatu elemen (*contains*), dan menghitung jumlah elemen dengan *prefix* tertentu (*count*).

Pertama-tama, mari kita lakukan implementasi *method* untuk *insert* terlebih dahulu.

Implementasi Naive Trie

```
void insert(string str) {
  uint n = str.size();
  string prefix = "";

  for (uint i = 0; i < n; i++) {
    prefix += str[i];
    if (this->records.find(prefix) == this->records.end())
      this->records[prefix] = 0;

    this->records[prefix]++;
  }
}
```

Implementasi Naive Trie

Dari algoritma *insert* di *slide* sebelumnya, dapat kita lihat bahwasannya *variable* **prefix** melakukan penyambungan String dari **str[0]** hingga ke **str[n - 1]**. Untuk masing-masing **prefix**, kita periksa apakah sudah terdaftar di **records** atau belum. Jika belum, kita inisialisasikan menjadi nol. Kemudian, *increment* nilai *value* dari **records[prefix]**.

Selanjutnya, kita implementasi *method* untuk melakukan pencarian **records** dan *method* untuk menghitung banyaknya elemen dengan *substring* **prefix**.

Implementasi Naive Trie

```
bool contains(string str) {
  return this->records.find(prefix) == this->records.end());
}

uint count(string str) {
  if (!this->contains(str)) return 0;

  return this->records[str];
}
```

Implementasi untuk kedua *method* di atas cukup mudah. Untuk *method* **contains**, jika *iterator* mencapai **end()** berarti tidak ada elemen yang dicari. Untuk *method* **count**, jika elemen dengan *prefix* yang dicari tidak ada, maka akan dikembalikan nilai nol.

Penggunaan Naive Trie

Untuk menggunakan Naive Trie yang sudah kita implementasi, dapat digunakan *source code* di bawah sebagai bahan uji coba.

```
int main() {
  Trie trie;
  trie.insert("setyo");

  cout << trie.count("setyo") << endl;

  return 0;
}
```

Kita pasti akan dapatkan keluaran berupa “0”.

Source Code Lengkap

Kode sumber untuk Naive Trie dapat ditemukan di *repository* GitHub milik COMTRAN. Lebih tepatnya terdapat di comtran/implementations.

Latihan Naive Trie

Untuk latihan, *problem* yang dapat dituntaskan menggunakan Naive Trie beberapa diantaranya adalah sebagai berikut.

- https://www.hackerrank.com/challenges/contacts/problem
- https://open.kattis.com/problems/bing

Pertanyaan atau Koreksi

Jika terdapat pertanyaan atau koreksi dari *slide* ini kami persilakan kamu untuk menghubungi kami melalui surel <info@comtran.club>.

Fin