PROJECT DESAIN ANALISIS
DAN ALGORITMA

2023B SAINS DATA UNESA

MINIMUM SPANNING TREE

DALAM

OPTIMISASI KABEL JARINGAN INTERNET ANTAR SEKOLAH MENENGAH AKHIR (SMA) DI 3 KOTA INDONESIA

MENGGUNAKAN

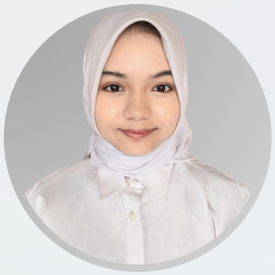
PRIM, KRUSKAL, DAN BORUVKA

Presentated by 2023B TEAM 7

Disusun Oleh

Ibrahim Frosly Alesandro

23031554021



Alisha Deana Tabina

23031554244

M. Rarendra Satiya

23031554117

Kontribusi

Ibrahim Frosly Alesandro

1. Mengekseskusi 3 kota dengan algoritma Kruskal
2. Visualisasi di streamlit
3. Membuat code Running Time

Alisha Deana Tabina

1. Mengekseskusi 3 kota dengan algoritma Prim
2. Membuat code untuk menemukan penggunaan memori setiap algoritma
3. Membuat laporan

M. Rarendra Satiya

1. Mengekseskusi 3 kota dengan algoritma Boruvka
2. Membuat code untuk menemukan jarak antar sekolah dan total jarak setelah MST
3. Finishing Code

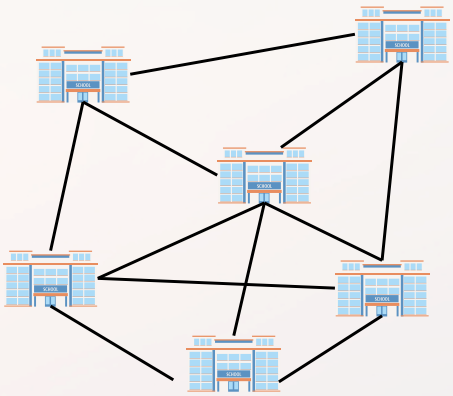
LATAR BELAKANG

Kebutuhan Jaringan Internet Antar Sekolah

Di era digital saat ini, internet telah menjadi hal penting untuk mendukung pendidikan. Setiap sekolah, terutama sekolah menengah, harus memiliki akses internet yang memadai untuk mendukung pembelajaran berbasis teknologi, mendapatkan akses ke informasi publik, dan melaksanakan berbagai program pendidikan yang berbasis internet. Namun, keterbatasan infrastruktur seringkali menjadi hambatan, terutama dalam menghubungkan sekolah-sekolah yang tersebar di berbagai tempat dengan jaringan internet yang kuat. Tantangan Utama dalam mendukung pendidikan berbasis digital adalah rendahnya kualitas koneksi. Langkah yang harus diambil kedepannya adalah dengan memastikan jaringan antar sekolah dapat dibangun dengan biaya instalasi kabel yang serendah mungkin tanpa mengurangi kualitas koneksi yang dihasilkan.

Minimum Spanning Tree (MST) sebagai Solusi Efisiensi Biaya

Membangun jaringan kabel antara sekolah dapat digambarkan sebagai masalah graf. Di dalam graf, setiap sekolah diwakili oleh titik (verteks/node), dan koneksi antar sekolah diwakili oleh sisi (edge), dengan jarak fisik antara lokasi diambil sebagai bobot. Untuk masalah seperti ini, metode yang paling sesuai untuk menemukan jalur koneksi dengan biaya minimum adalah Minimum Spanning Tree (MST). Minimum Spanning Tree (MST) adalah konsep dalam teori graf yang bertujuan untuk menghubungkan semua simpul (nodes) dalam sebuah graf berbobot dengan total bobot sisi (edges) yang paling minimal, tanpa membentuk siklus. Tiga algoritma yang digunakan untuk menemukan MST dalam optimisasi kabel jaringan internet adalah **Prim, Kruskal**, dan **Boruvka.**



Tujuan Penelitian

- Menentukan jaringan kabel dengan biaya minimum antar sekolah SMA di 3 kota.
- Membandingkan efektivitas algoritma Prim, Kruskal, dan Boruvka dalam penerapan MST.

Ruang Lingkup

- Data yang digunakan: Sekolah SMA di tiga kota (Wonosobo, Surabaya, dan Sidoarjo)
- Parameter jarak dihitung menggunakan koordinat geografis (latitude dan longitude).
- Tidak mempertimbangkan kondisi medan atau faktor non-geografis lainnya.

LANDASAN TEORI

Penerapan Algoritma MST untuk Optimasi Jaringan Internet Antar Sekolah

Masing-masing dari ketiga algoritma ini memiliki keunggulan dalam menemukan jalur minimum untuk menghubungkan semua titik dengan efisiensi tinggi.

Algoritma PRIM

Algoritma Prim adalah metode yang digunakan untuk menentukan pohon rentang minimum dalam grafik, secara efektif menemukan jalur terdekat untuk mengangkut barang. Kompleksitasnya adalah O (v^2), di mana n mewakili jumlah verteks dalam grafik. Algoritma ini sangat cocok untuk graf yang padat (dense) karena efisiensi dalam memilih sisi terdekat setiap kali dilakukan.

Algoritma Kruskal

Algoritma Kruskal menemukan pohon rentang minimum dalam grafik dengan mengurutkan bobot sisi dan menghubungkan simpul tanpa membentuk siklus. Kompleksitas waktunya adalah O (E log E), di mana E adalah jumlah sisi, Jumlah sis dihitung dengan rumus $e = n (n-1) / 2$. Algoritma ini cocok untuk graf jarang (Sparse Graph).

Algoritma BORUVKA

Algoritma Boruvka adalah metode untuk menemukan pohon merentang minimum dalam graf yang terhubung dengan menambahkan sisi dengan bobot terendah secara iteratif. Kompleksitas waktu algoritma ini adalah O(E log V), di mana E adalah jumlah sisi dan V adalah jumlah simpul dalam graf. .

	PRIM	KRUSKAL	BORUVKA
Pendekatan	Lokal (mulai dari satu sisi dengan jarak terpendek)	Global (menjelajahi semua sisi dengan menurutkan jarak terpendek)	Gabungan dari kelompok kelompok node
Kelebihan	Efisien untuk graf padat	Efisien untuk graf jarang	Cocok untuk graf besar
Kekurangan	Tidak efisien untuk graf jarang	Memerlukan pengurutan jarak awal	Implementasi jadi lebih kompleks
Kompleksitas Waktu	$O(V^2)$ atau $O(E + V \log V)$	$O(E \log E)$ atau $O(E \log V)$	$O(E \log V)$
Kompleksitas Ruang	$O(V + E)$ atau $O(V^2)$	$O(E)$	$O(V+E)$

Rumus Haversine

Rumus Haversine adalah metode untuk menghitung jarak antara dua titik di permukaan bumi menggunakan latitude dan longitude. Metode ini mengasumsikan bumi berbentuk bulat sempurna, sehingga mengabaikan ketinggian dan kedalaman, namun tetap akurat untuk sebagian besar perhitungan.

```python
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Radius bumi dalam kilometer
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    return R * c
```

Haversine Distance

- $d = 2r\ arcsine\left(\sqrt{sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + cos\varphi_1 . cos\varphi_2 . sin^2\left(\frac{\omega_2-\omega_1}{2}\right)}\right)$
- Where
  - φ_1, φ_2 are the latitude of point 1 and point 2 in radian form.
  - ω_1, ω_2 are the longitude of point 1 and point 2 in radian form.

Erdiansyah, Erdiansyah., O., Komarudin., Iqbal, Maulana. (2024). Implementasi rumus haversine dan algoritma dijkstra dalam pencarian rute perjalanan menggunakan krl pada aplikasi berbasis android. JATI (Jurnal Mahasiswa Teknik Informatika), doi: 10.36040/jati.v7i4.7258

METODOLOGI DAN HASIL ANALISIS

Data

Menggunakan data sekolah Indonesia yang ditemukan melalui platfrom dataset kaggle dengan link https://www.kaggle.com/datasets/greegtitan/school-dataset-idn.

Data diimpor menggunakan pustaka pandas dan difilter berdasarkan kriteria tertentu untuk mendapatkan subset data yang relevan. Proses pemfilteran dilakukan untuk memilih sekolah pada tingkat SMA di Kota Wonosobo, Surabaya, dan Sidoarjo.

Prepocessing Data

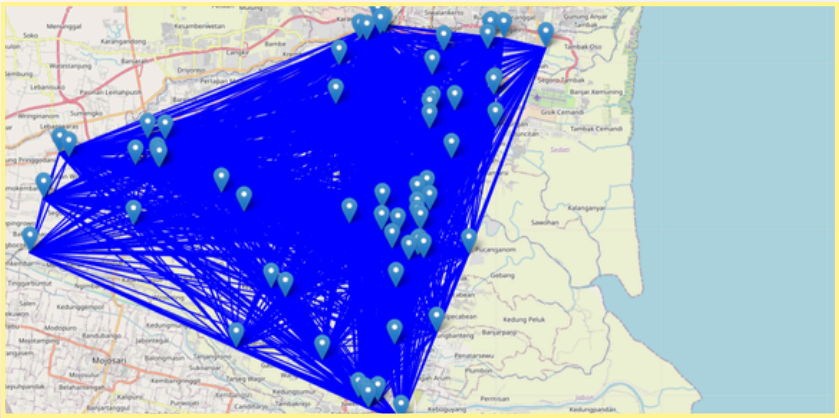
Setelah data terfilter, langkah selanjutnya adalah membersihkan data untuk memastikan keakuratan analisis. Baris data dengan nilai NaN pada kolom koordinat latitude (lat) dan longitude (long) dihapus menggunakan fungsi dropna. Selain itu, dilakukan penyaringan tambahan untuk menghindari penggunaan data dengan koordinat yang tidak valid (yaitu, nilai 0.0 pada kolom lat atau long). Hasil dari langkah ini adalah dataset yang bersih (nama kota_clean) dan siap digunakan untuk langkah analisis atau visualisasi selanjutnya.

```
sby = data[(data['province_name'] == 'Jawa Timur') & (data['stage'] == 'SMA') & (data['city_name'] == 'Kota Surabaya')]

# Menghapus baris dengan NaN pada kolom lat/long dan menghindari koordinat yang tidak valid (0.0)
sby_clean = sby.dropna(subset=['lat', 'long'])
sby_clean = sby_clean[(sby_clean['lat'] != 0.0) & (sby_clean['long'] != 0.0)]
```

Implementasi Algoritma

- Setiap sekolah sebagai simpul/verteks.
- Jarak antar sekolah sebagai bobot sisi (edge).

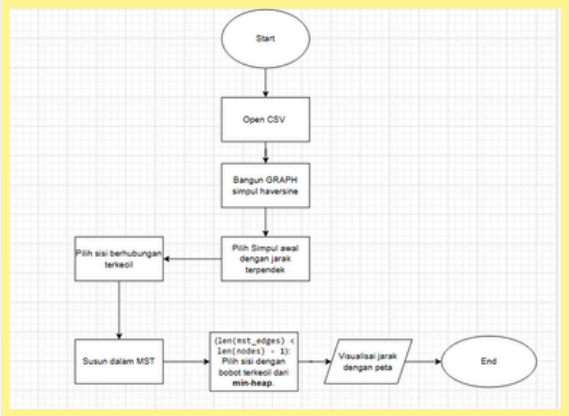
Langkah Awal sebelum masuk pada implementasi algoritma, kami menghubungkan satu sekolah dengan seluruh sekolah lainnya yang ada dalam kota tersebut, sehingga terjadi Complete Graph seperti gambar dibawah ini.



PRIM

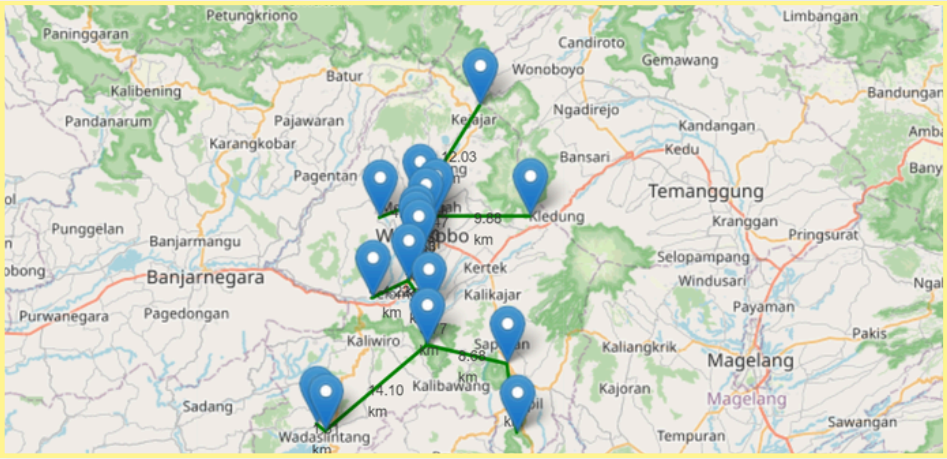
Cara kerja algoritma PRIM dalam masalah ini adalah

1. Inisialisasi sekolah awal yang memiliki jarak paling minimum dengan sekolah didekatnya
2. Setelah menemukan 2 sekolah dengan jarak minimum, tambahkan sekolah tersebut kedalam MST
3. Ulangi langkah 1 dan 2 hingga semua sekolah terhubung tapi tidak ada kabel jaringan yang ganda.




```python
# Inisialisasi komponen
mst_edges = []
visited = {node: False for node in nodes}  # Menandai apakah simpul sudah dikunjungi
min_heap = []  # Priority queue (min-heap) untuk memilih sisi terkecil

# Mulai dari simpul pertama (misalnya node pertama dalam list)
start_node = nodes[0]
visited[start_node] = True

# Masukkan semua sisi yang terhubung ke start_node ke dalam heap
for weight, node1, node2 in edges:
    if node1 == start_node or node2 == start_node:
        heapq.heappush(min_heap, (weight, node1, node2))

# Algoritma Prim: Menghubungkan semua simpul satu per satu
while len(mst_edges) < len(nodes) - 1 and min_heap:
    # Pilih sisi dengan bobot terkecil dari heap
    weight, node1, node2 = heapq.heappop(min_heap)

    # Pastikan kita hanya menambahkan sisi yang menghubungkan simpul yang belum dikunjungi
    if visited[node1] and not visited[node2]:
        visited[node2] = True
        mst_edges.append((node1, node2, weight))
        # Tambahkan semua sisi yang menghubungkan node2 ke heap
        for w, n1, n2 in edges:
            if (n1 == node2 and not visited[n2]) or (n2 == node2 and not visited[n1]):
                heapq.heappush(min_heap, (w, n1, n2))
    elif visited[node2] and not visited[node1]:
        visited[node1] = True
        mst_edges.append((node1, node2, weight))
        # Tambahkan semua sisi yang menghubungkan node1 ke heap
        for w, n1, n2 in edges:
            if (n1 == node1 and not visited[n2]) or (n2 == node1 and not visited[n1]):
                heapq.heappush(min_heap, (w, n1, n2))
end_time_prim = time.time()
# Tampilkan hasil di peta
```

PRIM Wonosobo



Jarak total dari semua sisi sebelum MST Wonosobo adalah 1904.78 km

Kompleksitas Ruang (dalam byte)

- Graph (Adjacency List): 0.03 KB
- Visited Dictionary: 0.62 KB
- Min-Heap: 1.05 KB
- MST Edges: 0.18 KB
- Total Memory: 1.89 KB

Waktu Eksekusi / Running Time

- Pembuatan edges: 0.1772 detik
- Running Time Prim Wonosobo: 0.00001 detik

Kompleksitas Teoretis

- Jumlah node: 17
- Jumlah edge: 136

Jarak total dari semua jalur di MST Wonosobo adalah 79.67 km

PRIM Surabaya

Jarak total dari semua sisi sebelum MST Surabaya adalah 60057.97 km

Kompleksitas Ruang (dalam byte)

- Graph (Adjacency List): 0.03 KB
- Visited Dictionary: 4.59 KB
- Min-Heap: 65.65 KB
- MST Edges: 1.21 KB
- Total Memory: 71.47 KB

Waktu Eksekusi / Running Time

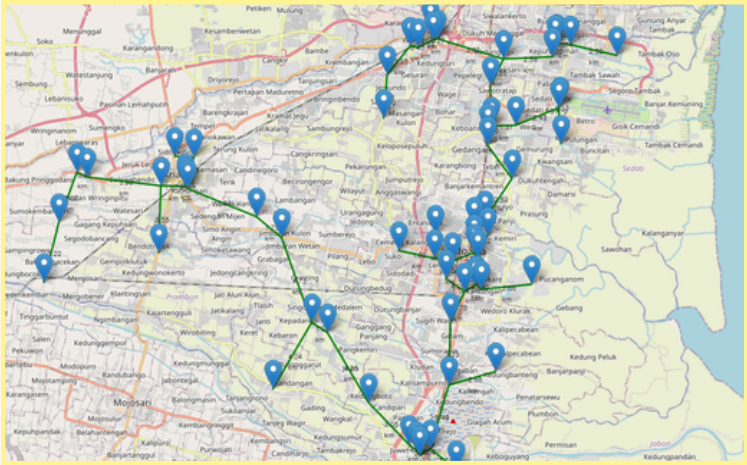
- Pembuatan edges: 3.6481 detik
- Running Time Prim Surabaya: 0.1001 detik

Kompleksitas Teoretis

- Jumlah node: 132
- Jumlah edge: 8646

Jarak total dari semua jalur di MST Surabaya adalah 107.97 km

PRIM Sidoarjo



Jarak total dari semua sisi sebelum MST Sidoarjo adalah 24904.72 km

Kompleksitas Ruang (dalam byte)

- Graph (Adjacency List): 0.03 KB
- Visited Dictionary: 2.22 KB
- Min-Heap: 15.80 KB
- MST Edges: 0.55 KB
- Total Memory: 18.61 KB

Waktu Eksekusi / Running Time

- Pembuatan edges: 0.8862 detik
- Running Time Prim: 0.0326 detik

Kompleksitas Teoretis

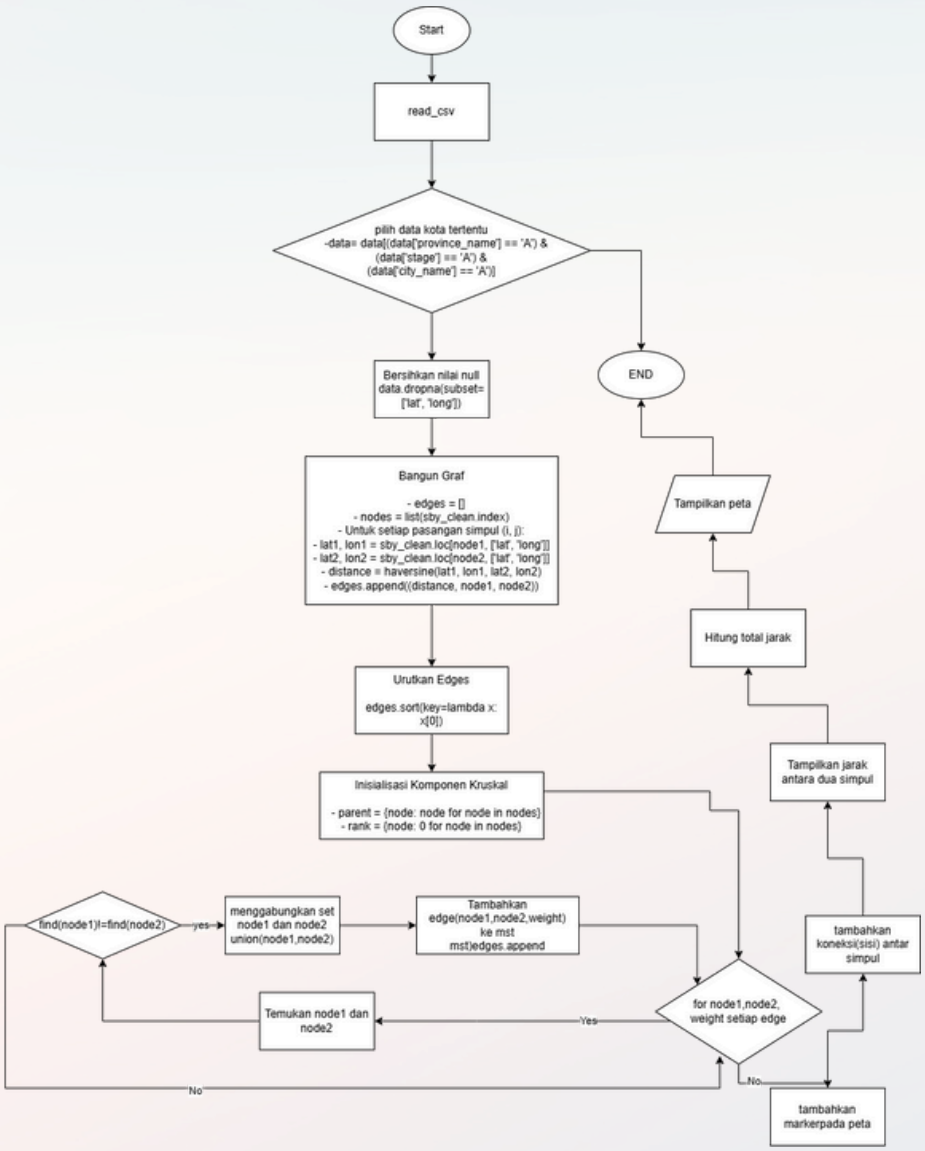
- Jumlah node: 64
- Jumlah edge: 2016

Jarak total dari semua jalur di MST Sidoarjo adalah 101.80 km

Kruskal

Cara kerja algoritma **Kruskal** dalam masalah ini adalah

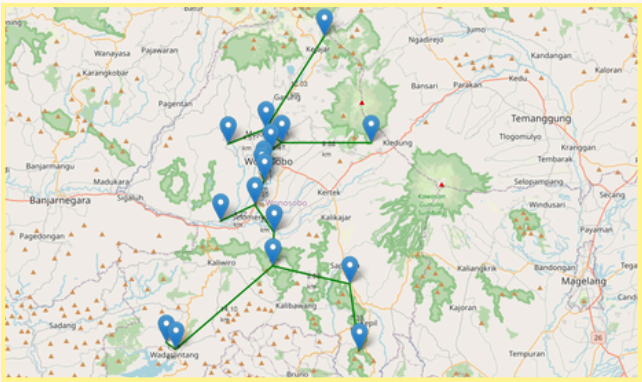
1. Urutkan jarak antar sekolah dari yang paling minimum
2. Mulai inisiasi sekolah awal dengan jarak sisi paling minimum dari hasil pengurutan
3. Lanjut untuk mencari jarak sekolah kedua yang paling minimum dari hasil pengurutan
4. Tambahkan ke MST
5. Lakukan perulangan langkah 2, 3, dan 4 hingga semua sekolah terhubung tanpa ada kabel ganda.



```python
# Bangun graf dengan jarak sebagai bobot
start_time_edges = time.time()
edges = []
nodes = list(wsb_clean.index)
for i in range(len(nodes)):
    for j in range(i + 1, len(nodes)):
        node1, node2 = nodes[i], nodes[j]
        lat1, lon1 = wsb_clean.loc[node1, ['lat', 'long']]
        lat2, lon2 = wsb_clean.loc[node2, ['lat', 'long']]
        distance = haversine(lat1, lon1, lat2, lon2)
        edges.append((distance, node1, node2))
end_time_edges = time.time()

start_time_kruskal = time.time()

edges.sort(key=lambda x: x[0])

# Inisialisasi komponen MST
parent = {node: node for node in nodes}
rank = {node: 0 for node in nodes}

def find(node):
    if parent[node] != node:
        parent[node] = find(parent[node])  # Path compression
    return parent[node]

def union(node1, node2):
    root1, root2 = find(node1), find(node2)
    if root1 != root2:
        if rank[root1] > rank[root2]:
            parent[root2] = root1
        elif rank[root1] < rank[root2]:
            parent[root1] = root2
        else:
            parent[root2] = root1
            rank[root1] += 1
```

```python
# Algoritma Kruskal
mst_edges = []
total_edge_distance = sum(weight for weight, _, _ in edges)
print(f"Jarak total dari semua sisi sebelum MST adalah {total_edge_distance:.2f} km")
for weight, node1, node2 in edges:
    if find(node1) != find(node2):
        union(node1, node2)
        mst_edges.append((node1, node2, weight))
end_time_kruskal = time.time()

# Hitung jarak total dari MST
total_distance = sum(weight for _, _, weight in mst_edges)
```

Kruskal Wonosobo



Jarak total dari semua sisi sebelum MST Wonosobo adalah 1904.78 km

Kompleksitas Ruang (dalam byte)

- Graph (Edge List): 1.21 KB
- Union-Find Structure: 0.62 KB
- MST Edges: 0.18 KB
- Total Memory: 2.02 KB

Waktu Eksekusi / Running Time

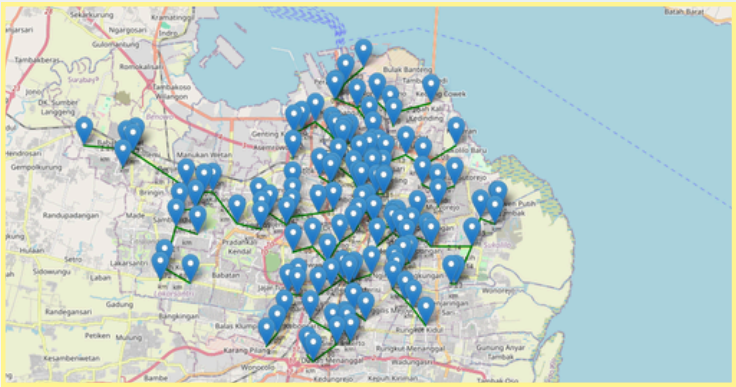
- Pembuatan edges: 0.0624 detik
- Running Time Kruskal: 0.00001 detik

Kompleksitas Teoretis

- Jumlah node: 17
- Jumlah edge: 136

Jarak total dari semua jalur di MST Wonosobo adalah 79.67 km

Kruskal Surabaya



Jarak total dari semua sisi sebelum MST Surabaya adalah 60057.97 km

Kompleksitas Ruang (dalam byte)

- Graph (Edge List): 73.90 KB
- Union-Find Structure: 4.59 KB
- MST Edges: 1.21 KB
- Total Memory: 79.70 KB

Waktu Eksekusi / Running Time

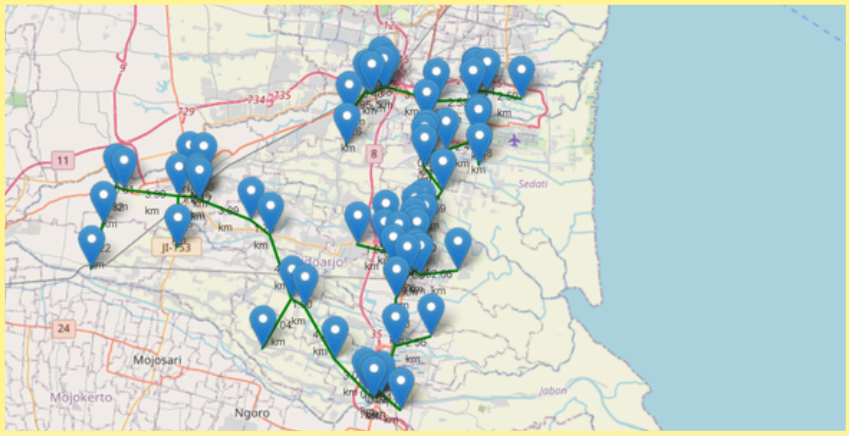
- Pembuatan edges: 3.0498 detik
- Running Time Kruskal: 0.0153 detik

Kompleksitas Teoretis

- Jumlah node: 132
- Jumlah edge: 8646

Jarak total dari semua jalur di MST Surabaya adalah 107.97 km

Kruskal Sidoarjo



Jarak total dari semua sisi sebelum MST adalah 24904.72 km

Kompleksitas Ruang (dalam byte)

- Graph (Edge List): 15.80 KB
- Union-Find Structure: 2.22 KB
- MST Edges: 0.55 KB
- Total Memory: 18.58 KB

Waktu Eksekusi / Running Time

- Pembuatan edges: 0.6642 detik
- Runing time Kruskal: 0.00001 detik

Kompleksitas Teoretis

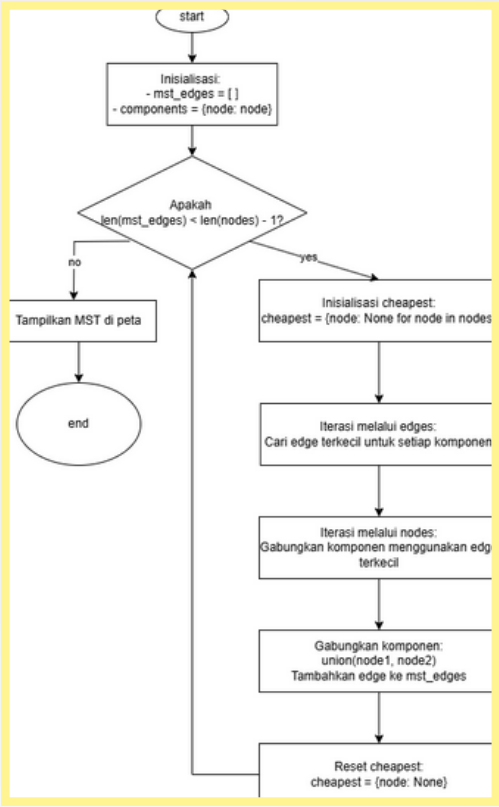
- Jumlah node: 64
- Jumlah edge: 2016

Jarak total dari semua jalur di MST adalah 101.80 km

BORUVKA

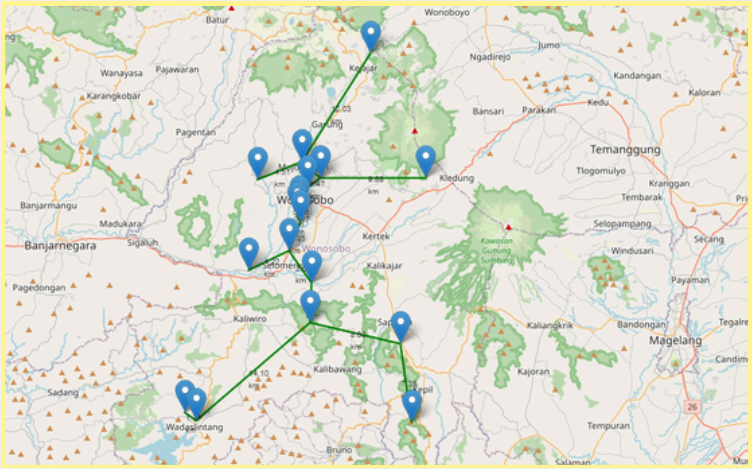
Cara kerja algoritma **BORUVKA** dalam masalah ini adalah

1. Mulai dengan setiap sekolah sebagai Tree terpisah
2. Untuk setiap sekolah pilih jarak terdekat dengan sekolah lainnya
3. Gabungkan sekolah-sekolah tersebut dengan menambahkan jarak terkecil yang ditemukan
4. Ulangi langkah 2 dan 3 hingga semua sekolah terhubung tanpa ada kabel ganda.




```python
# Algoritma Boruvka untuk MST
# Inisialisasi komponen
parent = {node: node for node in nodes}
rank = {node: 0 for node in nodes}

# Fungsi find dan union
def find(node):
    if parent[node] != node:
        parent[node] = find(parent[node])  # Path compression
    return parent[node]

def union(node1, node2):
    root1, root2 = find(node1), find(node2)
    if root1 != root2:
        if rank[root1] > rank[root2]:
            parent[root2] = root1
        elif rank[root1] < rank[root2]:
            parent[root1] = root2
        else:
            parent[root2] = root1
            rank[root1] += 1

# Algoritma Boruvka
mst_edges = []
components = {node: node for node in nodes}  # Komponen awal adalah node itu sendi
# Melakukan Boruvka hingga hanya ada satu komponen
while len(mst_edges) < len(nodes) - 1:
    # Menyimpan sisi terkecil untuk setiap komponen
    cheapest = {node: None for node in nodes}

    # Mencari sisi terkecil untuk setiap komponen
    for weight, node1, node2 in edges:
        root1, root2 = find(node1), find(node2)
        if root1 != root2:
            if cheapest[root1] is None or weight < cheapest[root1][0]:
                cheapest[root1] = (weight, node1, node2)
            if cheapest[root2] is None or weight < cheapest[root2][0]:
                cheapest[root2] = (weight, node1, node2)

    # Gabungkan komponen yang terhubung oleh sisi terkecil
    for node in nodes:
        if cheapest[node] is not None:
            weight, node1, node2 = cheapest[node]
            if find(node1) != find(node2):
                union(node1, node2)
                mst_edges.append((node1, node2, weight))

    # Reset cheapest untuk iterasi berikutnya
    cheapest = {node: None for node in nodes}
end_time_boruvka = time.time()
```

BORUVKA Wonosobo



Jarak total dari semua sisi sebelum MST adalah 1904.78 km

Kompleksitas Ruang (dalam byte)

- Graph (Edge List): 1.21 KB
- Union-Find Structure: 0.62 KB
- Cheapest Array: 0.62 KB
- MST Edges: 0.18 KB
- Total Memory: 2.64 KB

Waktu Eksekusi

- Pembuatan edges: 0.0784 detik
- Running Time Boruvka: 0.0010 detik

Kompleksitas Teoretis

- Jumlah node: 17
- Jumlah edge: 136

Jarak total dari semua jalur di MST adalah 79.67 km

BORUVKA Surabaya

Jarak total dari semua sisi sebelum MST Surabata adalah 60057.97 km

Kompleksitas Ruang (dalam byte)

- Graph (Edge List): 73.90 KB
- Union-Find Structure: 4.59 KB
- Cheapest Array: 4.59 KB
- MST Edges: 1.21 KB
- Total Memory: 84.28 KB

Waktu Eksekusi

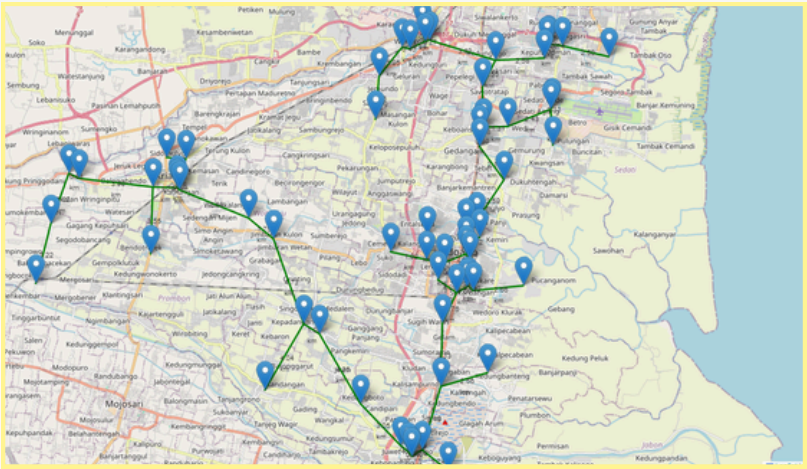
- Pembuatan edges: 3.0737 detik
- Running Time Boruvka: 0.0198 detik

Kompleksitas Teoretis

- Jumlah node: 132
- Jumlah edge: 8646

Jarak total dari semua jalur di MST Surabaya adalah 107.97 km

BORUVKA Sidoarjo



Jarak total dari semua sisi sebelum MST Sidoarjo adalah 24904.72 km

Kompleksitas Ruang (dalam byte) :

- Graph (Edge List): 15.80 KB
- Union-Find Structure: 2.22 KB
- Cheapest Array: 2.22 KB
- MST Edges: 0.55 KB
- Total Memory: 20.80 KB

Waktu Eksekusi / Running Time :

- Pembuatan edges: 1.0015 detik
- Running Time Boruvka 0.0167 detik

Kompleksitas Teoretis
Jumlah node: 64
Jumlah edge: 2016

Jarak total dari semua jalur di MST adalah 101.80 km

PERBANDINGAN ALGORITMA

Wonosobo (17 sekolah SMA)

	PRIM	KRUSKAL	BORUVKA
Jarak total sebelum MST	1904.78 KM	1904.78 KM	1904.78 KM
Jarak total setelah MST	79.67 KM	79.67 KM	79.67 KM
Running Time	0.00001 Detik	0.00001 Detik	0.0010 Detik
Kompleksitas Ruang	1,89 KB	2.02 KB	2.64 KB

Surabaya (132 sekolah SMA)

	PRIM	KRUSKAL	BORUVKA
Jarak total sebelum MST	60057.97 KM	60057.97 KM	60057.97 KM
Jarak total setelah MST	107.97 KM	107.97 KM	107.97 KM
Running Time	0.1001 Detik	0.0153 Detik	0.0198 Detik
Kompleksitas Ruang	71.47 KB	79.70 KB	84.28 KB

Sidoarjo (64 Sekolah SMA)

	PRIM	KRUSKAL	BORUVKA
Jarak total sebelum MST	60057.97 KM	60057.97 KM	60057.97 KM
Jarak total setelah MST	101.8 KM	101.8 KM	101.8 KM
Running Time	0.0326 Detik	0.00001 Detik	0.0167 Detik
Kompleksitas Ruang	18.61 KB	18.58 KB	20.80 KB

Pada kota **Wonosobo** yang memiliki 17 sekolah SMA (vertices), **algoritma Prim menjadi yang terbaik** dengan penggunaan ruang sebesar 1,89 KB dan waktu eksekusi tercepat, yaitu 0,00001 detik. Hal ini terjadi karena jumlah vertex yang kecil membuat adjacency list yang digunakan oleh algoritma Prim sangat efisien dalam mengelola data graf. Selain itu, penggunaan min-heap pada Prim mendukung pencarian sisi minimum dengan cepat, sehingga menghasilkan waktu eksekusi yang optimal.

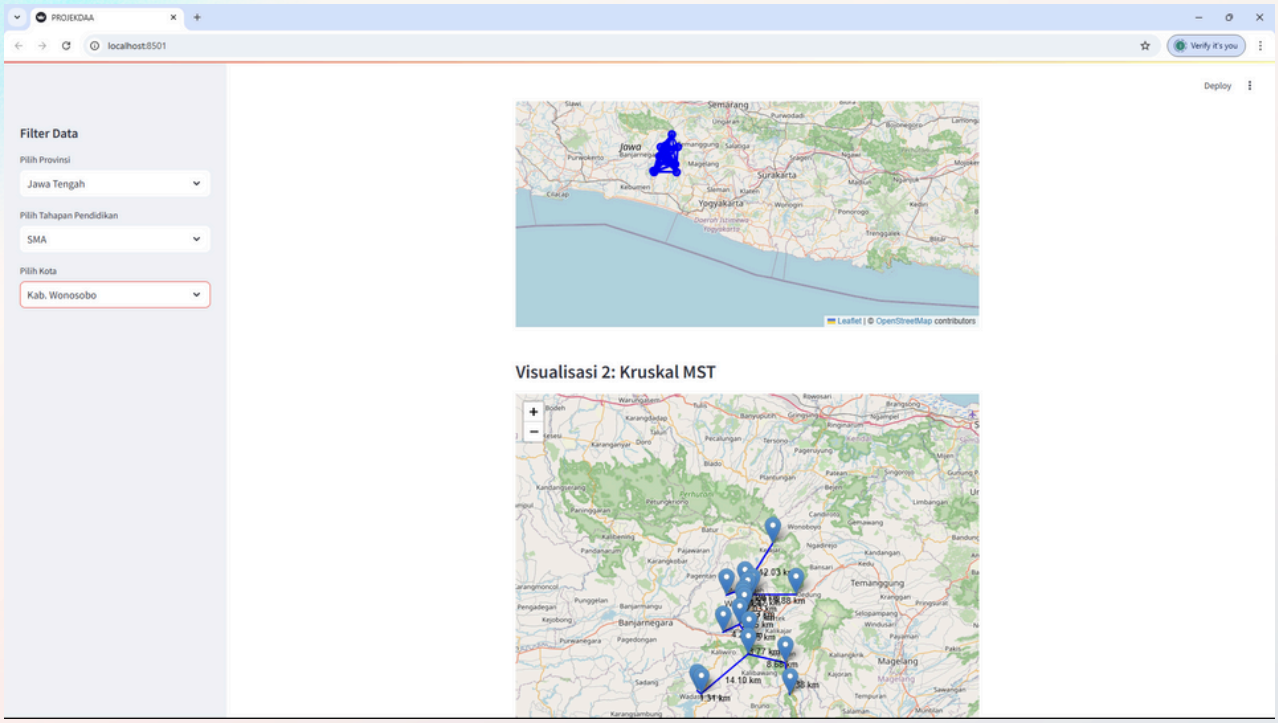
Di **Surabaya**, dengan 132 sekolah SMA (vertices), **algoritma Prim tetap unggul** dari sisi efisiensi ruang, yaitu 71,47 KB, berkat optimalisasi adjacency list dalam menangani graf dengan jumlah vertex besar. Namun, **untuk kecepatan waktu eksekusi, algoritma Kruskal menjadi yang terbaik** dengan waktu 0,0153 detik, meskipun penggunaan ruangnya sedikit lebih besar, yaitu 79,70 KB. Keunggulan Kruskal pada waktu eksekusi disebabkan oleh penggunaan sorting pada daftar sisi (edge list), yang menjadi lebih efisien dibanding traversal heap saat jumlah sisi meningkat signifikan pada graf besar seperti di Surabaya.

Sementara itu, di **Sidoarjo**, dengan 64 sekolah SMA (vertices), **algoritma Kruskal unggul dari sisi penggunaan ruang**, yaitu 18,58 KB, serta memiliki waktu eksekusi tercepat, yaitu 0,00001 detik. Pada ukuran graf menengah seperti ini, daftar sisi yang digunakan oleh Kruskal lebih hemat memori dibanding adjacency list pada Prim. Selain itu, proses sorting pada Kruskal bekerja lebih efisien untuk jumlah sisi yang tidak terlalu besar, menghasilkan waktu eksekusi yang sangat cepat.

VISUALISASI MST SELURUH KOTA DI INDONESIA

Peneliti telah membuat **visualisasi interaktif** menggunakan Streamlit yang memungkinkan pengguna melihat tampilan peta sekolah-sekolah dengan menggunakan tiga algoritma Minimum Spanning Tree (MST): Prim, Kruskal, dan Boruvka. Ketiga algoritma ini ditampilkan dalam satu layar yang bisa digulir, sehingga pengguna dapat membandingkan hasil jaringan yang dihitung oleh masing-masing algoritma. Pengguna hanya perlu memasukkan informasi seperti provinsi, jenjang pendidikan (SD, SMP, SMA), dan kota, untuk melihat jaringan sekolah yang relevan berdasarkan data yang diinputkan.

Dengan visualisasi ini, setiap algoritma akan menunjukkan jaringan sekolah yang optimal, tergantung pada posisi sekolah-sekolah yang ada. Hasilnya ditampilkan pada peta interaktif, memudahkan pengguna untuk melihat hubungan antar sekolah secara lebih jelas. Fitur ini memberikan gambaran yang lebih mudah dipahami tentang cara kerja masing-masing algoritma pada data sekolah, serta memberikan insight mengenai efisiensi jaringan sekolah di wilayah atau jenjang pendidikan yang berbeda.



KESIMPULAN DAN SARAN

- **Wonosobo (17 Sekolah):** Prim unggul karena ukuran graf kecil membuat adjacency list dan heap sangat efisien.

- **Surabaya (132 Sekolah)**: Prim unggul dalam penggunaan ruang pada graf besar, namun Kruskal unggul dari sisi waktu eksekusi karena efisiensi sorting daftar sisi.

- **Sidoarjo (64 Sekolah):** Kruskal unggul dalam penggunaan ruang dan waktu karena daftar sisi lebih sederhana dibanding adjacency list pada graf ukuran sedang.

Secara umum, untuk graf kecil (≤ 20 vertex), Prim adalah pilihan terbaik karena efisiensi ruang dan waktu yang lebih baik. Untuk graf besar (≥ 100 vertex), algoritma Prim unggul dalam efisiensi ruang, sedangkan Kruskal unggul dalam kecepatan. Untuk graf menengah (20–100 vertex), Kruskal lebih direkomendasikan karena penggunaan daftar sisi lebih sederhana dan sorting bekerja optimal pada ukuran menengah. Jika graf sangat padat, penggunaan Prim dengan adjacency matrix dapat dipertimbangkan karena traversal heap lebih efisien untuk banyak koneksi antar vertex. Dengan rekomendasi ini, setiap kota dapat memilih algoritma yang paling sesuai dengan kebutuhan dan karakteristik graf masing-masing.

REFERENCES

- Chen, J. (2023, October 23). The analysis and application of Prim algorithm, Kruskal algorithm, Boruvka algorithm. *EWA PUBLISHING*. https://doi.org/10.54254/2755-2721/19/20231012

- Dili, Y. N., Wulan, E. R., & Ilahi, F. (2021, May). PENYELESAIAN MASALAH TRANSPORTASI UNTUK MENCARI SOLUSI OPTIMAL DENGAN PENDEKATAN MINIMUM SPANNING TREE (MST) MENGGUNAKAN ALGORITMA KRUSKAL DAN ALGORITMA PRIM. *KUBIK: Jurnal Publikasi Ilmiah Matematika*, *6*(1), 44-49. https://doi.org/10.15575/kubik.v6i1.13907

- Ekasaputra, W. C. (2023, July). Visualisasi Algoritma Kruskal dan Prim dalam Mencari Rute Terpendek Berbasis Unity 3D. *Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)*, *7*(3), 459-463. https://doi.org/10.35870/jti k.v7i3.866

- Iqbal, M., Siahaan, A. P. U., Purba, N. E., & Purwanto, D. (2017). Prim's Algorithm for Optimizing Fiber Optic Trajectory Planning. *International Journal of Scientific Research in Science and Technology (www.ijsrst.com)*, *3*(6). https://ijsrst.com/paper/1402.pdf

- N, R. P. (2006, December 28). Perbandingan Algoritma - algoritma Pencarian Minimum Pohon Merentang dari Suatu Graf. *Program Studi Teknik Informatika, Institut Teknologi Bandung Jl. Ganesha 10, Bandung*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2006-2007/Makalah/Makalah0607-112.pdf

- Sanders, P., & Schimek, M. (2023, December 8). Engineering Massively Parallel MST Algorithms. *arXiv*. https://doi.org/10.48550/arXiv.2302.12199