Solusi Varian *Travelling Salesman Problem* dengan *Dynamic Programming* dan *Branch-and-bound*

Hanif Arroisi Mukhlis (13519072)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): hanif.arroisi@gmail.com

***Abstract*—Dinas Kesehatan Kabupaten Gresik, yang membawahi 30 Puskesmas di daratan, membutuhkan pemecahan masalah distribusi obat dan alat kesehatan, serta pemantauan Puskesmas. Adanya keterbatasan anggaran tenaga dan jam kerja menyebabkan perlunya solusi yang efektif. Dicari sekumpulan jalur-jalur minimum yang memenuhi kriteria-kriteria tertentu. Solusi dicari dengan gabungan *dynamic programming* dan *branch-and-bound*. Ditemukan sebuah solusi dengan berat total 870.**

Keywords—Travelling Salesman Problem; Dynamic Programming; Puskesmas; Kesehatan;

I. PENDAHULUAN

Travelling salesman problem menjadi masalah klasik dalam berbagai aspek kehidupan. Masalah ini muncul dimana-mana, mulai dari *networking*, distribusi barang, hingga transportasi. Walapun demikian, masalah ini masih sulit untuk dipecahkan.

Pada tahun 2004, permasalahan TSP berhasil diselesaikan untuk 24,978 kota di Swedia, dengan panjang 855,597. Peneliti berhasil membuktikan bahwa tidak ada solusi yang lebih pendek [1]. Solusi ini memecahkan rekor permasalahan TSP terbesar, yang sebelumnya dipegang untuk 15,112 kota di Jerman [2].

Dinas Kesehatan Kabupaten Gresik, yang membawahi 30 Puskesmas di daratan, membutuhkan pemecahan masalah distribusi obat dan alat kesehatan, serta pemantauan Puskesmas. Adanya keterbatasan anggaran tenaga dan jam kerja menyebabkan perlunya solusi yang efektif.

II. RUMUSAN MASALAH

Dicari jalur-jalur yang memiliki total waktu tempuh minimum dan memenuhi batasan sebagai berikut:

- Jalur harus dimulai dan berakhir di Dinas Kesehatan Kabupaten Gresik.
- Solusi harus meraih kesemua 30 Puskesmas.
- Tidak diharuskan sebuah jalur melalui semua Puskesmas.
- Tidak ada Puskesmas yang dilalui 2 kali, baik oleh jalur yang sama maupun berbeda.
- Setiap jalur harus dapat ditempuh dalam satu hari kerja (dari pukul 8 pagi hingga pukul 4 sore).

III. LANDASAN TEORI

Travelling Salesman Problem (TSP) adalah permasalahan optimasi pada teori graf yang mana tiap titik (kota) pada graf dihubungkan dengan sisi (rute) berarah, dimana beban suatu sisi mengindikasikan jarak dua kota [3]. Hingga saat ini, belum ada algoritma solusi dari TSP yang dapat bekerja dengan efisien. Namun untuk skala permasalahan ini, komputer dapat menyelesaikan dengan waktu yang cukup masuk akal.

Salah satu algoritma yang dapat menyelesaikan permasalahan TSP adalah algoritma Held-Karp. Algoritma Held-Karp adalah algoritma pemrogaman dinamis yang diusulkan oleh Held dan Karp untuk menyelesaikan TSP. Performa algoritma ini cukup cepat untuk ukuran kecil dan menengah, dengan kebutuhan memori yang rendah.

Dynamic Programming adalah cara untuk mengoptimasi solusi dengan memecah permasalahan menjadi sub-masalah yang bisa jadi identik secara rekursif. Tergantung urutan penyelesaian, *dynamic programming* dapat dibagi menjadi dua cara, yaitu *top-down* dan *bottom-up*.

Metode *top-down* memecahkan masalah dari atas, kemudian menyelesaikan sub-masalah jika diperlukan. Sebaliknya, *bottom-up* menyelesaikan sub-masalah yang paling sederhana terlebih dahulu, kemudian membangun solusi untuk masalah yang lebih rumit.

Branch-and-bound adalah pengembangan dari metode *backtracking*, dengan memberikan fungsi pembatas. Simpul akan dimatikan dan tidak diekspansi ketika suatu kriteria pembatas dipenuhi. Proses ekspansi simpul dilakukan secara rekursif, berhenti ketika tidak ada simpul hidup yang dapat diekspansi. Fungsi pembatas dapat di-*update* secara dinamik.

IV. METODOLOGI

Pemecahan masalah dibagi menjadi 2 tahap, yaitu:

1. Mencari semua *cycle* minimum yang mengandung *vertex* awal.
2. Mencari set *coverage* penuh minimum.

A. *Pencarian Siklus Minimum*

Untuk memecahkan tahap pertama, penulis menggunakan modifikasi dari algoritma Held-Karp. Modifikasi algoritma dilakukan sehingga ia dapat mencari semua jalur yang:

- Berawal dan berakhir di sebuah *vertex* tertentu.
- Memiliki total *cost* minimum untuk set *vertex* yang dilewati.
- *Cost* tidak melebihi suatu kapasitas.

Penulis menggunakan strategi *dynamic programming bottom-up* dengan memoisasi untuk mengimpementasikan algoritma. Deskripsi implementasi adalah sebagai berikut:

TABLE I. DEFINISI SIMBOL UNTUK ALGORITMA 1

Simbol	Definisi
K	Kapasitas.
v_0	Vertex awal.
S	Set *vertex* yang tidak mengandung vertex awal.
$g(x, S)$	*Cost* minimum jalur yang melewati semua *vertex* di S dan berakhir di x.[a]
C_{xy}	*Cost* dari x ke y.
$p(x, S)$	*Vertex* sebelum x di jalur minimum melalui semua *vertex* di $S \cup \{x\}$.[a]

[a] *Fungsi g dan p direpresentasikan sebagai map/dictionary*

Langkah-langkah algoritma:

3. Untuk semua *vertex* x yang bertetangga dengan *vertex* awal v_0:
   - Definisikan $g(x, \emptyset)$ dan $p(x, \emptyset)$ sebagai berikut:
   $$g(x, \emptyset) = C_{v_0 x}$$
   $$p(x, \emptyset) = v_0$$
   - Masukkan ke *queue* item $(x, \emptyset)$.
4. Ambil item $(x, \emptyset)$ dari *queue*.
5. Untuk semua *vertex* y yang bertetangga dengan x:
   - Jika *cost* $C_{xy} + g(y, S) > K$, jalur akan melebihi kapasitas. Sehingga jalur ini harus dibuang.
   - Jika $g(y, S \cup \{x\}) \leq C_{xy} + g(y, S)$, jalur bukanlah jalur minimum. Sehingga jalur ini harus dibuang.
   - Jika tidak, definisikan ulang sebagai berikut:
   $$g(y, S \cup \{x\}) = C_{xy} + g(y, S)$$
   $$p(y, S \cup \{x\}) = x$$
   - Masukkan ke queue item $(y, S \cup \{x\})$ jika ia belum ada dan y bukan *vertex* awal.
6. Ulangi langkah 2-3 hingga queue kosong.

Hasil akhir dari algoritma adalah:

- Cost siklus yang berhasil dihasilkan adalah $g(v_0, S)$.
- Jalur yang dilewati mengikuti relasi rekurens berikut:

$$v_{i+1} = p\left(v_i, S - \bigcup_{k=0}^{i} \{v_k\}\right)$$

B. *Pencarian Coverage Penuh Minimum*

Untuk mencari *coverage* penuh minimum, penulis menggunakan strategi *branch and bound*. Secara garis besar, algoritma dapat dideskripsikan sebagai berikut:

TABLE II. DEFINISI SIMBOL UNTUK ALGORITMA 2

Simbol	Deskripsi
S_C	Set semua siklus yang dihasilkan dari tahap 1. Set ini adalah kandidat solusi.
$p \in S_C$	Sebuah siklus.
$g(p)$	*Cost* siklus p.
$C(p)$	*Coverage* siklus, yaitu set *vertex* yang dilalui siklus, tidak termasuk *vertex* awal.
$A \subseteq S_C$	Sebuah *node*.

- Ruang status adalah *powerset* dari S_C.
- *Cost* dari *node* adalah jumlah dari semua siklus didalamnya.
- Pembangkitan anak dari *node* $A = \{p_1, p_2, \ldots, p_n\}$ adalah sebagai berikut:

$$A \cup p_{n+1} | p_{n+1} \in S_C, C(p_{n+1}) \cap \bigcup_{i=0}^{n} C(p_i) = \emptyset$$

- *Traversal* dimulai dari anak yang meng-*cover* paling besar. Jika *coverage* sama, maka diambil yang memiliki *cost* terkecil.
- *Node* diterminasi jika seluruh siklus didalamnya telah melewati semua *vertex*.
- *Node* dimatikan jika *cost* ia lebih besar dibandingkan dengan *cost node* terminal terbaik sekarang.

Hasil akhir dari algoritma adalah set siklus yang meng-*cover* semua *vertex* dan memiliki total *cost* minimum.

Karena keterbatasan kemampuan komputasi, penulis melakukan beberapa konsesi, yaitu:

- Jumlah *vertex* maksimum adalah 32 (termasuk *vertex* awal). Hal ini diperlukan untuk menyederhanakan

representasi set *vertex* menjadi sebuah *integer* 32-bit, dimana setiap bit merepresentasikan keberadaan suatu *vertex* tertentu didalam set.

- Representasi *node* diganti menjadi sebuah *list* siklus. Hal ini diperlukan untuk mempercepat *traversal* dengan mengorbankan *node* akan dicek kembali sebanyak permutasinya.[1]
- *Node* anak dipangkas sehingga hanya anak-anak yang meng-*cover* paling besar dan 1 lebih kecil yang akan dipertahankan. Hal ini diperlukan untuk mempercepat komputasi dengan mengorbankan kemungkinan solusi optimum di *node* yang terpangkas.

C. *Masukan Program*

Program menerima masukan berupa file csv yang berisi matriks ketetanggaan setiap *vertex*. Berat setiap entri adalah waktu tempuh rute tersebut dalam menit. Waktu tempuh dihitung menggunakan Google Maps.

Penulis menggunakan waktu 8 jam (8 * 60 menit) sebagai kapasitas maksimum setiap jalur, karena diasumsikan perjalanan dilakukan selama satu hari kerja (dari pukul 8 pagi hingga pukul 4 sore).

D. *Keluaran Program*

Proses menjalankan program membutuhkan waktu 15-30 menit dengan 8 thread. Fig. 1 berisi *screenshot* hasil akhir program.

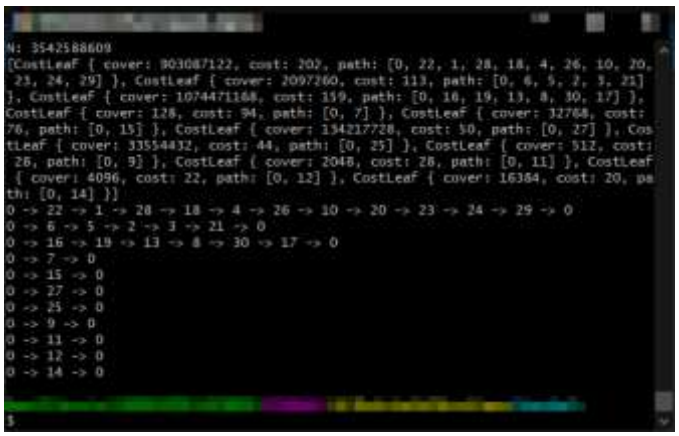


Fig. 1. *Screenshot* keluaran program.

Keluaran program berupa serangkaian baris. Setiap baris adalah sebuah siklus yang menunjukkan jalur dari sebuah ID node ke ID berikutnya. ID didefinisikan dari urutan kemunculan *vertex* di file input.

Jalur dan nama simpul dari hasil keluaran adalah:

- DINAS KESEHATAN → Puskesmas Nelayan → Puskesmas Alun-alun → Puskesmas Sukomulyo → Puskesmas Manyar → Puskesmas Bungah → Puskesmas Sidayu → Puskesmas Dukun → Puskesmas Mentaras → Puskesmas Panceng → Puskesmas Sekapuk → Puskesmas Ujungpangkah → DINAS KESEHATAN

  Cost: 202 menit
- DINAS KESEHATAN → Puskesmas Dadap Kuning → Puskesmas Cerme → Puskesmas Balongpanggang → Puskesmas Benjeng → Puskesmas Metatu → DINAS KESEHATAN

  Cost: 113 menit
- DINAS KESEHATAN → Puskesmas Kepatihan → Puskesmas Menganti → Puskesmas Karangandong → Puskesmas Driyorejo → Puskesmas Wringin Anom → Puskesmas Kesambenkulon → DINAS KESEHATAN

  Cost: 159 menit
- DINAS KESEHATAN → Puskesmas Dapet → DINAS KESEHATAN

  Cost: 128 menit
- DINAS KESEHATAN → Puskesmas Kedamean → DINAS KESEHATAN

  Cost: 76 menit
- DINAS KESEHATAN → Puskesmas Slempit → DINAS KESEHATAN

  Cost: 50 menit
- DINAS KESEHATAN → Puskesmas Sembayat → DINAS KESEHATAN

  Cost: 44 menit
- DINAS KESEHATAN → Puskesmas Duduk Sampeyan → DINAS KESEHATAN

  Cost: 28 menit
- DINAS KESEHATAN → Puskesmas Gending → DINAS KESEHATAN

  Cost: 28 menit
- DINAS KESEHATAN → Puskesmas Industri → DINAS KESEHATAN

  Cost: 22 menit
- DINAS KESEHATAN → Puskesmas Kebomas → DINAS KESEHATAN

  Cost: 20 menit

Total *cost* dari semua jalur adalah 202 + 113 + 159 + 128 + 76 + 50 + 44 + 28 + 28 + 22 + 20 = 870 menit.

V. KESIMPULAN

Algoritma mampu mengoptimasi permasalahan varian dari *Travelling Salesman Problem*. Untuk mempercepat program, dilakukan berbagai teknik dan konsesi. Algoritma mampu menemukan solusi dalam waktu kurang dari 30 menit dengan *cost* total 870 menit. Belum diketahui jika solusi ini adalah solusi optimum.

[1]Hal ini dapat dimitigasi dengan mensyaratkan pengurutan list dan menghapus anak yang tidak terurut.

VI. ACKNOWLEDGMENT

Penulis ingin memberikan terima kasih kepada dr. Anik Luthfiyah M. Ked. Trop. yang telah mensuplai data dan kemampuan komputasi. Penulis juga ingin berterimakasih kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. yang tanpanya makalah ini tidak akan terjadi.

REFERENCES

[1] https://www.math.uwaterloo.ca/tsp/sweden/ diakses pada tanggal 10 Mei 2021.

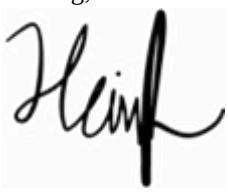
[2] https://www.math.uwaterloo.ca/tsp/d15sol/index.html diakses pada tanggal 10 Mei 2021.

[3] Britannica, The Editors of Encyclopaedia. "Traveling salesman problem". Encyclopedia Britannica, https://www.britannica.com/science/traveling-salesman-problem. Diakses pada tanggal 10 Mei 2021.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Mei 2021



Hanif Arroisi Mukhlis (13519072)

APENDIKS A
Source Code

I. MAIN.RS

```
pub mod btree;
pub mod tsp;

use std::collections::HashMap;
use std::sync::atomic::{AtomicUsize, Ordering};

use csv::ReaderBuilder;
use petgraph::prelude::*;

use btree::BTree;

pub type Weight = usize;

// Change with input file path
const FILEPATH: &str = "JARAK PUSKESMAS.csv";

static n_: AtomicUsize = AtomicUsize::new(0);
static end: AtomicUsize = AtomicUsize::new(0);

fn main() {
    let mut interned = HashMap::new();
    let mut index = Vec::new();
    let map = &mut interned;
    let iname = &mut index;
    let mut f = ReaderBuilder::new()
        .delimiter(b';')
        .has_headers(true)
        .from_path(FILEPATH)
        .unwrap();

    let mut graph = UnGraph::<_, _, tsp::NodeIndex>::default();

    fn intern_string(
        map: &mut HashMap<&'static str, tsp::NodeIndex>,
        iname: &mut Vec<&'static str>,
        str: &str,
    ) -> (bool, tsp::NodeIndex) {
        if let Some(v) = map.get(str) {
            (true, *v)
        } else {
            let l = iname.len();
            let s = Box::leak(String::from(str).into_boxed_str());
            iname.push(s);
            map.insert(s, l);
            (false, l)
        }
    }

    for s in f.headers().unwrap().iter().skip(1) {
        let (b, i) = intern_string(map, iname, s);
        if !b {
            graph.add_node(iname[i]);
        }
```

```
}
for rec in f.records() {
    let rec = rec.unwrap();
    let (b, src) = intern_string(map, iname, &rec[0]);
    if !b {
        graph.add_node(iname[src]);
    }
    for (dst, s) in rec.iter().skip(1).enumerate() {
        if s.len() == 0 {
            continue;
        }
        let v = Weight::from_str_radix(s, 10).unwrap();
        graph.update_edge(src.into(), dst.into(), v);
    }
}
debug_assert_eq!(petgraph::algo::connected_components(&graph), 1);

let start = 0;

let (_, _, mut btree) = tsp::mtsp(&graph, start, 8 * 60);
let mut total = (!(1 << graph.node_count()) + 1) | (1 << start);
for i in btree.iter_leaf() {
    total |= i.cover;
}
debug_assert_eq!(!total, 0);
btree.delete_leaf(&0);

fn best_cover_threaded<'a>(
    btree: &'a tsp::CostBTree,
    current: tsp::NodeSet,
) -> (Vec<&'a tsp::CostLeaf>, Weight) {
    n_.fetch_add(1, Ordering::Relaxed);
    let mut potential: Vec<_> = btree
        .iter_subset(!current)
        .map(|v| (v, v.cover.count_ones() as usize))
        .collect();
    let n_bound = potential.iter().fold(0, |i, (_, v)| i.max(*v));
    if n_bound > 1 {
        potential.retain(move |(_, v)| *v >= (n_bound - 1));
    }
    if potential.is_empty() {
        return (vec![], Weight::MAX);
    }
    potential.sort_unstable_by_key(|(v, x)| (*x, core::cmp::Reverse(v.cost), v.cover));

    // Thread count
    let mut vecs = vec![Vec::new(); 8];
    let mut i = 0usize;
    for (n, _) in potential.into_iter() {
        vecs[i].push(n);
        i = (i + 1) % vecs.len();
    }

    let mut handles = vec![];
    for v in vecs.into_iter() {
        let current = current;
        let mut nodeset: Vec<&'static tsp::CostLeaf> = Vec::new();
        // SAFETY: Not guranteed to be successful, please change to crossbeam_utils::scope
```

```
        let v: Vec<&'static tsp::CostLeaf> = unsafe { core::mem::transmute(v) };
        let btree: &'static tsp::CostBTree = unsafe { core::mem::transmute(btree) };
        handles.push(std::thread::spawn(move || {
            let mut bound = Weight::MAX;
            let mut bestset = Vec::new();
            for n in v {
                debug_assert!(n.cover & current == 0, "Error! {} {}", n.cover, current);
                let new_cover = n.cover | current;
                let new_total = n.cost;
                debug_assert!(new_cover.count_ones() > current.count_ones());
                debug_assert!(new_total > 0);
                if new_total >= bound {
                    continue;
                }
                nodeset.push(&n);
                if new_cover == !0 {
                    bestset.clone_from(&nodeset);
                    bound = new_total;
                    println!("Found! {}", bound);
                } else {
                    best_cover(
                        btree,
                        &mut nodeset,
                        new_cover,
                        new_total,
                        &mut bound,
                        &mut bestset,
                    );
                }
                nodeset.pop();
            }
            return (bestset, bound);
        }));
    }

    let mut bestset = Vec::new();
    let mut best = Weight::MAX;
    for h in handles.into_iter() {
        let (p, c) = h.join().unwrap();
        if best <= c {
            continue;
        }
        bestset = p;
        best = c;
    }

    // SAFETY: This is *very* unsafe
    unsafe { (core::mem::transmute(bestset), best) }
}

fn best_cover<'a>(
    btree: &'a tsp::CostBTree,
    nodeset: &mut Vec<&'a tsp::CostLeaf>,
    current: tsp::NodeSet,
    total: Weight,
    bound: &mut Weight,
    best: &mut Vec<&'a tsp::CostLeaf>,
```

```
    ) {
        n_.fetch_add(1, Ordering::Relaxed);
        let mut potential: Vec<_> = btree
            .iter_subset(!current)
            .map(|v| (v, v.cover.count_ones() as usize))
            .collect();
        let n_bound = potential.iter().fold(0, |i, (_, v)| i.max(*v));
        if n_bound > 1 {
            potential.retain(move |(_, v)| *v >= (n_bound - 1));
        }
        if potential.is_empty() {
            return;
        }
        potential.sort_unstable_by_key(|(v, x)| (*x, core::cmp::Reverse(v.cost), v.cover));
        for (n, _) in potential {
            debug_assert!(n.cover & current == 0, "Error! {} {}", n.cover, current);
            let new_cover = n.cover | current;
            let new_total = total + n.cost;
            debug_assert!(new_cover.count_ones() > current.count_ones());
            debug_assert!(new_total > total);
            if new_total >= *bound {
                continue;
            }
            nodeset.push(&n);
            if new_cover == !0 {
                best.clone_from(nodeset);
                *bound = new_total;
                println!("Found! {}", *bound);
            } else {
                best_cover(btree, nodeset, new_cover, new_total, bound, best);
            }
            nodeset.pop();
        }
    }

    let handle = std::thread::spawn(|| {
        while end.load(Ordering::Relaxed) == 0 {
            std::thread::sleep(std::time::Duration::from_secs(2));
            println!("N: {}", n_.load(Ordering::Relaxed));
        }
    });

    let (result, _) = best_cover_threaded(&btree, (!(1 << graph.node_count()) + 1) | (1 << start));

    end.store(1, Ordering::Relaxed);
    handle.join().unwrap();

    println!("{:?}", result);

    for n in result {
        for i in n.path.iter() {
            print!("{} -> ", i);
        }
        println!("{}", n.path[0]);
    }
}
```

II. TSP.RS

```rust
use std::collections::{HashMap, VecDeque};

use petgraph::prelude::*;
use petgraph::EdgeType;

use crate::btree::{BTree, IntBTree8};
use crate::Weight;

pub type NodeIndex = usize;
pub type NodeSet = u32;

#[derive(Clone, Copy, PartialEq, Eq, Hash)]
pub struct EntryKey(pub NodeIndex, pub NodeSet);

pub type G_ = HashMap<EntryKey, Weight>;
pub type P_ = HashMap<EntryKey, NodeIndex>;

#[derive(Debug)]
pub struct CostLeaf {
    pub cover: NodeSet,
    pub cost: Weight,
    pub path: Vec<NodeIndex>,
}
pub type CostBTree = IntBTree8<CostLeaf>;

pub fn mtsp<N, Ty: EdgeType>(
    graph: &Graph<N, Weight, Ty, NodeIndex>,
    start: NodeIndex,
    cap: Weight,
) -> (G_, P_, CostBTree) {
    let ncount = graph.node_count();
    assert!(ncount < 32, "Can\'t support too many nodes");

    let mut g = G_::new();
    let mut p = P_::new();
    let mut btree = CostBTree::default();
    let mut queue = VecDeque::new();

    for e in graph.edges_directed(start.into(), Direction::Outgoing) {
        let ix = e.target().index();
        debug_assert_ne!(ix, start);
        let weight = *e.weight();
        g.insert(EntryKey(ix, 0), weight);
        queue.push_back(EntryKey(ix, 0));
    }

    while let Some(EntryKey(i, s)) = queue.pop_front() {
        debug_assert_ne!(i, start);
        debug_assert_eq!((1 << start) as NodeSet & s, 0);
        let s_ = s | (1 << i) as NodeSet;
        let c = g[&EntryKey(i, s)];
        for e in graph.edges_directed(i.into(), Direction::Outgoing) {
            let ix = e.target().index();
            if (1 << ix) as NodeSet & s_ != 0 {
                continue;
            }
```

```
            let cost = c + *e.weight();
            if cost > cap {
                continue;
            }

            let k = EntryKey(ix, s_);
            if let Some(&v) = g.get(&k) {
                if v <= cost {
                    continue;
                }
            } else if ix != start {
                queue.push_back(k);
            } else {
                let mut vec = vec![0; (s_.count_ones() + 1) as _];
                vec[0] = start;
                vec[1] = i;
                let mut s = s_ & !(1 << i);
                for i in 1..vec.len() - 1 {
                    debug_assert_ne!(s, 0);
                    let x = vec[i];
                    s &= !(1 << x);
                    vec[i + 1] = p[&EntryKey(x, s)];
                }
                assert!(matches!(
                    btree.add_leaf(
                        &s_,
                        CostLeaf {
                            cover: s_,
                            cost: cost,
                            path: vec,
                        }
                    ),
                    Ok(())
                ));
            }
            g.insert(k, cost);
            p.insert(k, i);
        }
    }

    (g, p, btree)
}
```

I. BTREE.RS

```rust
pub trait BTree {
    type Key;
    type Leaf: Sized;

    fn num_children(&self) -> usize;
    fn num_leaf(&self) -> usize;
    fn get_leaf(&self, key: &Self::Key) -> Option<&Self::Leaf>;
    fn get_leaf_mut(&mut self, key: &Self::Key) -> Option<&mut Self::Leaf>;
    fn add_leaf(&mut self, key: &Self::Key, leaf: Self::Leaf) -> Result<(), Self::Leaf>;
    fn delete_leaf(&mut self, key: &Self::Key) -> Option<Self::Leaf>;
}

pub type IntBTreeKey = u32;
const BTREE_BITS: usize = 4usize;
pub const BTREE_BRANCH: usize = 1 << BTREE_BITS;
const BTREE_MASK: usize = BTREE_BRANCH - 1;

#[derive(Default, Clone, PartialEq, Eq)]
pub struct IntBTree<C> {
    child: [Option<Box<C>>; BTREE_BRANCH],
}

impl<L, C> BTree for IntBTree<C>
where
    C: Default + BTree<Key = IntBTreeKey, Leaf = L>,
{
    type Key = IntBTreeKey;
    type Leaf = L;

    fn num_children(&self) -> usize {
        self.child
            .iter()
            .fold(0, |v, i| if i.is_some() { v + 1 } else { v })
    }
    fn num_leaf(&self) -> usize {
        self.child
            .iter()
            .fold(0, |v, i| i.as_deref().map_or(v, |x| v + x.num_leaf()))
    }
    fn get_leaf(&self, key: &Self::Key) -> Option<&Self::Leaf> {
        if let Some(child) = self.get_child(key) {
            child.get_leaf(&((key >> BTREE_BITS) as _))
        } else {
            None
        }
    }
    fn get_leaf_mut(&mut self, key: &Self::Key) -> Option<&mut Self::Leaf> {
        if let Some(child) = self.get_child_mut(key) {
            child.get_leaf_mut(&((key >> BTREE_BITS) as _))
        } else {
            None
        }
    }
    fn add_leaf(&mut self, key: &Self::Key, leaf: Self::Leaf) -> Result<(), Self::Leaf> {
        let i = (*key as usize) & BTREE_MASK;
        let ci = *key >> BTREE_BITS;
```

```
        if let Some(child) = self.get_child_mut(key) {
            child.add_leaf(&ci, leaf)
        } else {
            let child = self.child[i].get_or_insert_with(Default::default);
            let ret = child.add_leaf(&ci, leaf);
            if ret.is_err() {
                self.child[i].take();
            }
            ret
        }
    }
    fn delete_leaf(&mut self, key: &Self::Key) -> Option<Self::Leaf> {
        let i = (*key as usize) & BTREE_MASK;
        let ci = *key >> BTREE_BITS;
        if let Some(child) = self.get_child_mut(key) {
            let ret = child.delete_leaf(&ci);
            if child.num_children() == 0 {
                self.child[i].take();
            }
            ret
        } else {
            None
        }
    }
}

impl<C> IntBTree<C>
where
    Self: BTree<Key = IntBTreeKey>,
{
    fn get_child(&self, key: &IntBTreeKey) -> Option<&C> {
        self.child[(*key as usize) & BTREE_MASK].as_deref()
    }
    fn get_child_mut(&mut self, key: &IntBTreeKey) -> Option<&mut C> {
        self.child[(*key as usize) & BTREE_MASK].as_deref_mut()
    }

    pub fn child_subset(&self, key: IntBTreeKey) -> impl Iterator<Item = &C> {
        let mask = !key as usize;
        self.child
            .iter()
            .enumerate()
            .filter_map(move |(i, v)| if i & mask == 0 { v.as_deref() } else { None })
    }
    pub fn child_superset(&self, key: IntBTreeKey) -> impl Iterator<Item = &C> {
        let mask = key as usize;
        self.child
            .iter()
            .enumerate()
            .filter_map(move |(i, v)| if i & mask == mask { v.as_deref() } else { None })
    }
}

#[derive(Clone, PartialEq, Eq)]
pub struct IntBTreeEnd<L> {
    child: [Option<Box<L>>; BTREE_BRANCH],
}
```

```rust
impl<L> Default for IntBTreeEnd<L> {
    fn default() -> Self {
        // SAFETY: None value for Option<Box<_>> is zero
        unsafe { core::mem::zeroed() }
    }
}

impl<L> BTree for IntBTreeEnd<L> {
    type Key = IntBTreeKey;
    type Leaf = L;

    fn num_children(&self) -> usize {
        self.num_leaf()
    }
    fn num_leaf(&self) -> usize {
        self.child
            .iter()
            .fold(0, |v, i| if i.is_some() { v + 1 } else { v })
    }
    fn get_leaf(&self, key: &Self::Key) -> Option<&Self::Leaf> {
        self.child[(*key as usize) & BTREE_MASK].as_deref()
    }
    fn get_leaf_mut(&mut self, key: &Self::Key) -> Option<&mut Self::Leaf> {
        self.child[(*key as usize) & BTREE_MASK].as_deref_mut()
    }
    fn add_leaf(&mut self, key: &Self::Key, leaf: Self::Leaf) -> Result<(), Self::Leaf> {
        let i = (*key as usize) & BTREE_MASK;
        if self.get_leaf_mut(key).is_some() {
            Err(leaf)
        } else {
            self.child[i].get_or_insert(Box::new(leaf));
            Ok(())
        }
    }
    fn delete_leaf(&mut self, key: &Self::Key) -> Option<Self::Leaf> {
        let i = (*key as usize) & BTREE_MASK;
        if let Some(v) = self.child[i].take() {
            Some(unsafe { core::ptr::read(&*v as *const _) })
        } else {
            None
        }
    }
}

impl<C> IntBTreeEnd<C> {
    pub fn child_subset(&self, key: IntBTreeKey) -> impl Iterator<Item = &C> {
        let mask = !key as usize;
        self.child
            .iter()
            .enumerate()
            .filter_map(move |(i, v)| if i & mask == 0 { v.as_deref() } else { None })
    }
    pub fn child_superset(&self, key: IntBTreeKey) -> impl Iterator<Item = &C> {
        let mask = key as usize;
        self.child
            .iter()
            .enumerate()
            .filter_map(move |(i, v)| if i & mask == mask { v.as_deref() } else { None })
```

```
    }
}

pub type IntBTree8<L> =
    IntBTree<IntBTree<IntBTree<IntBTree<IntBTree<IntBTree<IntBTree<IntBTreeEnd<L>>>>>>>>;

impl<L> IntBTree8<L>
where
    Self: BTree<Key = IntBTreeKey, Leaf = L>,
{
    pub fn iter_leaf(&self) -> impl Iterator<Item = &L> {
        self.child
            .iter()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref().map(|v| &v.child))
            .flatten()
            .filter_map(|v| v.as_deref())
    }
    pub fn iter_subset(&self, key: IntBTreeKey) -> impl Iterator<Item = &L> {
        let key = key as usize;
        let k8 = ((key >> (BTREE_BITS * 7)) & BTREE_MASK) as _;
        let k7 = ((key >> (BTREE_BITS * 6)) & BTREE_MASK) as _;
        let k6 = ((key >> (BTREE_BITS * 5)) & BTREE_MASK) as _;
        let k5 = ((key >> (BTREE_BITS * 4)) & BTREE_MASK) as _;
        let k4 = ((key >> (BTREE_BITS * 3)) & BTREE_MASK) as _;
        let k3 = ((key >> (BTREE_BITS * 2)) & BTREE_MASK) as _;
        let k2 = ((key >> (BTREE_BITS * 1)) & BTREE_MASK) as _;
        let k1 = (key & BTREE_MASK) as _;
        self.child_subset(k1).flat_map(move |v| {
            v.child_subset(k2).flat_map(move |v| {
                v.child_subset(k3).flat_map(move |v| {
                    v.child_subset(k4).flat_map(move |v| {
                        v.child_subset(k5).flat_map(move |v| {
                            v.child_subset(k6).flat_map(move |v| {
                                v.child_subset(k7).flat_map(move |v| v.child_subset(k8))
                            })
                        })
                    })
                })
            })
        })
    }
    pub fn iter_superset(&self, key: IntBTreeKey) -> impl Iterator<Item = &L> {
        let key = key as usize;
        let k8 = ((key >> (BTREE_BITS * 7)) & BTREE_MASK) as _;
        let k7 = ((key >> (BTREE_BITS * 6)) & BTREE_MASK) as _;
        let k6 = ((key >> (BTREE_BITS * 5)) & BTREE_MASK) as _;
```

```
        let k5 = ((key >> (BTREE_BITS * 4)) & BTREE_MASK) as _;
        let k4 = ((key >> (BTREE_BITS * 3)) & BTREE_MASK) as _;
        let k3 = ((key >> (BTREE_BITS * 2)) & BTREE_MASK) as _;
        let k2 = ((key >> (BTREE_BITS * 1)) & BTREE_MASK) as _;
        let k1 = (key & BTREE_MASK) as _;
        self.child_superset(k1).flat_map(move |v| {
            v.child_superset(k2).flat_map(move |v| {
                v.child_superset(k3).flat_map(move |v| {
                    v.child_superset(k4).flat_map(move |v| {
                        v.child_superset(k5).flat_map(move |v| {
                            v.child_superset(k6).flat_map(move |v| {
                                v.child_superset(k7).flat_map(move |v| v.child_superset(k8))
                            })
                        })
                    })
                })
            })
        })
    }
}
```

APENDIKS B
File Input (*JARAK PUSKESMAS.csv*)

```
NAMA;DINAS KESEHATAN;Puskesmas Alun-alun;Puskesmas Balongpanggang;Puskesmas Benjeng;Puskesmas
Bungah;Puskesmas Cerme;Puskesmas Dadap Kuning;Puskesmas Dapet;Puskesmas Driyorejo;Puskesmas Duduk
Sampeyan;Puskesmas Dukun;Puskesmas Gending;Puskesmas Industri;Puskesmas Karangandong;Puskesmas
Kebomas;Puskesmas Kedamean;Puskesmas Kepatihan;Puskesmas Kesambenkulon;Puskesmas Manyar;Puskesmas
Menganti;Puskesmas Mentaras;Puskesmas Metatu;Puskesmas Nelayan;Puskesmas Panceng;Puskesmas Sekapuk;Puskesmas
Sembayat;Puskesmas Sidayu;Puskesmas Slempit;Puskesmas Sukomulyo;Puskesmas Ujungpangkah;Puskesmas Wringin
Anom
DINAS KESEHATAN;;13;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Alun-alun;13;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Balongpanggang;40;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Benjeng;32;;10;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Bungah;26;;;;;;;;;;;;;;;;;;16;;;;;;;;;;;;
Puskesmas Cerme;19;;24;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Dadap Kuning;29;;;15;;11;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Dapet;47;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Driyorejo;51;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;20
Puskesmas Duduk Sampeyan;14;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Dukun;44;;;;;;;;;;;;;;;;;;;;;;;;;16;;;;
Puskesmas Gending;14;6;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Industri;11;6;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Karangandong;44;;;;;;;;11;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Kebomas;10;7;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Kedamean;38;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Kepatihan;35;;;;;;;;;;;;;;;;;;14;;;;;;;;;;;
Puskesmas Kesambenkulon;41;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;18
Puskesmas Manyar;10;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Menganti;40;;;;;;;;;;;;;20;;;;;;;;;;;;;;;;
Puskesmas Mentaras;58;;;;;;;;;;;16;;;;;;;;;;;;;;18;;;24;;;;
Puskesmas Metatu;32;;;7;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Nelayan;16;5;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Panceng;55;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Sekapuk;47;;;;;;;;;;;;;;;;;;;;;;;;10;;;;;;15;
Puskesmas Sembayat;22;;;;;;;;;;;;;;;;;;12;;;;;;;;;;;;
Puskesmas Sidayu;33;;;;8;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Slempit;25;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Sukomulyo;10;15;;;;;;;;;;;;;;;;;;14;;;;;;;;;;;;
Puskesmas Ujungpangkah;53;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Puskesmas Wringin Anom;60;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```