# Implementasi BFS untuk Perencanaan Rute Pipa AC Sentral pada Gedung Bertingkat

Ferdin Arsenarendra Purtadi – 13523117
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: arxenarendra@gmail.com , 13523117@std.stei.itb.ac.id

***Abstrak*—Perencanaan jalur pipa AC sentral dalam gedung bertingkat sering dilakukan secara manual dan rentan terhadap ketidakefisienan. Penelitian ini mengusulkan pendekatan otomatis menggunakan algoritma Breadth-First Search (BFS) untuk menentukan jalur terpendek dari sumber udara ke ruangan tujuan dalam representasi bangunan berbasis matriks tiga dimensi. Setiap elemen bangunan dimodelkan sebagai titik dalam grid yang mempertimbangkan lantai, rintangan, dan ruang kosong. Program ini mampu menghasilkan jalur optimal serta mendeteksi ruangan yang tidak dapat dijangkau. Hasil simulasi menunjukkan efektivitas BFS dalam sistem perencanaan HVAC berbasis komputasi.**

***Kata kunci—BFS, HVAC, Ducting, Gedung Bertingkat, Jalur Optimal, Representasi 3D.***

## I. Latar Belakang

Sistem pendingin udara sentral (AC sentral) telah menjadi bagian penting dalam perancangan bangunan modern, terutama gedung-gedung bertingkat seperti perkantoran, pusat perbelanjaan, rumah sakit, dan kampus. Sistem ini bekerja dengan mengalirkan udara dingin dari satu titik pusat—sering kali berada di lantai dasar atau lantai teknis—ke berbagai ruangan tujuan melalui saluran pipa atau ducting. Di sinilah muncul tantangan utama: bagaimana merancang jalur pipa AC sedemikian rupa sehingga udara bisa menjangkau seluruh ruangan dengan efisien, tanpa boros panjang pipa, tanpa terlalu banyak belokan (yang bisa menurunkan tekanan), dan tentu saja tanpa menabrak struktur bangunan seperti dinding, tangga, atau kolom [1].

Selama ini, proses penentuan jalur pipa AC dalam sistem HVAC (Heating, Ventilation, and Air Conditioning) masih banyak dilakukan secara manual. Seorang perencana HVAC akan menggunakan gambar denah bangunan dan mencoba menghubungkan titik sumber udara ke berbagai ruangan tujuan dengan garis yang mewakili ducting. Sayangnya, proses ini tidak hanya memakan waktu dan rentan kesalahan, tetapi juga sangat subjektif, karena bergantung pada intuisi dan pengalaman teknisi. Akibatnya, banyak desain yang secara visual terlihat rapi, namun pada praktiknya menghasilkan sistem yang boros energi dan sulit dirawat [2].

Untuk mengatasi masalah tersebut, pendekatan berbasis algoritma mulai banyak digunakan dalam simulasi arsitektur dan perancangan sistem bangunan. Salah satu algoritma yang terbukti andal dan sederhana dalam konteks pencarian jalur adalah Breadth-First Search (BFS). BFS merupakan algoritma graf yang bekerja dengan menjelajahi semua kemungkinan lintasan secara bertahap berdasarkan jarak langkah. Kelebihan utama BFS adalah jaminan bahwa ia akan menemukan jalur terpendek (dalam jumlah langkah) jika jalur tersebut memang ada. Hal ini sangat sesuai dengan kebutuhan sistem pipa AC yang ingin meminimalkan panjang ducting tanpa memperhitungkan bobot atau biaya per segmen jalur [3].

Dalam implementasinya, bangunan disimulasikan sebagai sebuah matriks tiga dimensi, dengan dimensi lantai, baris, dan kolom. Setiap sel dalam matriks merepresentasikan kondisi nyata bangunan: jalur kosong (.), rintangan (W untuk wall), sumber udara (S), dan ruangan yang membutuhkan udara dingin (R). Jalur pipa yang dicari adalah dari S ke setiap R, menghindari W dan tetap berada dalam batas bangunan. BFS digunakan untuk mengeksplorasi jalur-jalur ini secara efisien. Karena BFS tidak membutuhkan heuristik seperti A* atau biaya bobot seperti Dijkstra, maka implementasinya relatif lebih stabil dan cocok untuk struktur grid 3D yang besar dan kompleks [4].

Melalui pendekatan ini, perencanaan jalur AC dapat dilakukan secara otomatis, akurat, dan dapat diskalakan untuk gedung dengan banyak lantai dan ruangan. Selain memberikan efisiensi energi, hasil dari proses ini juga bisa digunakan sebagai dasar untuk visualisasi blueprint instalasi pipa AC, sehingga memudahkan kolaborasi antara desainer arsitektur, perencana MEP, dan pelaksana konstruksi.

## II. Dasar Teori

### A. Pencarian Solusi pada Ruang Status

Banyak persoalan komputasi dapat dimodelkan sebagai proses pencarian solusi di dalam sebuah ruang status (*state space*) [6]. Ruang status merupakan himpunan dari semua konfigurasi atau keadaan yang mungkin dari suatu persoalan. Proses pencarian ini dapat divisualisasikan menggunakan pohon ruang status (*state space tree*), di mana [6]:

- Akar (root) merepresentasikan keadaan awal (initial state) dari persoalan.
- Simpul (node) merepresentasikan sebuah status persoalan (problem state) yang layak untuk membentuk solusi.

- Daun (leaf) merupakan status akhir yang menyatakan sebuah solusi telah tercapai (solution/goal state).
- Cabang (branch) merepresentasikan operator atau langkah yang diterapkan untuk transisi dari satu status ke status lainnya.

Solusi dari sebuah persoalan didefinisikan sebagai sebuah lintasan (*path*) dari simpul akar menuju salah satu simpul daun yang merupakan status solusi [6]. Algoritma pencarian bertugas untuk menelusuri pohon ruang status ini secara sistematis untuk menemukan lintasan tersebut [5, 6].

B. Algoritma *Breath-First Search* (BFS)

The *Breadth-First Search* (BFS) adalah salah satu metode pencarian buta (*uninformed/blind search*), yang artinya algoritma ini tidak memiliki informasi tambahan mengenai mana simpul yang lebih menjanjikan untuk menuju solusi [5]. BFS melakukan penelusuran graf secara sistematis dengan mengunjungi simpul-simpul berdasarkan level atau tingkat kedalamannya [6].

Prinsip kerja BFS adalah sebagai berikut [5]:

- Penelusuran dimulai dari simpul awal (akar).
- Algoritma akan mengunjungi semua simpul yang bertetangga secara langsung dengan simpul saat ini.
- Setelah semua simpul pada level tersebut selesai dikunjungi, barulah algoritma melanjutkan penelusuran ke simpul-simpul pada level berikutnya [6].

Untuk mengelola urutan simpul yang akan dikunjungi, BFS menggunakan struktur data antrian (*queue*) dengan mekanisme *First-In, First-Out* (FIFO) [5]. Simpul yang lebih dulu dimasukkan ke dalam antrian akan lebih dulu diproses untuk dieksplorasi tetangganya. Proses penelusuran ini secara visual akan membentuk sebuah Pohon BFS (*BFS Tree*) [5].

C. Analisis Karakteristik Algoritma BFS

Kinerja sebuah algoritma pencarian dapat dievaluasi berdasarkan empat aspek utama [6]. Karakteristik BFS terhadap aspek-aspek tersebut adalah sebagai berikut [6]:

- Kelengkapan (Completeness): Apakah algoritma menjamin penemuan solusi jika memang ada?. BFS bersifat lengkap (complete), dengan syarat faktor percabangan (branching factor) b terbatas.
- Optimalitas (Optimality): Apakah algoritma menjamin penemuan solusi terbaik (misalnya, dengan biaya lintasan terendah). BFS bersifat optimal dengan syarat biaya atau ongkos untuk setiap langkah adalah sama (seragam).
- Kompleksitas Waktu (Time Complexity): Berapa lama waktu yang dibutuhkan untuk mencapai solusi?. Kompleksitas waktu BFS adalah O(bd).
- Kompleksitas Ruang (Space Complexity): Berapa banyak memori yang dibutuhkan selama pencarian?. Kompleksitas ruang BFS adalah O(bd), yang mana merupakan salah satu kelemahan utamanya karena membutuhkan memori yang besar.

Dalam analisis kompleksitas di atas, b adalah faktor percabangan maksimum (jumlah anak terbanyak dari sebuah simpul), dan d adalah kedalaman dari solusi terbaik yang ditemukan [6].

## III. Implementasi

Bab ini merinci aspek teknis dari implementasi sistem pencarian rute pipa AC, mulai dari format data masukan yang digunakan hingga arsitektur kode dan alur eksekusi program. Seluruh logika program diimplementasikan menggunakan bahasa pemrograman Python dan dienkapsulasi dalam kelas ACPathfinder untuk modularitas.

A. Format Data Masukan

Sistem ini menerima masukan berupa sebuah file teks (.txt) yang merepresentasikan denah gedung 3D. Format ini dipilih karena kemudahannya untuk dibuat dan dimodifikasi secara manual.

Struktur masukan adalah sebagai berikut :

- Setiap lantai direpresentasikan sebagai matriks karakter 2D.
- Antar lantai dipisahkan oleh sebuah baris yang hanya berisi tiga tanda hubung (---).
- Setiap karakter dalam matriks merepresentasikan satu sel dalam denah dengan makna sebagai berikut:

  - S: Lokasi unit outdoor AC yang menjadi titik awal (Sumber).

  -R: Lokasi unit indoor yang menjadi titik akhir (Ruangan) tujuan.

  -W: Dinding atau halangan solid yang tidak dapat dilewati.

  -T: Titik akses vertikal (Tangga/Lift) yang memungkinkan perpindahan antar lantai.

  -.: Area kosong yang dapat dilewati.

Contoh masukan dalam bentuk txt (input.txt) :

```
WWWWWWWWWWWWWWWWW

W.T.W.........R.W

W...W.WWWWWWWW.W.W

W.R...W.....W.W.W

W...WWW...W.R.W.W

W.R.W.....W...W.W

W...WWWWWWWW.W.W.W
```

```
W.S.............T.W

WWWWWWWWWWWWWWWWWW

---

WWWWWWWWWWWWWWWWWW

W.T.W.R.......W.W

W...W.WWWWWWW.W.W

W.R.W.W.....W.W.W

W...W.W.R...W.R.W

WWWWWWW.W.W.....W

W.......W.WWWWW.W

W..............T.W

WWWWWWWWWWWWWWWWWW

---

WWWWWWWWWWWWWWWWWW

W.W.W.........R.W

W...W.WWWWWWW.W.W

W.R.W.W R W...W.W

W.....WWWWW...W.W

W.R.............W

WWWWWWWWWWWWWWWW.W

W..............T.W

WWWWWWWWWWWWWWWWWW

---

WWWWWWWWWWWWWWWWWW

W.R..........W.W

W.WWWWWWWWWW.W.W

W............W.W

W.WWWWWWWWWW.W.W

W...R.........W.W

W.WWWWWWWWWW.W.W

W..............T.W

WWWWWWWWWWWWWWWWWW
```

Fungsi read_building_from_file bertanggung jawab untuk membaca file ini, mem-parsing datanya, dan melakukan sanitasi untuk memastikan semua baris dalam satu lantai memiliki panjang yang sama.

```
def read_building_from_file(filepath: str) ->
List[List[List[str]]]:

    building = []
    map_chars = {'S', 'R', 'T', 'W', '.'}
    try:
        with open(filepath, 'r') as f:
            floor_data = []
            for line in f:
                line = line.strip()
                if not line:
                    continue

                if line == '---':
                    if floor_data:

building.append(floor_data)
                    floor_data = []
                    continue

                if ']' in line:
                    line = line.split(']',
1)[-1].strip()
                row = [char for char in
line.replace(" ", "") if char in map_chars]
                if row:
                    floor_data.append(row)

            if floor_data:
                building.append(floor_data)
    except FileNotFoundError:
        print(f"Error: File tidak ditemukan
di '{filepath}'")
        return []
    except Exception as e:
        print(f"Terjadi error saat membaca
file: {e}")
        return []

    if not building:
        return []

    max_cols = 0
    for floor in building:
        for row in floor:
            if len(row) > max_cols:
                max_cols = len(row)
```

```
    sanitized_building = []
    for floor in building:
        sanitized_floor = []
        for row in floor:
            current_len = len(row)
            if current_len < max_cols:
                row.extend(['.'] * (max_cols
- current_len))
            sanitized_floor.append(row)

sanitized_building.append(sanitized_floor)

    return sanitized_building
```

B. Arsitektur dan Modul Program

Seluruh logika sistem dienkapsulasi dalam kelas ACPathfinder untuk modularitas dan kemudahan pengelolaan.

1. **Inisialisasi dan Representasi Data**
   Fungsi konstruktor ini dieksekusi saat objek ACPathfinder dibuat. Tugas utamanya adalah menerima matriks gedung 3D, lalu menginisialisasi properti-properti penting seperti dimensi gedung, serta mengidentifikasi dan menyimpan koordinat dari titik sumber ('S'), semua target ('R'), dan semua titik akses vertikal ('T'). Di dalam fungsi ini pula, didefinisikan kamus self.energy_costs yang menjadi dasar kalkulasi efisiensi energi.

```
class ACPathfinder:
    def __init__(self, building_matrix:
List[List[List[str]]]):
        self.building = building_matrix
        self.floors                    =
len(building_matrix)
        self.rows                      =
len(building_matrix[0])              if
building_matrix else 0
        self.cols                      =
len(building_matrix[0][0])           if
building_matrix  and  building_matrix[0]
else 0
        self.source                    =
self._find_position('S')
        self.targets                   =
self._find_all_positions('R')
        self.stairs                    =
self._find_all_positions('T')

        self.energy_costs = {
            'horizontal': 1.0,
            'vertical_up': 3.0,
            'vertical_down': 2.0,
            'turn': 0.5,
            'base_pressure': 0.1
        }
```

2. **Pencarian Jalur Awal**
   Fungsi ini merupakan implementasi inti dari algoritma Breadth-First Search (BFS) untuk menemukan jalur dengan jumlah langkah minimum dari sumber ke setiap target secara terpisah. Fungsi ini menggunakan antrian (deque) untuk menyimpan status (posisi, path) dan sebuah himpunan (set) visited untuk efisiensi. Pergerakan dieksplorasi secara horizontal di lantai yang sama dan secara vertikal hanya di titik 'T'.

```
def       bfs_pathfinding(self)       ->
List[Dict]:

        if   not   self.source   or   not
self.targets:
            return []

        paths_to_all_targets = []

        for   target_idx,   target   in
enumerate(self.targets):
            queue                      =
deque([(self.source, [self.source])])
            visited = {self.source}

            path_found = False

            while queue:
                current_pos,   path   =
queue.popleft()

                if    current_pos    ==
target:
                    energy_cost       =
self._calculate_energy_cost(path)
                    path_info = {
                        'target_index':
target_idx,

'target_position': target,
                        'path': path,
                        'steps':
len(path),
                        'energy_cost':
energy_cost
                    }

paths_to_all_targets.append(path_info)
                    path_found = True
                    break

                f, r, c = current_pos
```

```
                for dr, dc in self.directions: # Menggunakan arah 2D
                    new_r, new_c = r + dr, c + dc
                    new_pos = (f, new_r, new_c) # Lantai (f) tetap sama

                    if self._is_valid_position(f, new_r, new_c) and new_pos not in visited:

visited.add(new_pos)
                        new_path = path + [new_pos]

queue.append((new_pos, new_path))

                if self.building[f][r][c] == 'T':
                    for new_f in range(self.floors):
                        if new_f != f:
                            if self.building[new_f][r][c] == 'T':
                                new_pos = (new_f, r, c)
                                if new_pos not in visited:

visited.add(new_pos)

new_path = path + [new_pos]

queue.append((new_pos, new_path))

            if not path_found:
                print(f"BFS: Tidak dapat menemukan jalur ke ruangan di {target}")

        return paths_to_all_targets
```

3. **Kalkulasi Biaya Energi**

Setelah sebuah jalur ditemukan, fungsi ini menghitung "skor energi" totalnya. Perhitungan dilakukan dengan mengiterasi setiap segmen pada jalur dan mengakumulasi biaya berdasarkan jenis pergerakan (horizontal, naik/turun lantai, belokan) dan biaya tekanan dasar per segmen.

```
def _calculate_energy_cost(self, path: List[Tuple[int, int, int]]) -> float:

        if len(path) < 2:
            return 0.0

        total_energy = 0.0

        for i in range(1, len(path)):
            prev_pos = path[i-1]
            curr_pos = path[i]

            prev_f, prev_r, prev_c = prev_pos
            curr_f, curr_r, curr_c = curr_pos

            total_energy += self.energy_costs['base_pressure']

            if curr_f != prev_f:

                if curr_f > prev_f:
                    total_energy += self.energy_costs['vertical_up']
                else:
                    total_energy += self.energy_costs['vertical_down']
            else:
                total_energy += self.energy_costs['horizontal']

            if i >= 2:
                prev_prev_pos = path[i-2]
                if self._is_turn(prev_prev_pos, prev_pos, curr_pos):
                    total_energy += self.energy_costs['turn']

        return total_energy
```

4. **Optimasi Jalur**

Tahap ini bertujuan untuk mengurangi total biaya energi dari jalur yang ditemukan oleh BFS. Fungsi ini menerapkan strategi sederhana dengan mencoba meluruskan segmen-segmen jalur, yaitu menghilangkan titik-titik perantara yang tidak perlu jika tidak ada dinding yang menghalangi.

```
def optimize_energy_usage(self, path_info_list: List[Dict]) -> List[Dict]:
        optimized_paths = []

        for path_info in path_info_list:
            original_path = path_info['path']
            original_energy = path_info['energy_cost']

            optimized_path = self._optimize_path_turns(original_path)
            optimized_energy = self._calculate_energy_cost(optimized_path)
```

```
            if optimized_energy < 
original_energy:
                optimized_info = 
path_info.copy()
                optimized_info['path'] = 
optimized_path
                optimized_info['energy_cost'] 
= optimized_energy
                optimized_info['steps'] = 
len(optimized_path)

optimized_info['energy_saved'] = 
original_energy - optimized_energy

optimized_paths.append(optimized_info)
            else:
                path_info['energy_saved'] = 
0.0

optimized_paths.append(path_info)

        return optimized_paths
```

**5. Visualisasi Program**

Untuk menyajikan hasil kepada pengguna, program dilengkapi dengan dua fungsi visualisasi utama.

- create_professional_blueprint: Membangun representasi visual dari denah dan jalur pipa yang sudah dioptimalkan menggunakan canvas berbasis karakter.
- _draw_energy_legend: Menghasilkan tabel ringkasan yang berisi legenda, analisis energi per jalur, dan rincian biaya

C. Alur Eksekusi Program

Blok if __name__ == "__main__": di akhir file berfungsi sebagai titik masuk program. Alurnya adalah sebagai berikut:

1. Program mengelola pengambilan nama file masukan dari pengguna, baik melalui argumen baris perintah maupun input interaktif.
2. Objek ACPathfinder diinisialisasi dengan data denah yang telah dibaca dari file.
3. Secara berurutan, program memanggil metode-metode utama: bfs_pathfinding untuk menemukan jalur, optimize_energy_usage untuk memperbaikinya, print_path_with_energy untuk rincian teks, dan create_professional_blueprint untuk visualisasi akhir

## IV. Analisis dan Pembahasan

Analisis mendalam terhadap hasil eksekusi program ACPathfinder dengan menggunakan skenario pengujian yang kompleks. Pembahasan akan mencakup evaluasi kemampuan program dalam menangani berbagai tantangan, analisis efisiensi rute yang dihasilkan, efektivitas modul optimasi, serta interpretasi dari visualisasi blueprint akhir..

A. Skenario dan Pengujian

Pengujian dilakukan menggunakan denah gedung 4 lantai yang didefinisikan dalam file input.txt. Skenario ini secara spesifik dirancang untuk menguji kapabilitas program dalam situasi yang menantang, antara lain:

- Akses Vertikal Ganda: Terdapat dua jalur tangga/lift ('T') yang terpisah, satu di sisi kiri dan satu di sisi kanan gedung.
- Akses Vertikal Terputus: Jalur tangga di sisi kiri sengaja dibuat terputus dan tidak tersedia di Lantai 3 dan 4, untuk menguji kemampuan program memilih jalur vertikal yang valid.
- Ruangan Terisolasi: Sebuah ruangan ('R') di Lantai 3 ditempatkan dalam posisi terkurung total oleh dinding ('W') untuk menguji penanganan kasus di mana sebuah target tidak dapat dijangkau.

B. Analisis Hasil Eksekusi Program

Berikut adalah hasil log dari program yang menerima input input.txt

```
[INFO] Menjalankan algoritma BFS untuk mencari 
jalur awal...
BFS: Tidak dapat menemukan jalur ke ruangan di (2, 
3, 7)
[INFO] BFS selesai dalam 0.0071 detik.

[INFO] Menjalankan optimasi energi pada jalur yang 
ditemukan...
[INFO] Optimasi selesai dalam 0.0000 detik.

=== JALUR Jalur Optimal ke Ruangan 1 ===
Panjang jalur: 5 langkah
Biaya energi: 5.90 unit
Efisiensi energi: 1.18 unit per langkah
Penghematan energi: 15.40 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 4, Kolom 3 [Ruangan Tujuan]
  3. Lantai 1, Baris 4, Kolom 6
  4. Lantai 1, Baris 2, Kolom 6
  5. Lantai 1, Baris 2, Kolom 15 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 2 ===
Panjang jalur: 2 langkah
Biaya energi: 1.10 unit
Efisiensi energi: 0.55 unit per langkah
Penghematan energi: 3.30 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 4, Kolom 3 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 3 ===
Panjang jalur: 4 langkah
Biaya energi: 4.30 unit
Efisiensi energi: 1.08 unit per langkah
Penghematan energi: 11.00 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 12
```

```
  3. Lantai 1, Baris 5, Kolom 12
  4. Lantai 1, Baris 5, Kolom 13 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 4 ===
Panjang jalur: 2 langkah
Biaya energi: 1.10 unit
Efisiensi energi: 0.55 unit per langkah
Penghematan energi: 1.10 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 6, Kolom 3 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 5 ===
Panjang jalur: 8 langkah
Biaya energi: 12.70 unit
Efisiensi energi: 1.59 unit per langkah
Penghematan energi: 26.40 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 2, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 2, Baris 8, Kolom 16
  5. Lantai 2, Baris 5, Kolom 16
  6. Lantai 2, Baris 5, Kolom 14
  7. Lantai 2, Baris 2, Kolom 14
  8. Lantai 2, Baris 2, Kolom 7 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 6 ===
Panjang jalur: 4 langkah
Biaya energi: 6.30 unit
Efisiensi energi: 1.57 unit per langkah
Penghematan energi: 6.60 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 2, Kolom 3 [Tangga/Lift]
  3. Lantai 2, Baris 2, Kolom 3 [Tangga/Lift]
  4. Lantai 2, Baris 4, Kolom 3 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 7 ===
Panjang jalur: 8 langkah
Biaya energi: 12.70 unit
Efisiensi energi: 1.59 unit per langkah
Penghematan energi: 20.90 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 2, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 2, Baris 8, Kolom 16
  5. Lantai 2, Baris 6, Kolom 16
  6. Lantai 2, Baris 6, Kolom 12
  7. Lantai 2, Baris 5, Kolom 12
  8. Lantai 2, Baris 5, Kolom 9 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 8 ===
Panjang jalur: 6 langkah
Biaya energi: 9.50 unit
Efisiensi energi: 1.58 unit per langkah
Penghematan energi: 16.50 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 2, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 2, Baris 8, Kolom 16
  5. Lantai 2, Baris 5, Kolom 16
  6. Lantai 2, Baris 5, Kolom 15 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 9 ===
Panjang jalur: 6 langkah
Biaya energi: 9.50 unit
Efisiensi energi: 1.58 unit per langkah
Penghematan energi: 19.80 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 3, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 3, Baris 8, Kolom 16
  5. Lantai 3, Baris 2, Kolom 16
  6. Lantai 3, Baris 2, Kolom 15 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 10 ===
Panjang jalur: 10 langkah
Biaya energi: 15.90 unit
Efisiensi energi: 1.59 unit per langkah
Penghematan energi: 26.40 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 3, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 3, Baris 8, Kolom 16
  5. Lantai 3, Baris 6, Kolom 16
  6. Lantai 3, Baris 6, Kolom 6
  7. Lantai 3, Baris 5, Kolom 6
  8. Lantai 3, Baris 5, Kolom 4
  9. Lantai 3, Baris 4, Kolom 4
  10. Lantai 3, Baris 4, Kolom 3 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 12 ===
Panjang jalur: 6 langkah
Biaya energi: 9.50 unit
Efisiensi energi: 1.58 unit per langkah
Penghematan energi: 28.60 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 3, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 3, Baris 8, Kolom 16
  5. Lantai 3, Baris 6, Kolom 16
  6. Lantai 3, Baris 6, Kolom 3 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 13 ===
Panjang jalur: 6 langkah
Biaya energi: 9.50 unit
Efisiensi energi: 1.58 unit per langkah
Penghematan energi: 33.00 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 4, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 4, Baris 8, Kolom 14
  5. Lantai 4, Baris 2, Kolom 14
  6. Lantai 4, Baris 2, Kolom 3 [Ruangan Tujuan]

=== JALUR Jalur Optimal ke Ruangan 14 ===
Panjang jalur: 6 langkah
Biaya energi: 9.50 unit
Efisiensi energi: 1.58 unit per langkah
```

```
Penghematan energi: 26.40 unit
Koordinat jalur (Lantai, Baris, Kolom):
  1. Lantai 1, Baris 8, Kolom 3 [Sumber AC]
  2. Lantai 1, Baris 8, Kolom 17 [Tangga/Lift]
  3. Lantai 4, Baris 8, Kolom 17 [Tangga/Lift]
  4. Lantai 4, Baris 8, Kolom 14
  5. Lantai 4, Baris 6, Kolom 14
  6. Lantai 4, Baris 6, Kolom 5 [Ruangan Tujuan]
```

Tabel analisis jalur yang optimal

```
ANALISIS JALUR OPTIMAL
Jalur ke Ruangan @(0, 1, 14): 19 langkah, 5.9 unit energi (efisiensi: 0.31)
Jalur ke Ruangan @(0, 3, 2): 5 langkah, 1.1 unit energi (efisiensi: 0.22)
Jalur ke Ruangan @(0, 4, 12): 14 langkah, 4.3 unit energi (efisiensi: 0.31)
Jalur ke Ruangan @(0, 5, 2): 3 langkah, 1.1 unit energi (efisiensi: 0.37)
Jalur ke Ruangan @(1, 1, 6): 32 langkah, 12.7 unit energi (efisiensi: 0.40)
Jalur ke Ruangan @(1, 3, 2): 10 langkah, 6.3 unit energi (efisiensi: 0.63)
Jalur ke Ruangan @(1, 4, 8): 27 langkah, 12.7 unit energi (efisiensi: 0.47)
Jalur ke Ruangan @(1, 4, 14): 21 langkah, 9.5 unit energi (efisiensi: 0.45)
Jalur ke Ruangan @(2, 1, 14): 24 langkah, 9.5 unit energi (efisiensi: 0.40)
Jalur ke Ruangan @(2, 3, 2): 34 langkah, 15.9 unit energi (efisiensi: 0.47)
Jalur ke Ruangan @(2, 5, 2): 32 langkah, 9.5 unit energi (efisiensi: 0.30)
Jalur ke Ruangan @(3, 1, 2): 36 langkah, 9.5 unit energi (efisiensi: 0.26)
Jalur ke Ruangan @(3, 5, 4): 30 langkah, 9.5 unit energi (efisiensi: 0.32)

BIAYA ENERGI PER JENIS
Horizontal: 1.0 | Naik: 3.0 | Turun: 2.0 | Belok: 0.5 | Tekanan: 0.1 (semua per unit)

INFORMASI TEKNIS KESELURUHAN
Total Energi Sistem: 107.50 unit
Total Panjang Pipa (Visual): 287 unit
Efisiensi Rata-rata: 0.375 unit energi per langkah
Jumlah Ruangan: 13
Jumlah Lantai: 4 | Dimensi: 9x19 | Tangga/Lift: 6
```

Berikut hal-hal yang bisa dianalisis dari hasil yang diperoleh dari output program:

1. **Penanganan Kasus Tidak Terjangkau**
   Salah satu hasil paling signifikan adalah kemampuan program untuk mengidentifikasi ruangan yang tidak dapat dijangkau. Log output dengan jelas menyatakan: BFS: Tidak dapat menemukan jalur ke ruangan di (2, 3, 7). Verifikasi manual pada file input.txt menunjukkan bahwa ruangan pada koordinat tersebut (Lantai 3, Baris 4, Kolom 8) memang terisolasi total oleh dinding. Ini membuktikan bahwa algoritma BFS yang diimplementasikan bekerja dengan benar dan tuntas dalam menjelajahi semua kemungkinan tanpa menemukan solusi untuk target tersebut.
2. **Pemilihan Rute Berdasarkan Efisiensi**
   Dengan adanya dua akses vertikal, program menunjukkan kemampuannya untuk memilih rute yang paling efisien, tidak hanya berdasarkan kedekatan.
   - Pemilihan Tangga Terdekat: Untuk mencapai Ruangan 6 di Lantai 2, yang lokasinya di sisi kiri gedung, program dengan tepat memilih tangga kiri yang lebih dekat (jalur melewati titik (1, 2, 2) lalu ke (2, 2, 2)).
   - Adaptasi Terhadap Tangga Terputus: Untuk semua target di Lantai 3 dan 4, program secara cerdas selalu menggunakan tangga di sisi kanan (koordinat kolom 18). Contohnya pada jalur ke Ruangan 9 (Lantai 3), program melewati titik (1, 8, 17) lalu ke (3, 8, 17). Ini menunjukkan algoritma berhasil mengenali bahwa tangga kiri tidak dapat digunakan untuk mencapai lantai-lantai tersebut dan secara otomatis merutekan ulang pencarian melalui akses vertikal yang valid.
3. **Rincian Jalur Optimal dan Biaya Energi**
   Program menghasilkan rincian untuk 13 jalur yang berhasil ditemukan. Analisis beberapa jalur menunjukkan variasi kompleksitas dan biaya:
   - Jalur Sederhana: Rute ke Ruangan 2 adalah contoh jalur paling efisien, hanya memerlukan 2 langkah di lantai yang sama dengan biaya energi minimal 1.10 unit.
   - Jalur Kompleks: Rute ke Ruangan 10 di Lantai 3 adalah salah satu yang paling kompleks, memerlukan 10 langkah dengan biaya energi 15.90 unit. Rute ini melibatkan perpindahan vertikal melalui tangga kanan, diikuti oleh serangkaian pergerakan horizontal dan belokan yang rumit untuk menghindari halangan di Lantai 3.
   - Efisiensi per Langkah: Metrik efisiensi (unit energi per langkah) memberikan gambaran biaya relatif. Jalur horizontal yang lurus memiliki efisiensi tinggi (contoh: 0.55 untuk Ruangan 2 dan 4), sedangkan jalur yang melibatkan banyak belokan dan perpindahan vertikal memiliki efisiensi lebih rendah (contoh: 1.59 untuk beberapa rute ke lantai atas).
4. **Efektivitas Tahap Optimasi**
   Kolom "Penghematan energi" pada setiap hasil jalur menunjukkan efektivitas dari modul optimasi. Sebagai contoh, pada jalur ke Ruangan 13, program berhasil menghemat 33.00 unit energi dibandingkan dengan jalur awal yang mungkin ditemukan oleh BFS. Penghematan signifikan ini dicapai dengan meluruskan jalur dan mengurangi jumlah belokan yang tidak perlu, yang secara langsung mengurangi akumulasi biaya dari energy_costs['turn'] dan energy_costs['horizontal']

C. Analisis Kinerja

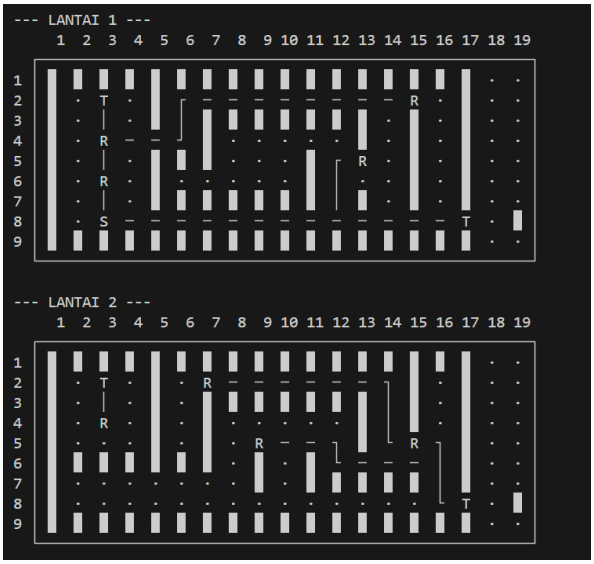
Program menghasilkan rincian untuk 13 jalur yang berhasil ditemukan. Analisis beberapa jalur menunjukkan variasi kompleksitas dan biaya:

- Jalur Sederhana: Rute ke Ruangan 2 adalah contoh jalur paling efisien, hanya memerlukan 2 langkah di lantai yang sama dengan biaya energi minimal 1.10 unit.

- Jalur Kompleks: Rute ke Ruangan 10 di Lantai 3 adalah salah satu yang paling kompleks, memerlukan 10 langkah dengan biaya energi 15.90 unit. Rute ini melibatkan perpindahan vertikal melalui tangga kanan, diikuti oleh serangkaian pergerakan horizontal dan belokan yang rumit untuk menghindari halangan di Lantai 3.
- Efisiensi per Langkah: Metrik efisiensi (unit energi per langkah) memberikan gambaran biaya relatif. Jalur horizontal yang lurus memiliki efisiensi tinggi (contoh: 0.55 untuk Ruangan 2 dan 4), sedangkan jalur yang melibatkan banyak belokan dan perpindahan vertikal memiliki efisiensi lebih rendah (contoh: 1.59 untuk beberapa rute ke lantai atas).

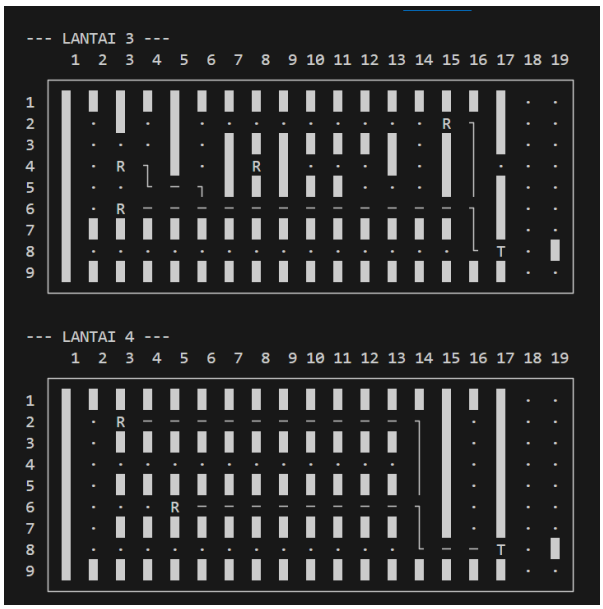
D. Interpretasi Visualisasi Blueprint

Berikut adalah visualisasi blueprint dengan pemetaan yang dihasilkan oleh program :



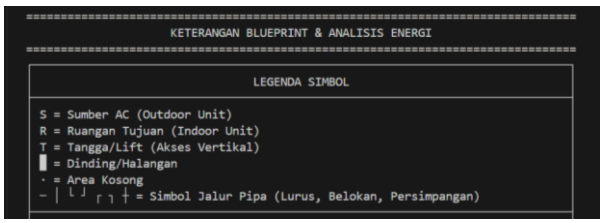

**Gambar 1**. Blueprint 1

**Sumber :** Arsip Pribadi




**Gambar 2.** Blueprint 2

**Sumber :** Arsip Pribadi




**Gambar 3.** Legenda Blueprint

**Sumber :** Arsip Pribadi

Blueprint yang dihasilkan memberikan representasi visual yang sangat membantu untuk perencanaan instalasi.

- Blueprint Lantai 1: Terlihat jelas bagaimana semua jalur berawal dari titik 'S' (di kiri bawah) dan menyebar. Beberapa jalur langsung menuju target di lantai yang sama, sementara jalur lain mengarah ke salah satu dari dua titik 'T' untuk distribusi ke lantai atas.
- Blueprint Lantai 2-4: Pada lantai-lantai berikutnya, terlihat pipa "muncul" dari titik 'T' dan kemudian menjalar menuju ruangan target di lantai tersebut. Hal ini mensimulasikan bagaimana pipa akan didistribusikan dari poros vertikal utama.

Hasil pengujian menunjukkan bahwa program ACPathfinder berhasil tidak hanya menemukan jalur yang valid dalam denah yang kompleks, tetapi juga mengoptimalkannya berdasarkan model biaya energi, serta menyajikannya dalam format laporan dan visual yang informatif dan mudah dipahami.

E. Interpretasi Unit Energi

Metrik "unit energi" dalam program ini pada dasarnya bukanlah satuan energi fisik, melainkan sebuah indeks biaya abstrak yang secara komprehensif merepresentasikan berbagai

faktor kesulitan dalam proses instalasi di dunia nyata. Indeks ini memperhitungkan biaya dasar pemasangan pipa lurus (horizontal), biaya tambahan untuk material dan tenaga kerja saat membuat belokan (turn), tingkat kesulitan yang lebih tinggi untuk menembus struktur lantai (vertical_up/vertical_down), hingga mensimulasikan biaya operasional jangka panjang akibat penurunan tekanan pada pipa yang panjang (base_pressure). Dengan demikian, skor ini tidak hanya mencerminkan biaya instalasi awal, tetapi juga efisiensi sistem dalam jangka panjang.

Untuk dapat diimplementasikan di dunia nyata, skor abstrak ini dapat diubah menjadi estimasi biaya finansial melalui proses kalibrasi sederhana. Seorang perencana atau kontraktor hanya perlu menentukan nilai moneter untuk satu unit dasar, misalnya biaya total pemasangan 1 meter pipa lurus adalah Rp 150.000,-. Setelah nilai-nilai biaya lain di dalam program disesuaikan sebagai rasio terhadap acuan ini, program dijalankan untuk mendapatkan total skor "unit energi" untuk keseluruhan proyek. Hasil skor akhir tersebut kemudian dikalikan dengan nilai moneter unit dasar yang telah ditentukan untuk menghasilkan estimasi biaya proyek yang akurat dalam Rupiah, mengubah model teoretis ini menjadi alat bantu pengambilan keputusan yang praktis.

## V. Kesimpulan

Makalah ini telah mengimplementasikan dan mengevaluasi sebuah sistem otomatis untuk perencanaan rute pipa AC sentral pada gedung bertingkat, yang bertujuan untuk mengatasi inefisiensi dan subjektivitas pada proses perancangan manual. Sistem yang diusulkan menggunakan pendekatan dua tahap yang mengombinasikan algoritma *Breadth-First Search* (BFS) dengan modul optimasi berbasis biaya energi. Hasil pengujian pada skenario gedung bertingkat yang kompleks menunjukkan bahwa sistem mampu menangani kasus-kasus sulit secara efektif. Kapabilitas ini mencakup identifikasi akurat terhadap ruangan yang tidak dapat dijangkau, pemilihan rute vertikal yang logis di antara beberapa alternatif, serta kemampuan untuk beradaptasi terhadap jalur akses yang terputus. Lebih lanjut, tahap optimasi terbukti memberikan peningkatan efisiensi yang substansial dengan mengurangi total biaya energi dari rute awal yang ditemukan oleh BFS.

Berdasarkan hasil analisis, dapat disimpulkan bahwa pendekatan hibrida antara BFS dan optimasi spesifik merupakan strategi yang efektif dan efisien untuk masalah perencanaan rute pipa AC. Metode ini berhasil menyediakan solusi yang terukur dan konsisten, yang dapat menjadi alternatif unggul dibandingkan proses perencanaan manual. Untuk pengembangan di masa mendatang, beberapa area dapat dieksplorasi. Implementasi algoritma pencarian berbobot seperti Dijkstra atau A* dapat diteliti untuk mencapai optimalitas global secara langsung. Penambahan parameter fisis lainnya, seperti diameter pipa dan jenis material, juga akan meningkatkan akurasi serta relevansi simulasi terhadap kondisi lapangan yang sebenarnya.

## VI. Ucapan Terima Kasih

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga makalah ini dapat terselesaikan dengan baik. Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Rinaldi Munir atas bimbingan, ilmu, dan wawasan yang sangat berharga. Terima kasih juga penulis sampaikan kepada seluruh teman-teman mahasiswa atas dukungan dan diskusi yang membangun selama proses pengerjaan.

Penulis menyadari bahwa makalah ini masih jauh dari sempurna dan terbuka untuk segala bentuk kritik maupun saran yang membangun. Besar harapan penulis agar makalah ini dapat memberikan manfaat serta menjadi referensi yang berguna bagi para pembaca.

## VII. Referensi

[1] A. Jain and S. Garg, “Design and Optimization of HVAC Duct Systems,” *International Journal of Engineering Research & Technology*, vol. 4, no. 6, pp. 712–717, 2015.

[2] M. F. A. Rahman and S. A. Hassan, “Automated HVAC Planning in Smart Buildings Using Graph-Based Pathfinding,” *Proceedings of the 2022 IEEE International Smart Cities Conference (ISC2)*, pp. 341–346, Sep. 2022.

[3] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson, 2020.

[4] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed., Addison-Wesley, 2011.

[5] R. Munir and N. U. Maulidevi, “Breadth/Depth First Search (BFS/DFS) (Bagian 1),” Bahan Kuliah IF2211 Strategi Algoritmik, Program Studi Teknik Informatika, Institut Teknologi Bandung, 2025.

[6] R. Munir and N. U. Maulidevi, “Breadth/Depth First Search (BFS/DFS) (Bagian 2),” Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Institut Teknologi Bandung, 2025.

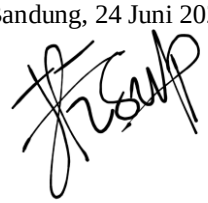
## VIII. Pranala GitHub

Tautan kode : https://github.com/Ferdin-Arsenic/HVAC.git

## 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, 24 Juni 2025



Ferdin Arsenarendra Purtadi
13523117