LAPORAN AKHIR TUGAS BESAR 2
PENGENALAN KOMPUTASI
“SISTEM LIFT GEDUNG BERTINGKAT”

Disusun oleh:
KELOMPOK 04 - K31

Yusuf Faishal Listyardi	19624217
Jonathan Alveraldo Bangun	19624234
Kayla Fiyaza Nawal Zaghbi	19624249
Muhmmad Okto Huzainy	19624263
Natanael I. Manurung	19624283

Dosen: Dr. Fazat Nur Azizah, S.T., M.Sc., Dr. Maya Nabila, S.Si., M.Si.

Tahap Persiapan Bersama
Sekolah Teknik Elektro dan Informatika - Komputasi
Institut Teknologi Bandung
2024

SURAT PERNYATAAN PENGGUNAAN KECERDASAN BUATAN/AI

Dengan ini, kami:

Kelompok : 4
Fakultas/Sekolah : Sekolah Teknik elektro dan Informatika - Komputasi
Kode Mata Kuliah : WI1102
Nama Mata Kuliah : Berpikir Komputasional
Judul Tugas/Proposal : Sistem Lift Gedung Bertingkat

Menyatakan bahwa kami menggunakan Ai / kecerdasan buatan dalam pengerjaan maupun penyusunan luaran tugas pada mata kuliah yang tertulis diatas.

Jika Ya, maka alat AI/kecerdasan buatan yang digunakan adalah:

Lingkup Pekerjaan	Digunakan?	Tingkat Penggunaan (level 1-5)
Pemeriksaan Ejaan: menggunakan tools seperti Grammarly, Deepl, Quillbot, Grammarbot, LanguangeTool, ProWritingAid, ChatGPt, Google Gemini, atau sejenisnya. **Sebutkan tool yang digunakan:**	Tidak	
Pembuatan Teks: menggunakan tools seperti ChatGPT, Gemini, Coilot, GrammarlyGO, WordAI, WriteSonic, JAsper, Jenni AI, atau sejenisnya. **Sebutkan tool yang digunakan:**	Tidak	
Bantuan Diskusi dalam Penyusunan Konten: menggunakan tools seperti ChatGPT, Gemini, Copilot, Perplexity, Jenni AI, Zoom-Companion, atau sejenisnya. **Sebutkan tool yang digunakan:** ChatGPT	Ya	1
Pembuatan Gambar, Video, dan/atau Grafik: menggunakan tools seperti Craiyon, DALL-E, Midjourney, Stable Diffusion, Microsoft Designer, Gemini, Canva AI, atau sejenisnya. **Sebutkan tool yang digunakan:**	Tidak	

Kami juga menyatakan bahwa setiap penggunaan AI/kecerdasan buatan yang dilakukan pada mata kuliah tersebut telah dinyatakan di dalam surat ini.

Jatinangor, 11 Desember 2024

Penulis

KATA PENGANTAR

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga kami dapat menyelesaikan tugas besar ini yang berjudul Sistem Penentuan Lift Tercepat dengan baik dan tepat waktu. Penulisan laporan ini merupakan bagian dari tugas besar dalam mata kuliah Berpikir Komputasional (WI1102)

Laporan ini dirancang untuk memaparkan proyek simulasi sistem kerja lift yang bertujuan meningkatkan efisiensi operasional serta meminimalkan waktu tunggu penumpang. Melalui proyek ini, kami berupaya mengintegrasikan konsep-konsep pemrograman komputasional dan algoritma dalam menyelesaikan masalah nyata yang berkaitan dengan sistem transportasi vertikal di gedung bertingkat.

Kami menyadari bahwa laporan ini tidak lepas dari keterbatasan, namun kami berharap hasil dari proyek ini dapat memberikan kontribusi dalam pengembangan sistem lift yang lebih baik di masa depan. Kami juga ingin menyampaikan terima kasih yang sebesar-besarnya kepada dosen dan teman-teman yang telah memberikan bimbingan, dukungan, dan masukan yang sangat berharga selama proses pengerjaan tugas ini.

Akhir kata, semoga laporan ini dapat memberikan manfaat bagi pembaca serta menjadi referensi bagi pengembangan teknologi serupa di masa mendatang. Kami sangat terbuka terhadap kritik dan saran untuk menyempurnakan karya ini di masa depan.

DAFTAR ISI

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi yang semakin pesat di era digital ini, menjadikan teknologi sebagai solusi dari berbagai permasalahan dan aspek yang penting dalam kehidupan sehari-hari. Salah satu bentuk teknologi yang merupakan dasar dari kemajuan teknologi modern yang amat pesat adalah bahasa pemrograman yang memungkinkan manusia dalam mengendalikan serta menciptakan suatu teknologi secara efisien. Bahasa pemrograman akan memecahkan berbagai permasalahan secara efisien melalui berbagai instruksi yang dapat dipahami oleh komputer. Adapun pendekatan ini dikenal sebagai berpikir komputasional, yaitu cara berpikir secara sistematis dan logis dalam menemukan suatu solusi dari permasalahan yang kompleks.

Kemajuan teknologi memungkinkan berbagai penerapan pemrograman untuk meningkatkan infrastruktur dari suatu bangunan. Salah satu penerapannya adalah dalam pembangunan lift pada suatu gedung. Pada gedung bertingkat tinggi, lift menjadi suatu solusi dalam aspek transportasi antar lantai untuk mempermudah dalam perpindahan lantai. Akan tetapi, tata kelola lift yang kurang maksimal, seperti lama nya waktu tunggu lift, dapat mengurangi efisiensi dan pengalaman pengguna. Selain itu, seperti alat teknologi pada umumnya, suatu lift pasti memiliki batas penggunaan maksimum. Untuk menghindari terjadinya error ketika sedang digunakan oleh penumpang. Untuk itulah kami merancang program lift untuk efisiensi waktu penumpang dan mencegah terjadinya error yang dapat mengurangi pengalaman pengguna.

Proyek ini dirancang untuk menentukan lift yang paling efisien bagi pengguna berdasarkan waktu tempuh, arah tujuan, posisi lantai, dan kapasitas muatan, serta untuk menentukan waktu suatu lift harus berhenti beroperasi untuk di servis. Lift akan dipilih berdasarkan waktu tempuh yang paling singkat untuk mencapai posisi pengguna. Selain itu, lift akan memperhitungkan kapasitas muatan sebagai waktu tempuh tambahan ketika melebihi batas. Sementara itu, untuk penentuan waktu servis, lift akan memperhitungkan jumlah trip yang telah dilakukan. Proyek ini memperhitungkan situasi yang mungkin terjadi pada sistem lift yang sesungguhnya, sehingga dapat dijadikan langkah awal dalam pengembangan perangkat lunak sistem lift yang lebih kompleks.

1.2 Tujuan

1. Memenuhi tugas besar 1 mata kuliah Berpikir Komputasional
2. Mengembangkan sistem kerja lift yang memaksimalkan efisiensi operasional lift dan meminimalkan waktu tunggu penumpang.
3. Mengetahui batas operasional suatu lift

1.3 Manfaat

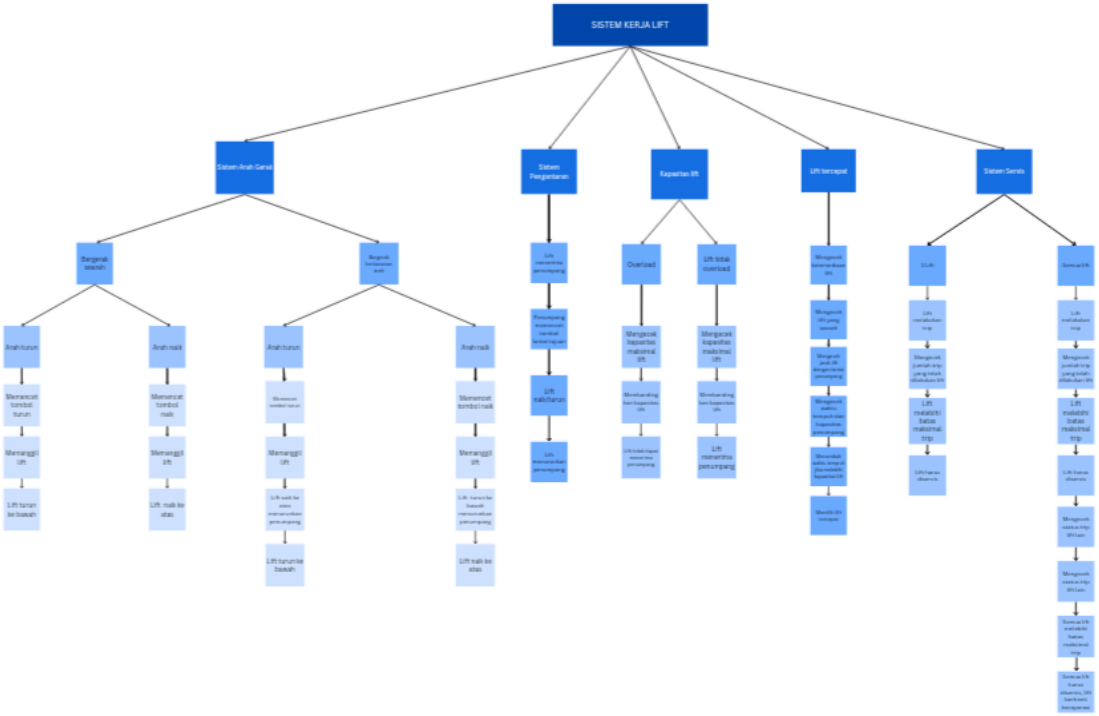
1. Mengetahui cara pengaplikasian program python dalam meningkatkan efisiensi penggunaan lift.
2. Membantu meminimalkan waktu tunggu pengguna dengan memilih lift tercepat untuk mengantarkan pengguna berdasarkan lokasi dan tujuannya.
3. Mengetahui cara pengaplikasian program python dalam menentukan batas penggunaan suatu lift di gedung bertingkat.

1.4 Rumusan Masalah

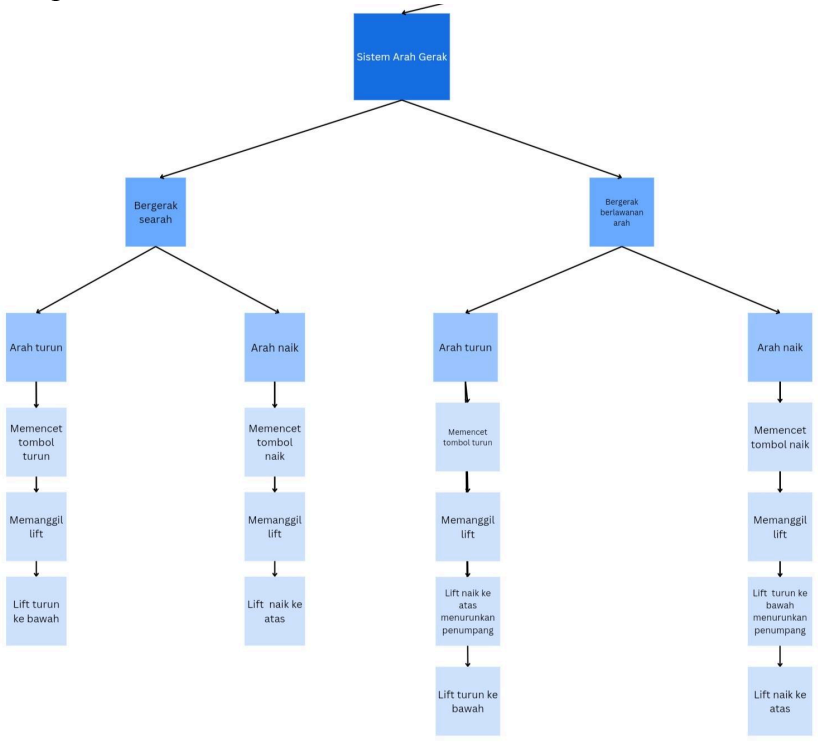
1. Bagaimana cara menentukan lift yang paling efisien untuk pengguna berdasarkan waktu tempuh menggunakan program komputasi python?
2. Bagaimanakan cara mengetahui batas operasional suatu lift?

BAB 2
PEMBAHASAN

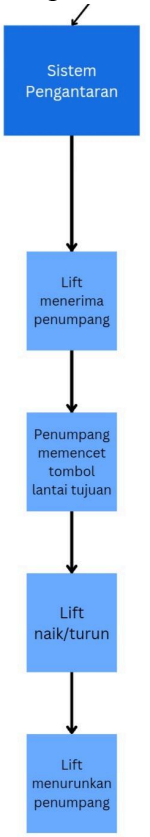
2.1. Dekomposisi Masalah



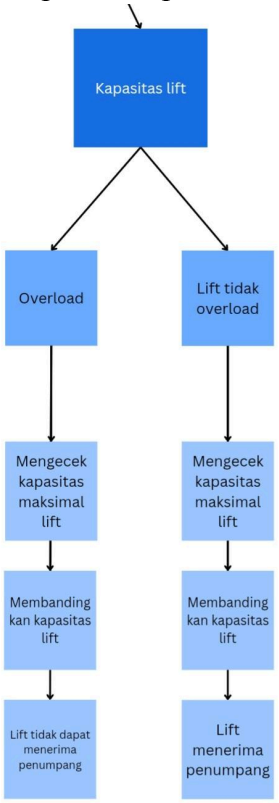
a. Dekomposisi Sistem Gerak



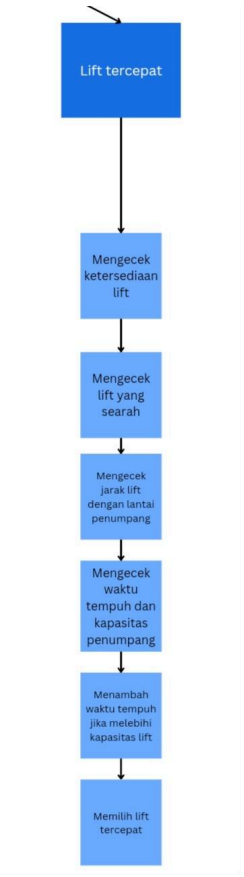
b. Dekomposisi Sistem Pengantaran



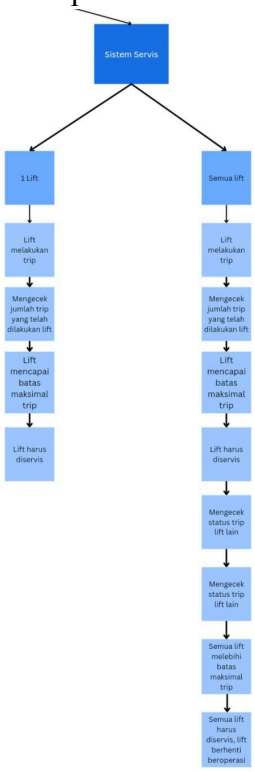
c. Dekomposisi Kapasitas Lift



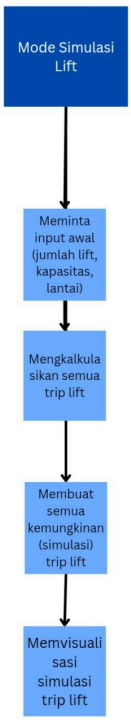
d. Dekomposisi Lift Tercepat



e. Dekomposisi Servis



2.2 Dekomposisi Simulasi



2.3 Abstraksi

Lift 1

- Posisi Awal: Lantai awal tempat lift berada (misalnya, lantai 2).
- Target Awal: Lantai yang menjadi tujuan pertama lift sebelum ada permintaan baru (lantai 5).
- Status Awal: Apakah lift sedang digunakan atau tidak (sedang digunakan).
- Arah Gerak: Arah lift bergerak (naik berdasarkan posisi dan tujuan).
- Kapasitas: Beban yang ada di dalam lift, ditentukan oleh jumlah orang yang diangkut (4 orang, kapasitas maksimum 10 orang).
- Antrian Tujuan: Daftar lantai tambahan yang harus dikunjungi jika ada lebih dari satu permintaan (tidak ada).
- Indikator Servis: Telah menempuh 30 trip (belum perlu servis karena batas 50 trip).

Lift 2

- Posisi Awal: Sama seperti Lift 1, tetapi posisi awalnya berbeda (lantai 7).
- Target Awal: Tujuan pertama Lift 2 sebelum ada permintaan baru (lantai 3).
- Status Awal: Sama seperti Lift 1, dapat sedang digunakan atau tidak (tidak digunakan).
- Arah Gerak: Bergantung pada posisi awal Lift 2 dan tujuannya (diam).
- Kapasitas: Sama seperti Lift 1, dihitung berdasarkan jumlah orang yang ada di dalamnya (0 orang, kapasitas maksimum 10 orang).
- Antrian Tujuan: Berbeda dengan Lift 1 karena permintaan pengguna yang berbeda (tidak ada).
- Indikator Servis: Telah menempuh 45 trip (hampir mendekati batas servis 50 trip).

Lift 3

- Posisi Awal: Lokasi awal lift sebelum bergerak (lantai 1).
- Target Awal: Sama seperti Lift 1 dan Lift 2, targetnya bisa berbeda (lantai 10).
- Status Awal: Menentukan apakah Lift 3 sedang digunakan atau tidak (sedang digunakan).

- Arah Gerak: Sama seperti lift lainnya, bisa naik atau turun (naik).
- Kapasitas: Kapasitas Lift 3 sama, tetapi jumlah beban aktual dapat berbeda (8 orang, kapasitas maksimum 10 orang).
- Antrian Tujuan: Sesuai dengan permintaan yang diterima Lift 3 (lantai 6).
- Indikator Servis: Telah menempuh 50 trip (sudah harus diservis).

Abstraksi:

Perbedaan di antara lift-lift ini terutama terletak pada:

1. Posisi Awal: Setiap lift berada di lantai yang berbeda ketika memulai.
2. Target Awal: Tujuan awal masing-masing lift bervariasi tergantung pengaturan awal.
3. Antrian Tujuan: Setiap lift memiliki daftar lantai yang berbeda untuk dikunjungi berdasarkan permintaan pengguna.
4. Arah Gerak: Arah gerakan lift dapat berbeda tergantung posisi dan tujuan awal.
5. Indikator Servis: Waktu servis untuk masing-masing lift bergantung pada jumlah trip yang telah ditempuh.

2.4 Pengenalan Pola

Pola Pergerakan Lift:

- Lift hanya dapat bergerak dalam tiga kondisi:
    - Naik (up): Menuju lantai yang lebih tinggi.
    - Turun (down): Menuju lantai yang lebih rendah.
    - Diam (stationary): Menunggu di lantai tertentu tanpa ada permintaan.

Pola Pengelompokan Permintaan:

- Penumpang dengan arah tujuan yang sama dikelompokkan dalam perjalanan yang sama.
- Lift memprioritaskan permintaan searah dengan arah geraknya sebelum menangani permintaan lain.

Pola Overload Kapasitas:

- Jika kapasitas lift terlampaui, sistem membagi penumpang menjadi beberapa perjalanan (trip).
- Perjalanan tambahan diatur untuk meminimalkan waktu tunggu penumpang.

Pola Prioritas Servis:

- Lift yang telah mencapai batas perjalanan (trip) masuk ke mode servis.
- Sistem memastikan lift lain menggantikan lift yang diservis untuk menjaga operasional.

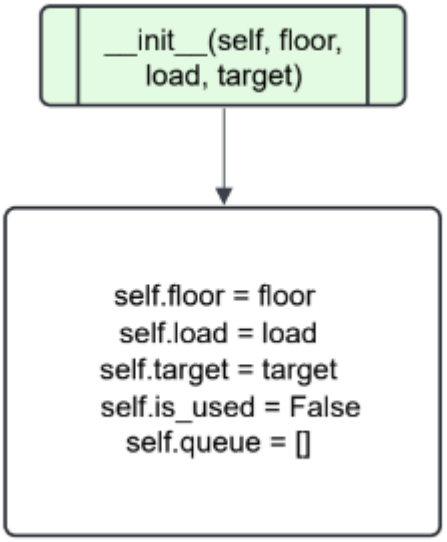
Pola Efisiensi Waktu:

- Lift dengan waktu tempuh terpendek diprioritaskan untuk memenuhi permintaan.
- Faktor yang dipertimbangkan: posisi lift, arah gerak, dan antrian tujuan.

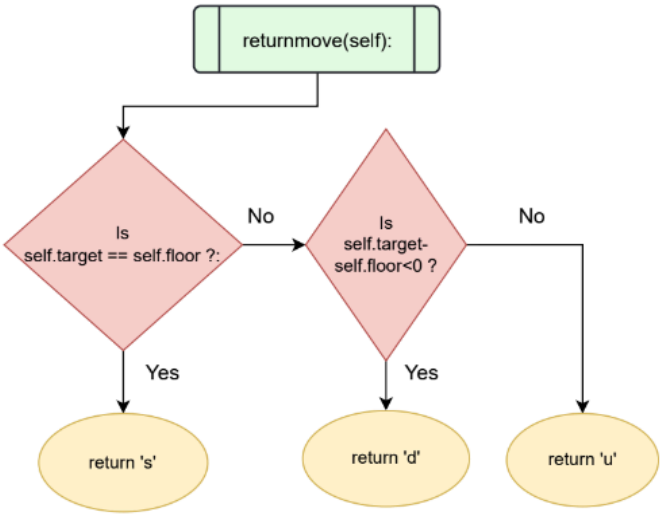
2.5 Algoritma

2.5.1. Flowchart

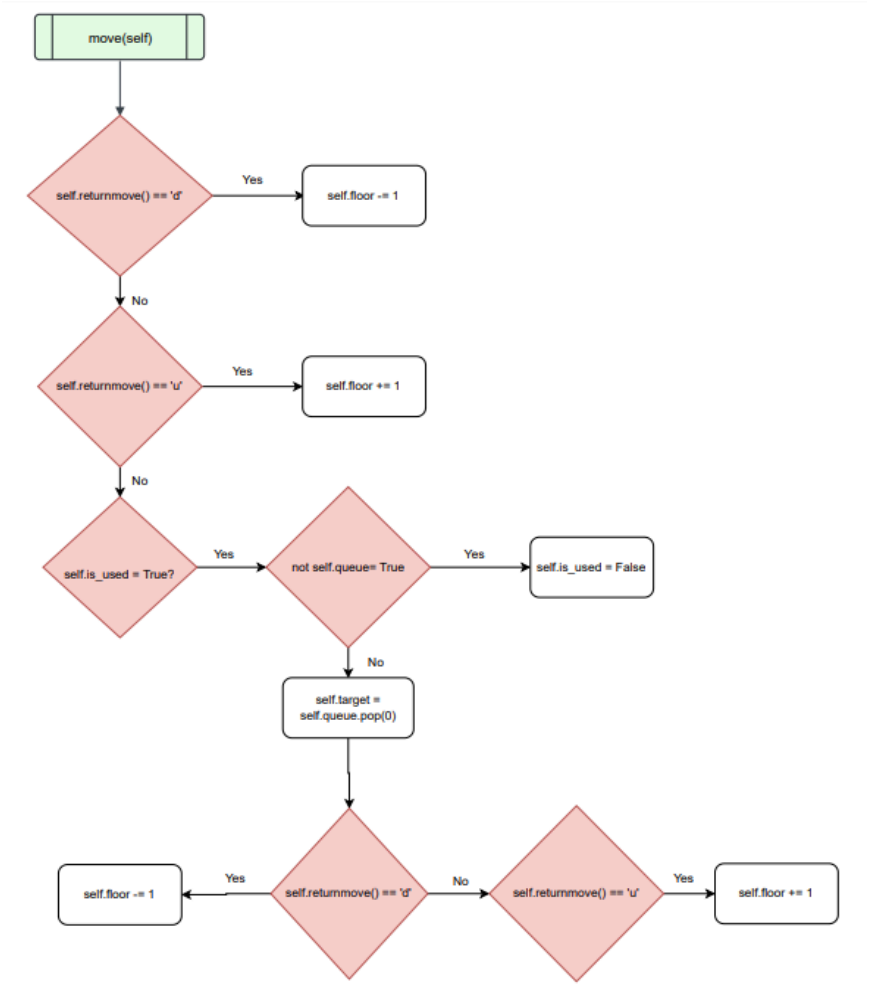
1. Subprogram Atribut Setiap Lift



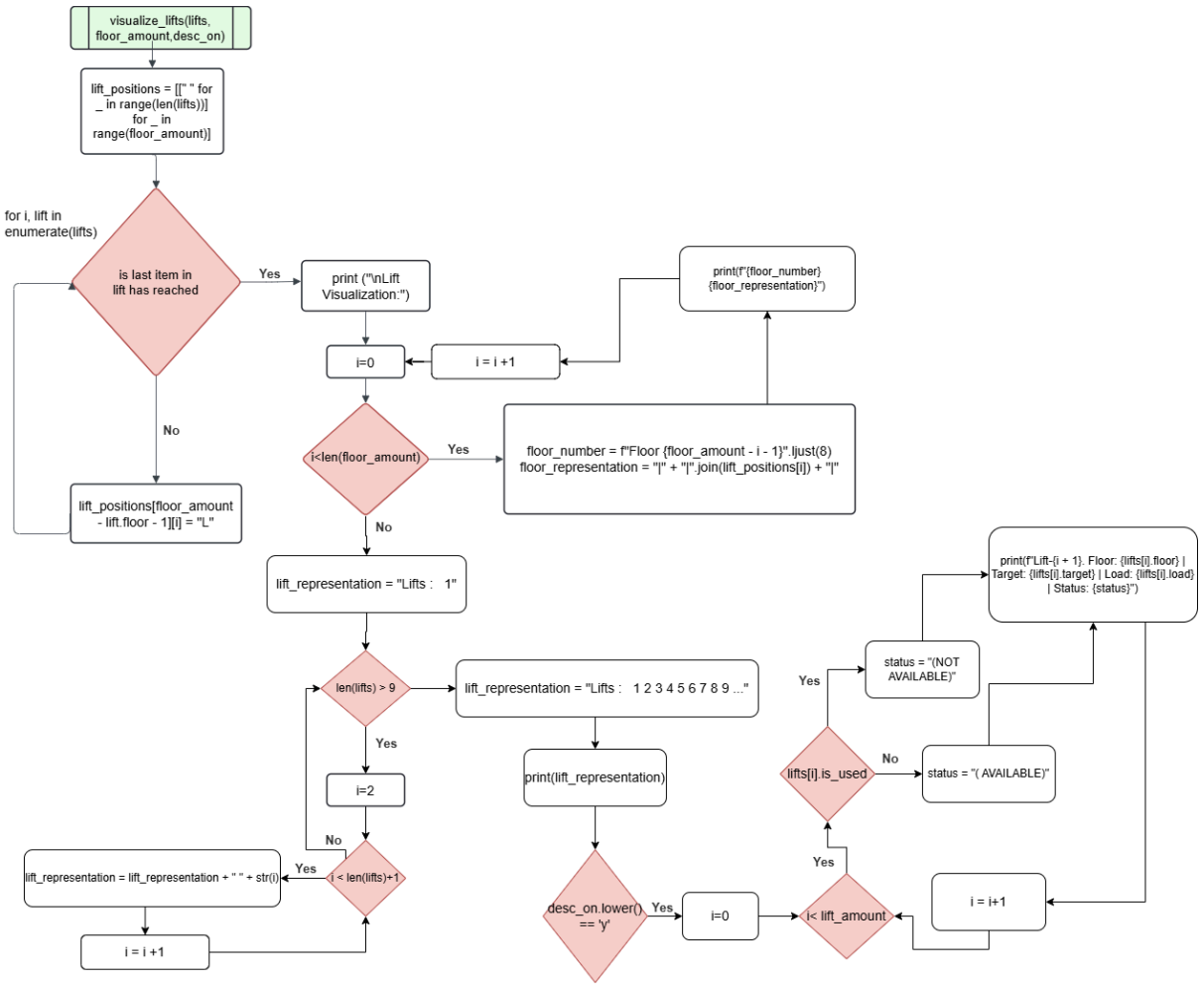
2. Subprogram Return Move



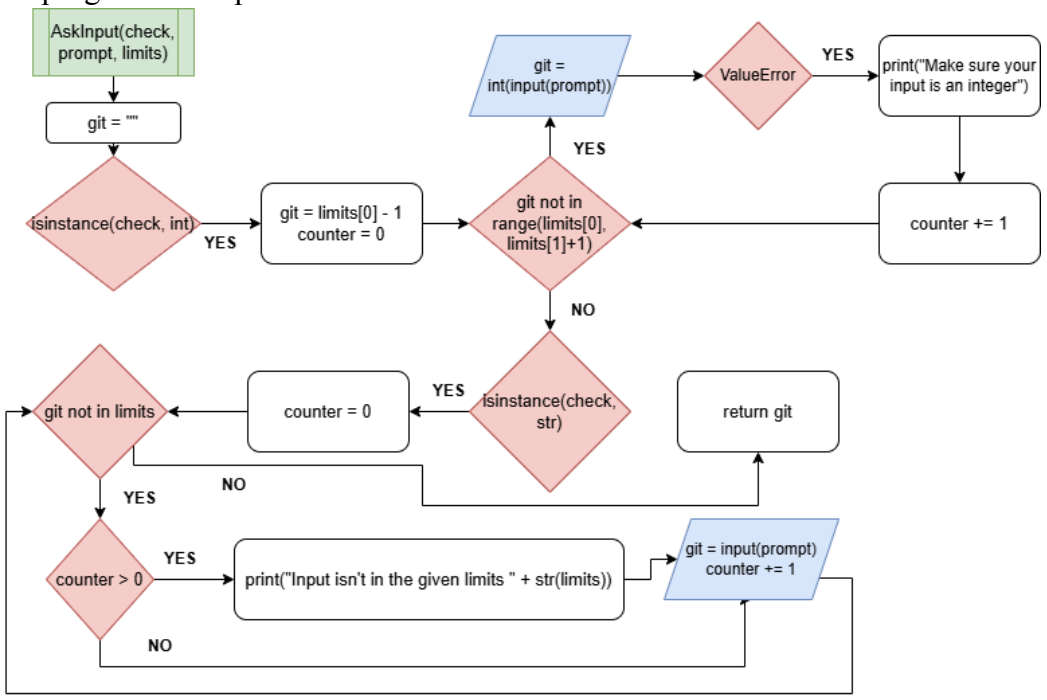
3. Subprogram Move



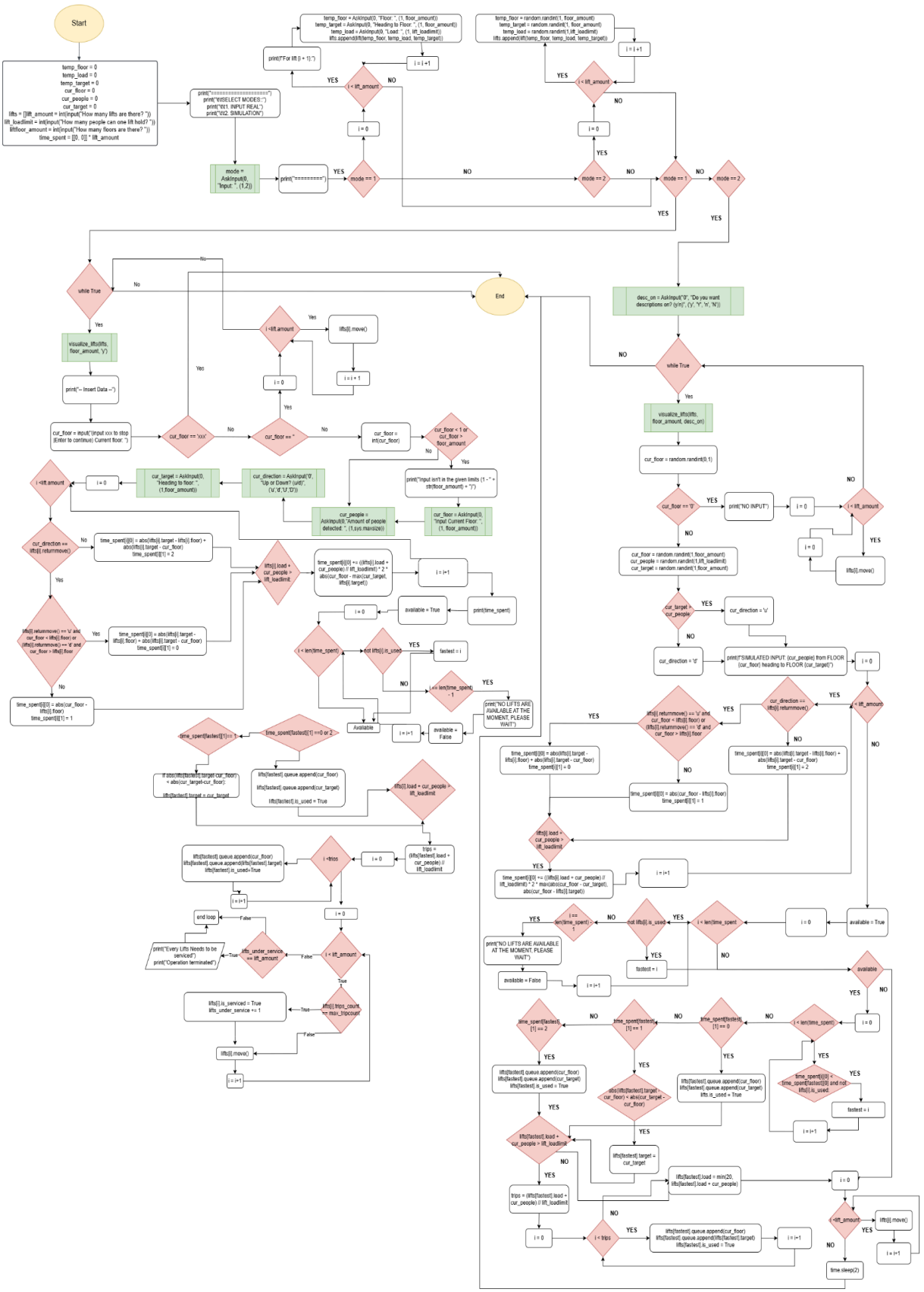
4. Subprogram Visualize Lift



5. Subprogram Askinput



6. Flowchart Keseluruhan



2.5 Source Code

Berikut ini adalah kode program kami

```
# Simulasi Sistem Lift

# Program ini mensimulasikan operasi beberapa lift di sebuah gedung.
# Pengguna memberikan input untuk jumlah lift, jumlah lantai,
# dan kapasitas muatan tiap lift, dan sistem mensimulasikan bagaimana lift
# bergerak berdasarkan permintaan penumpang.

# Kamus:
#   - Titlecard           (str): Judul besar program
#   - lifts              (list): Daftar yang berisi objek lift. Setiap objek lift berisi:
#       - floor           (int): Posisi lantai lift saat ini.
#       - load            (int): Jumlah orang yang ada di dalam lift saat ini.
#       - target          (int): Lantai target yang ingin dituju lift.
#       - is_used        (bool): Flag yang menandakan apakah lift sedang digunakan.
#       - queue          (list): Daftar lantai yang harus dikunjungi lift untuk perjalanan tambahan.
#   - lift_amount         (int): Jumlah lift yang ada di gedung.
#   - lift_loadlimit      (int): Batas maksimum muatan (jumlah orang) yang bisa ditampung oleh lift.
#   - floor_amount        (int): Jumlah lantai yang ada di gedung./
#   - mode                (int): memilih mode (1 = input real, 2 = simulasi)
#   - time_spent         (list): Daftar yang berisi perhitungan waktu untuk setiap lift, dengan:
#       - time_spent[x][0] (int): Waktu yang dibutuhkan lift untuk memenuhi permintaan.
#       - time_spent[x][1] (int): Jenis pergerakan (0: arah yang sama, 1: arah yang sama dalam jarak tertentu, 2: arah berbeda).
#   - temp_floor          (int): Variabel sementara untuk menyimpan lantai lift saat inisialisasi.
#   - temp_load           (int): Variabel sementara untuk menyimpan jumlah muatan lift saat inisialisasi.
#   - temp_target         (int): Variabel sementara untuk menyimpan lantai target lift saat inisialisasi.
#   - cur_floor           (int): Lantai saat ini dari pengguna yang meminta lift.
```

```
#   - cur_floor           (int): Lantai saat ini dari pengguna yang meminta lift.
#   - cur_people          (int): Jumlah orang yang terdeteksi oleh sistem di lantai saat ini.
#   - cur_target          (int): Lantai target dari pengguna yang meminta lift.
#   - lifts_under_service (int): Jumlah lift yang sedang diservis
#   - max_tripcount       (int): Jumlah maksimal trip yang dapat dilalui
#   - available          (bool): Flag yang menandakan apakah ada lift yang tersedia.
#   - fastest             (int): Indeks lift tercepat yang akan memenuhi permintaan saat ini.

# Fungsi:
#   - lift.returnmove()                   : Menentukan pergerakan lift: diam (s), naik (u), atau turun (d).
#   - lift.move()                         : Memindahkan lift menuju lantai target.
#   - visualize_lifts(lifts, floor_amount) : Memvisualisasikan posisi dan status setiap lift.
#   - AskInput(check, prompt, limits)     : Meminta input dan memastikan input tidak menimbulkan error di program
#   - main loop                           : Menangani input pengguna dan mengendalikan permintaan serta pergerakan lift.

import sys, random, time
```

```
titlecard = """
================================================================================
     .-') _     ('-.             .-. .-')             _ (`-. .-. .-')
    ( OO ) )  ( OO ).-.         \  ( OO )           ( (OO  )\  ( OO )
,--./ ,--,'   / . --. /  ,--.-') ,--. ,--.  ,--.-')  _.`     \,--. ,--.
|   \ |  |\   | \-.  \   |  |OO)|  .'   /  |  |OO)(__...--''|  .'   /
|    \|  | ).-'-'  |  |  |  |  \|      /,  |  |  \ |  /  | ||      /,
|  .     |/  \| |_.'  |  |  |(_/|     ' _) |  |(_/ |  |_.' ||     ' _)
|  |\    |    |  .-.  | ,|  |_.'|  .   \  ,|  |_.' |  .___.'|  .   \\
|  | \   |    |  | |  |(_|  |   |  |\   \(_|  |    |  |     |  |\   \\
`--'  `--'    `--' `--'  `--'   `--' '--'  `--'    `--'     `--' '--'

                    v 1.4
                    Jangan liftnya doang yang naik bre...
================================================================================
"""
print(titlecard)

class lift:
    def __init__(self, floor, load, target):
        # Inisialisasi objek lift
        self.floor = floor  # Lokasi lift sekarang
        self.load = load  # Beban di lift sekarang
        self.target = target  # Tujuan lift sekarang
```

```
        self.is_used = False  # Indikator apakah lift sedang digunakan untuk beberapa trip
        self.queue = []  # Barisan beberapa trip lift
        self.trips_count = 0 # Berapa lantai sudah dijalani lift
        self.is_serviced = False # Indikator apakah lift sedang diservis

    def returnmove(self):
        # Menentukan pergerakan lift, s = stationary, u = up, d = down
        if self.target == self.floor:
            return 's'
        return 'd' if self.target - self.floor < 0 else 'u'

    def move(self):
        # Mengerakkan lift menuju targetnya
        if self.returnmove() == 'd':
            self.floor -= 1  # Ke atas
            self.trips_count += 1
        elif self.returnmove() == 'u':
            self.floor += 1  # Ke bawah
            self.trips_count += 1
        else:
```

```
            # Ketika berhenti, mengecek jika misalkan ada target lainnya di barisan queue
            if self.is_used:
                if not self.queue:  # Kalau kosong, bearti lift sudah siap digunakan untuk beberapa trip
                    self.is_used = False
                    self.load = 0
                else:
                    # Mengubah target berdasarkan queue
                    self.target = self.queue.pop(0)
                    # Mulai menggerakkan lift kembali
                    if self.returnmove() == 'd':
                        self.floor -= 1
                        self.trips_count += 1
                    elif self.returnmove() == 'u':
                        self.floor += 1
                        self.trips_count += 1


def visualize_lifts(lifts, floor_amount, desc_on):
    # Visualisasi posisi lift di setiap lantai
    lift_positions = [[" " for _ in range(len(lifts))] for _ in range(floor_amount)]
    for i, lift in enumerate(lifts):
        lift_positions[floor_amount - lift.floor][i] = "L"  # Penandaan posisi lift
    print("\nLift Visualization:")
```

```
    for i in range(floor_amount):
        # Penampilan lantai
        floor_number = f"Floor {floor_amount - i}".ljust(8)
        floor_representation = "|" + "|".join(lift_positions[i]) + "|"
        print(f"{floor_number} {floor_representation}")
    # Penampilan status setiap lift
    lift_representation = "Lifts :   1"
    print(len(lifts))
    if len(lifts) <= 9:
        for i in range(2,len(lifts)+1):
            lift_representation = lift_representation + " " + str(i)
    else:
        lift_representation = "Lifts :   1 2 3 4 5 6 7 8 9 ..."
    print(lift_representation)
    if desc_on.lower() == 'y':
        for i in range(lift_amount):
            status = "(NOT AVAILABLE)" if lifts[i].is_used else "AVAILABLE"
            if lifts[i].is_serviced:
                status = "(NEEDS SERVICE)"
            print(f"Lift-{i + 1}. Floor: {lifts[i].floor} | Target: {lifts[i].target} | Load: {lifts[i].load} | Trips: {lifts[i].trips_count} | Statu
```

```
### BARU 1
def AskInput(check, prompt, limits):
    # Meminta input untuk setiap variabel dan memastikan input tidak menimbulkan error
    git = ""
    if isinstance(check, int):
        git = limits[0] - 1
        counter = 0
        while git not in range(limits[0], limits[1]+1):
            if counter > 0:
                print("Input isn't in the given limits (" + str(limits[0]) + " - " + str(limits[1]) + ")")
            try:
                git = int(input(prompt))
            except ValueError:
                print("Make sure your input is an integer")
            counter += 1
    if isinstance(check, str):
        counter = 0
        while git not in limits:
            if counter > 0:
                print("Input isn't in the given limits " + str(limits))
            git = input(prompt)
            counter += 1
    return git
```

```
# Variabel input pengguna terkait kondisi awal
temp_floor = 0
temp_load = 0
temp_target = 0
cur_floor = 0
cur_people = 0
cur_target = 0
lifts_under_service = 0
lifts = []
lift_amount = AskInput(0, "How many lifts are there? ", (1, sys.maxsize))  # Jumlah lift
lift_loadlimit = AskInput(0, "How many people can one lift hold? ", (1, sys.maxsize))# Batasan spesifikasi lift
floor_amount = AskInput(0, "How many floors are there? ", (1, sys.maxsize)) # Jumlah lantai
max_tripcount = AskInput(0, "How many trips before service? ", (1, sys.maxsize)) # Jumlah penggunaan maksimal
time_spent = [[0, 0] for i in range(lift_amount)]  # Pengecek waktu yang dibutuhkan setiap lift

#time_spent[[x,y]], x menandakan waktu yang dibutuhkan untuk memenuhi permintaan, sementara y menandakan jenis gerak yang dilalui (lebih rinci dibawa

print("================================================================================")
print("\t\tSELECT MODES::")
print("\t\t1. INPUT REAL")
print("\t\t2. SIMULATION")
mode = AskInput(0, "Input: ", (1,2))
print("================================================================================")
```

```
# Inisialisasi objek lift
if mode == 1:
    for i in range(lift_amount):
        print(f"For lift {i + 1}:")
        temp_floor = AskInput(0, "Floor: ", (1, floor_amount))
        temp_target = AskInput(0, "Heading to Floor: ", (1, floor_amount))
        temp_load = AskInput(0, "Load: ", (1, lift_loadlimit))
        lifts.append(lift(temp_floor, temp_load, temp_target))
elif mode == 2:
    for i in range(lift_amount):
        temp_floor = random.randint(1, floor_amount)
        temp_target = random.randint(1, floor_amount)
        temp_load = random.randint(1,lift_loadlimit)
        lifts.append(lift(temp_floor, temp_load, temp_target))

# Main loop
if mode == 1:
    while True:
        visualize_lifts(lifts, floor_amount, 'y')
        print("-- Insert Data --")
        cur_floor = input("(input xxx to stop |Enter to continue) Current floor: ")
        if cur_floor == 'xxx':  # kondisi terminasi
            break
        elif cur_floor == '':
```

```
            # Menggerakkan semua lift (ibarat 1 satuan waktu berlewat dan tidak ada yang memencet)
            for i in range(lift_amount):
                lifts[i].move()
        else:
            # Menerima input
            cur_floor = int(cur_floor)
            if cur_floor < 1 or cur_floor > floor_amount:
                print("Input isn't in the given limits (1 - " + str(floor_amount) + ")")
                cur_floor = AskInput(0, "Input Current Floor: ", (1, floor_amount))
            cur_people = AskInput(0,"Amount of people detected: ", (1,sys.maxsize))
            cur_direction = AskInput('0', "Up or Down? (u/d)", ('u','d','U','D'))
            cur_target = AskInput(0, "Heading to floor: ", (1,floor_amount))
            for i in range(lift_amount):
                # Kalkulasi waktu yang dibutuhkan setiap lift untuk memenuhi request
                if cur_direction == lifts[i].returnmove():
                    # Ketika lift bergerak searah
                    if (lifts[i].returnmove() == 'u' and cur_floor < lifts[i].floor) or (lifts[i].returnmove() == 'd' and cur_floor > lifts[i].floor)
                        # Ketika penumpang berada di luar jangkauan gerak lift (Jarak lift ke penumpang)
                        time_spent[i][0] = abs(lifts[i].target - lifts[i].floor) + abs(lifts[i].target - cur_floor)
                        time_spent[i][1] = 0
                    else:
                        # Ketika penumpang berada di dalam jangkauan gerak lift (Jarak lift ke penumpang)
                        time_spent[i][0] = abs(cur_floor - lifts[i].floor)
```

```
                    time_spent[i][1] = 1
            else:
                # Ketika lift bergerak beda arah (Lift ke target dulu, baru dari target ke penumpang)
                time_spent[i][0] = abs(lifts[i].target - lifts[i].floor) + abs(lifts[i].target - cur_floor)
                time_spent[i][1] = 2
            # time_spent[x][1] menandakan jenis gerak, 0 = searah, di luar jangkauan, 1 = searah, di dalam jangkauan, 2 = berbeda arah

            # Mengecek kondisi overload
            if lifts[i].load + cur_people > lift_loadlimit:
                # Menambah waktu sesuai dengan berapa trip yang dibutuhkan
                time_spent[i][0] += ((lifts[i].load + cur_people) // lift_loadlimit) * 2 * max(abs(cur_floor - cur_target), abs(cur_floor - lifts
        print(time_spent)
        available = True
        # Mengecek ketersediaan lift
        for i in range(len(time_spent)):
            if not lifts[i].is_used:
                fastest = i
                break
            if i == len(time_spent) - 1:
                print("NO LIFTS ARE AVAILABLE AT THE MOMENT, PLEASE WAIT")
                available = False
        # Menentukan lift yang tercepat
        if available:
            for i in range(len(time_spent)):
                if time_spent[i][0] < time_spent[fastest][0] and not lifts[i].is_used and not lifts[i].is_serviced:
                    fastest = i
            # Mengubah target sesuai dengan jenis gerak
            if time_spent[fastest][1] == 0:
                lifts[fastest].queue.append(cur_floor)
                lifts[fastest].queue.append(cur_target)
                lifts[fastest].is_used = True
            elif time_spent[fastest][1] == 1:
                if abs(lifts[fastest].target-cur_floor) < abs(cur_target-cur_floor):
                    lifts[fastest].target = cur_target
            elif time_spent[fastest][1] == 2:
                lifts[fastest].queue.append(cur_floor)
                lifts[fastest].queue.append(cur_target)
                lifts[fastest].is_used = True
            # Menambahkan ke queue untuk kasus overload
            if lifts[fastest].load + cur_people > lift_loadlimit:
                trips = (lifts[fastest].load + cur_people) // lift_loadlimit
                for i in range(trips):
                    lifts[fastest].queue.append(cur_floor)
                    lifts[fastest].queue.append(lifts[fastest].target)
                    lifts[fastest].is_used=True
            lifts[fastest].load = min(20, lifts[fastest].load + cur_people)
        # Menggerakkan semua lift
        for i in range(lift_amount):
            if lifts[i].trips_count == max_tripcount:
                lifts[i].is_serviced = True
                lifts_under_service += 1
            lifts[i].move()
        if lifts_under_service == lift_amount:
            print("Every Lifts Needs to be serviced")
            print("Operation terminated")
            break

elif mode == 2:
    desc_on = AskInput("0", "Do you want descriptions on? (y/n) ", ('y', 'Y', 'n', 'N'))
    while True:
        visualize_lifts(lifts, floor_amount, desc_on)
        cur_floor = random.randint(0,1) # 1/2 kemungkinan untuk tidak ada penumpang
        if cur_floor == '0':
            # Menggerakkan semua lift (ibarat 1 satuan waktu berlewat dan tidak ada yang memencet)
            print("NO INPUT")
            for i in range(lift_amount):
                lifts[i].move()
        else:
            cur_floor = random.randint(1,floor_amount)
            cur_people = random.randint(1,lift_loadlimit)
            cur_target = random.randint(1,floor_amount)
```

```
            if cur_target > cur_floor:
                cur_direction = 'u'
            else:
                cur_direction = 'd'
            print(f"SIMULATED INPUT: {cur_people} people from FLOOR {cur_floor} heading to FLOOR {cur_target}")
            for i in range(lift_amount):
                # Kalkulasi waktu yang dibutuhkan setiap lift untuk memenuhi request
                if cur_direction == lifts[i].returnmove():
                    # Ketika lift bergerak searah
                    if (lifts[i].returnmove() == 'u' and cur_floor < lifts[i].floor) or (lifts[i].returnmove() == 'd' and cur_floor > lifts[i].floor):
                        # Ketika penumpang berada di luar jangkauan gerak lift (Jarak lift ke penumpang)
                        time_spent[i][0] = abs(lifts[i].target - lifts[i].floor) + abs(lifts[i].target - cur_floor)
                        time_spent[i][1] = 0
                    else:
                        # Ketika penumpang berada di dalam jangkauan gerak lift (Jarak lift ke penumpang)
                        time_spent[i][0] = abs(cur_floor - lifts[i].floor)
                        time_spent[i][1] = 1
                else:
                    # Ketika lift bergerak beda arah (Lift ke target dulu, baru dari target ke penumpang)
                    time_spent[i][0] = abs(lifts[i].target - lifts[i].floor) + abs(lifts[i].target - cur_floor)
                    time_spent[i][1] = 2
                # time_spent[x][1] menandakan jenis gerak, 0 = searah, di luar jangkauan, 1 = searah, di dalam jangkauan, 2 = berbeda arah
```

```
                # Mengecek kondisi overload
                if lifts[i].load + cur_people > lift_loadlimit:
                    # Menambah waktu sesuai dengan berapa trip yang dibutuhkan
                    time_spent[i][0] += ((lifts[i].load + cur_people) // lift_loadlimit) * 2 * max(
                        abs(cur_floor - cur_target), abs(cur_floor - lifts[i].target))
            available = True
            # Mengecek ketersediaan lift
            for i in range(len(time_spent)):
                if not lifts[i].is_used:
                    fastest = i
                    break
                if i == len(time_spent) - 1:
                    print("NO LIFTS ARE AVAILABLE AT THE MOMENT, PLEASE WAIT")
                    available = False
            # Menentukan lift yang tercepat
            if available:
                for i in range(len(time_spent)):
                    if time_spent[i][0] < time_spent[fastest][0] and not lifts[i].is_used and not lifts[i].is_serviced:
                        fastest = i

                # Mengubah target sesuai dengan jenis gerak
                if time_spent[fastest][1] == 0:
```

```
                    lifts[fastest].queue.append(cur_floor)
                    lifts[fastest].queue.append(cur_target)
                    lifts[fastest].is_used = True
                elif time_spent[fastest][1] == 1:
                    if abs(lifts[fastest].target-cur_floor) < abs(cur_target-cur_floor):
                        lifts[fastest].target = cur_target
                elif time_spent[fastest][1] == 2:
                    lifts[fastest].queue.append(cur_floor)
                    lifts[fastest].queue.append(cur_target)
                    lifts[fastest].is_used = True
                # Menambahkan ke queue untuk kasus overload
                if lifts[fastest].load + cur_people > lift_loadlimit:
                    trips = (lifts[fastest].load + cur_people) // lift_loadlimit
                    for i in range(trips):
                        lifts[fastest].queue.append(cur_floor)
                        lifts[fastest].queue.append(lifts[fastest].target)
                        lifts[fastest].is_used=True
                lifts[fastest].load = min(20, lifts[fastest].load + cur_people)
            # Menggerakkan semua lift
            for i in range(lift_amount):
                if lifts[i].trips_count == max_tripcount:
                    lifts[i].is_serviced = True
```

```
                    lifts_under_service += 1
                lifts[i].move()
            if lifts_under_service == lift_amount:
                print("Every Lifts Needs to be serviced")
                print("Operation terminated")
                break

elif mode == 2:
    desc_on = AskInput("0", "Do you want descriptions on? (y/n) ", ('y', 'Y', 'n', 'N'))
    while True:
        visualize_lifts(lifts, floor_amount, desc_on)
        cur_floor = random.randint(0,1) # 1/2 kemungkinan untuk tidak ada penumpang
        if cur_floor == '0':
            # Menggerakkan semua lift (ibarat 1 satuan waktu berlewat dan tidak ada yang memencet)
            print("NO INPUT")
            for i in range(lift_amount):
                lifts[i].move()
        else:
            cur_floor = random.randint(1,floor_amount)
            cur_people = random.randint(1,lift_loadlimit)
            cur_target = random.randint(1,floor_amount)
            if cur_target > cur_floor:
                cur_direction = 'u'
            else:
                cur_direction = 'd'
```

```
print(f"SIMULATED INPUT: {cur_people} people from FLOOR {cur_floor} heading to FLOOR {cur_target}")
for i in range(lift_amount):
    # Kalkulasi waktu yang dibutuhkan setiap lift untuk memenuhi request
    if cur_direction == lifts[i].returnmove():
        # Ketika lift bergerak searah
        if (lifts[i].returnmove() == 'u' and cur_floor < lifts[i].floor) or (lifts[i].returnmove() == 'd' and cur_floor > lifts[i].floor):
            # Ketika penumpang berada di luar jangkauan gerak lift (Jarak lift ke penumpang)
            time_spent[i][0] = abs(lifts[i].target - lifts[i].floor) + abs(lifts[i].target - cur_floor)
            time_spent[i][1] = 0
        else:
            # Ketika penumpang berada di dalam jangkauan gerak lift (Jarak lift ke penumpang)
            time_spent[i][0] = abs(cur_floor - lifts[i].floor)
            time_spent[i][1] = 1
    else:
        # Ketika lift bergerak beda arah (Lift ke target dulu, baru dari target ke penumpang)
        time_spent[i][0] = abs(lifts[i].target - lifts[i].floor) + abs(lifts[i].target - cur_floor)
        time_spent[i][1] = 2
    # time_spent[x][1] menandakan jenis gerak, 0 = searah, di luar jangkauan, 1 = searah, di dalam jangkauan, 2 = berbeda arah

    # Mengecek kondisi overload
    if lifts[i].load + cur_people > lift_loadlimit:
        # Menambah waktu sesuai dengan berapa trip yang dibutuhkan
        time_spent[i][0] += ((lifts[i].load + cur_people) // lift_loadlimit) * 2 * max(
            abs(cur_floor - cur_target), abs(cur_floor - lifts[i].target))
```

```
available = True
# Mengecek ketersediaan lift
for i in range(len(time_spent)):
    if not lifts[i].is_used:
        fastest = i
        break
    if i == len(time_spent) - 1:
        print("NO LIFTS ARE AVAILABLE AT THE MOMENT, PLEASE WAIT")
        available = False
# Menentukan lift yang tercepat
if available:
    for i in range(len(time_spent)):
        if time_spent[i][0] < time_spent[fastest][0] and not lifts[i].is_used and not lifts[i].is_serviced:
            fastest = i

    # Mengubah target sesuai dengan jenis gerak
    if time_spent[fastest][1] == 0:
        lifts[fastest].queue.append(cur_floor)
        lifts[fastest].queue.append(cur_target)
        lifts[fastest].is_used = True
    elif time_spent[fastest][1] == 1:
        if abs(lifts[fastest].target - cur_floor) < abs(cur_target - cur_floor):
            lifts[fastest].target = cur_target
    elif time_spent[fastest][1] == 2:
```

```
        lifts[fastest].queue.append(cur_floor)
        lifts[fastest].queue.append(cur_target)
        lifts[fastest].is_used = True
    # Menambahkan ke queue untuk kasus overload
    if lifts[fastest].load + cur_people > lift_loadlimit:
        trips = (lifts[fastest].load + cur_people) // lift_loadlimit
        for i in range(trips):
            lifts[fastest].queue.append(cur_floor)
            lifts[fastest].queue.append(lifts[fastest].target)
            lifts[fastest].is_used = True
    lifts[fastest].load = min(20, lifts[fastest].load + cur_people)
# Menggerakkan semua lift
for i in range(lift_amount):
    if lifts[i].trips_count == max_tripcount:
        lifts[i].is_serviced = True
        lifts_under_service += 1
    lifts[i].move()
if lifts_under_service == lift_amount:
    print("Every Lifts Needs to be serviced")
    print("Operation terminated")
    break
time.sleep(2)
```

2.6. Dokumentasi Program

2.6.1 Deskripsi Program

Program ini mensimulasikan operasi beberapa lift di sebuah gedung berdasarkan permintaan pengguna. Pengguna dapat memberikan input mengenai jumlah lift, jumlah lantai, kapasitas muatan tiap lift, serta permintaan naik/turun. Sistem akan menghitung dan menentukan lift terbaik untuk memenuhi permintaan tersebut berdasarkan kondisi dan waktu tempuh yang paling optimal.

2.6.2 Kamus Variabel

- title card (str): Judul besar program.
- lifts (list): Daftar objek lift, di mana setiap objek memiliki atribut berikut:
  - floor (int): Posisi lantai lift saat ini.
  - load (int): Jumlah orang di dalam lift saat ini.
  - target (int): Lantai tujuan lift saat ini.
  - is_used (bool): Indikator apakah lift sedang digunakan.
  - queue (list): Daftar lantai yang harus dikunjungi lift (jika ada permintaan tambahan).
  - trips_count (int): Jumlah lantai yang sudah dilalui lift.
  - is_serviced (bool): Indikator apakah lift sedang diservis
- lift_amount (int): Jumlah lift dalam gedung.
- lift_loadlimit (int): Kapasitas maksimum muatan tiap lift (dalam jumlah orang).
- floor_amount (int): Jumlah lantai di gedung.
- max_tripcount (int): Batas maksimal jumlah perjalanan sebelum lift memerlukan servis.
- mode (int): Mode operasi program (1 = input manual, 2 = simulasi).
- time_spent (list): Daftar waktu yang diperlukan setiap lift untuk memenuhi permintaan.
  - time_spent[x][0]: Waktu tempuh.
  - time_spent[x][1]: Jenis pergerakan (0 = arah sama di luar jangkauan, 1 = arah sama di dalam jangkauan, 2 = arah berbeda).
- cur_floor (int): Lantai asal pengguna yang meminta lift.
- cur_people (int): Jumlah orang yang meminta lift.
- cur_target (int): Lantai tujuan pengguna.
- lifts_under_service (int): Jumlah lift yang sedang diservis.
- available (bool): Indikator apakah ada lift yang tersedia.
- fastest (int): Indeks lift tercepat untuk memenuhi permintaan

2.6.3 Fungsi Utama

1. lift.returnmove()
   Menentukan arah gerakan lift:
   - 's': Diam.
   - 'u': Naik.
   - 'd': Turun.

2. lift.move()
   Menggerakkan lift menuju lantai target:
   - Mengubah posisi floor sesuai arah gerakan.
   - Menambah jumlah perjalanan trips_count.
   - Mengatur target baru dari queue jika lift berhenti di target sebelumnya.

3. visualize_lifts(lifts, floor_amount, desc_on)
   Memvisualisasikan posisi lift di setiap lantai gedung, serta menampilkan status detail setiap lift (jika desc_on diaktifkan).

4. AskInput(check, prompt, limits)
   Meminta input pengguna dan memastikan input sesuai batasan yang ditentukan:

- check: Tipe data yang diharapkan (integer/string).
- prompt: Pesan yang ditampilkan kepada pengguna.
- limits: Rentang atau daftar nilai yang valid untuk input.

2.6.4 Alur Program

1. Inisialisasi
   - Menampilkan titlecard.
   - Meminta input awal:
     - Jumlah lift, jumlah lantai, kapasitas maksimum muatan lift, dan batas perjalanan sebelum servis.
     - Mode operasi: Input manual atau Simulasi otomatis.
   - Membuat objek lift berdasarkan jumlah lift yang dimasukkan.
   - Jika mode simulasi, posisi, target, dan muatan lift diatur secara acak.

2. Operasi (Main Loop)
   - Mode Input Manual
     Pengguna memasukkan: Lantai asal, jumlah orang, arah tujuan (naik/turun), dan lantai tujuan.
   - Sistem menghitung waktu tempuh untuk setiap lift dan memilih lift tercepat.
   - Jika lift sedang penuh, sistem akan mengatur perjalanan tambahan ke dalam queue.
   - Sistem menggerakkan semua lift dan memperbarui status.
   - Mode Simulasi Otomatis
     - Sistem menghasilkan permintaan secara acak (lantai asal, jumlah orang, lantai tujuan).
     - Perhitungan waktu tempuh dan pemilihan lift dilakukan otomatis.
     - Sistem menggerakkan semua lift dan memperbarui status secara berkala.
   - Penghentian Operasi
     Jika semua lift memerlukan servis, program akan berhenti dan menampilkan pesan terminasi.

2.6.5 Fitur Utama

1. Visualisasi Posisi Lift: Menampilkan posisi dan status lift dalam bentuk tabel.
2. Simulasi Otomatis: Memungkinkan pengguna untuk mengamati operasi lift tanpa input manual.
3. Penanganan Overload: Menambahkan perjalanan tambahan jika jumlah penumpang melebihi kapasitas lift.
4. Servis Otomatis: Lift yang melebihi batas perjalanan akan dihentikan sementara untuk servis.

2.6.6 Catatan Penggunaan

- Mode Input Manual cocok untuk pengujian dengan skenario tertentu.
- Mode Simulasi berguna untuk pengamatan sistem dalam kondisi acak.
- Pastikan untuk memasukkan jumlah lantai dan kapasitas muatan yang realistis sesuai kebutuhan simulasi.

2.6.7 Apa yang telah ditambahkan

1. Mode simulasi, dimana input terkait request yang terjadi merandomkan, guna menguji sistem lift yang ada
2. Sistem servis otomatis lift

BAB 3
PENUTUP

3.1 Kesimpulan

1. Memenuhi Tugas Besar 1:
   Proyek ini berhasil diselesaikan sebagai bagian dari Tugas Besar 1 mata kuliah Berpikir Komputasional, dengan mengintegrasikan konsep algoritma, dekomposisi masalah, dan pengenalan pola untuk menciptakan solusi yang efektif dan aplikatif.
2. Sistem Lift yang Efisien:
   Sistem kerja lift yang dikembangkan mampu memaksimalkan efisiensi operasional dengan memilih lift tercepat berdasarkan waktu tempuh, arah gerak, dan kapasitas. Hal ini meminimalkan waktu tunggu penumpang dan memastikan pengalaman pengguna yang lebih baik.

3.2 Lesson Learned

Dari tugas ini, kami belajar pentingnya menerapkan dekomposisi masalah untuk memecah persoalan kompleks menjadi bagian-bagian yang lebih kecil dan mudah dikelola, seperti menghitung waktu tempuh, menentukan arah lift, dan memperhitungkan kapasitas muatan. Dengan pendekatan ini, setiap komponen dapat diselesaikan secara terpisah dan lebih efisien. Kami juga memahami bahwa abstraksi sangat diperlukan untuk menyederhanakan masalah, misalnya dengan fokus pada aspek-aspek utama seperti posisi lift, arah gerak, dan jumlah muatan, tanpa terlalu terpengaruh oleh detail teknis yang tidak relevan. Pendekatan ini membantu kami menciptakan sistem yang lebih sederhana namun tetap mampu menangani skenario yang kompleks.

Kami juga belajar tentang pentingnya mengenali pola dalam perilaku sistem lift, seperti pola pergerakan lift yang searah dengan permintaan penumpang, yang dapat dimanfaatkan untuk meningkatkan efisiensi waktu. Selain itu, kami menyadari bahwa algoritma yang dirancang dengan baik adalah kunci untuk memastikan sistem dapat mengambil keputusan yang optimal secara real-time. Dengan menggabungkan dekomposisi masalah, abstraksi, pengenalan pola, dan algoritma yang efisien, kami berhasil membangun simulasi sistem lift yang mampu memberikan solusi praktis dan bermanfaat. Hal ini menunjukkan pentingnya penerapan konsep berpikir komputasional dalam pengembangan perangkat lunak untuk menyelesaikan permasalahan sehari-hari.

3.3 Pembagian Tugas

1. **Yusuf Faishal Listyardi (19624217)**
   a. Flowchart
   b. Edit Video
   c. BAB 2
2. **Jonathan Alveraldo Bangun (19624234)**
   a. BAB 3
   b. Kata pengantar
   c. Daftar pustaka
   d. Abstraksi dan pengenalan pola
3. **Kayla Fiyaza Nawal Zaghbi (19624249)**
   a. PPT
   b. BAB 1
   c. Dekomposisi
   d. Source code (sangat sedikit)
4. **Muhammad Okto Huzainy (19624263)**
   a. Flowchart
   b. Edit video presentasi
   c. BAB 2
5. **Natanael I. Manurung (19624283)**
   a. Source code
   b. Dokumentasi program

Link Source Code: https://bit.ly/4hVrDGz
Link PPT Laporan Akhir: https://bit.ly/3UX2Ece
Link Video Presentasi : Video Presen Berkom - Google Drive

DAFTAR PUSTAKA

Elevator World Inc. (2020). *The Vertical Transportation Handbook* (4th ed.). Wiley.

He, Z., & Zhang, X. (2018). Optimization of Elevator Scheduling Using Real-Time Data. *Journal of Transportation Engineering, Part A: Systems, 144*(7), 04018038.

Google. (2024). Google Colab: Documentation. Retrieved from https://colab.research.google.com/

Python Software Foundation. (2024). Python Documentation. Retrieved from https://www.python.org/doc/

Manikandan, S., & Varun, M. (2020). A Real-Time Elevator Monitoring System Using IoT. *Journal of Modern Technology & Engineering, 7*(2), 87-94.

Van Steen, M., & Tanenbaum, A. S. (2017). *Distributed Systems* (3rd ed.). Pearson.

Gaddis, T. (2021). *Starting Out with Python* (5th ed.). Pearson.