Penerapan Algoritma Program Dinamis dalam Penyelesaian *Multitasking Problem*

Ahmad Hasan Albana - 13522041
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): ahmadalbana98@gmail.com

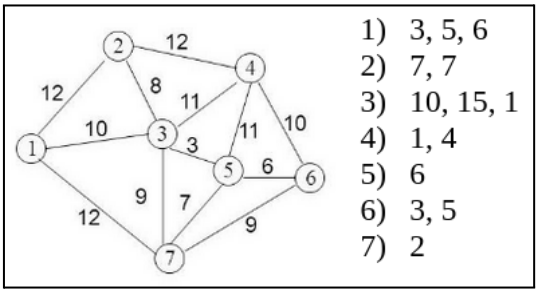
***Abstract*—Multitasking Problem adalah salah satu persoalan optimasi rute yang memiliki tujuan melakukan seluruh pekerjaan yang harus dikerjakan secara terurut dan memiliki waktu penyelesaian pekerjaan masing-masing sehingga seluruh pekerjaan dapat dilakukan dalam waktu yang seminimal mungkin . Multitasking Problem dapat direpresentasikan oleh suatu graf yang setiap simpulnya dapat memiliki beberapa pekerjaan dan waktu penyelesaiannya. Makalah ini membahas tentang cara penyelesaian Multitasking Problem yang akan dianalisis dengan menggunakan algoritma program dinamis.**

Keywords—Multitasking Problem, Program Dinamis, Graf, Optimasi.

I. Pendahuluan

Multitasking Problem adalah salah satu persoalan optimasi rute dimana seseorang perlu mencari cara paling efektif untuk menyelesaikan beberapa pekerjaan ‘otomatis’ yang perlu diselesaikan pada tempat tertentu dan memiliki waktu penyelesaiannya masing-masing sehingga seluruh pekerjaan dapat diselesaikan dengan waktu paling minimum. Pekerjaan ‘otomatis’ merujuk kepada suatu pekerjaan yang hanya memerlukan suatu pemicu diawal pekerjaan sehingga pekerjaan tersebut dapat selesai dengan sendirinya pada waktu tertentu dan perlu dikunjungi kembali untuk mendapatkan hasilnya.

Multitasking Problem dapat digambarkan dalam bentuk suatu graf dengan sisi yang nilainya mewakili jarak antar tempat dan simpul yang mewakili suatu tempat. Suatu simpul memiliki sebuah urutan dari waktu penyelesaian pekerjaan pada simpul tersebut dan juga waktu yang tersisa hingga pekerjaan yang sedang dilakukan pada simpul tersebut selesai.



***Gambar 1**: Representasi Graf dari Multitasking Problem*

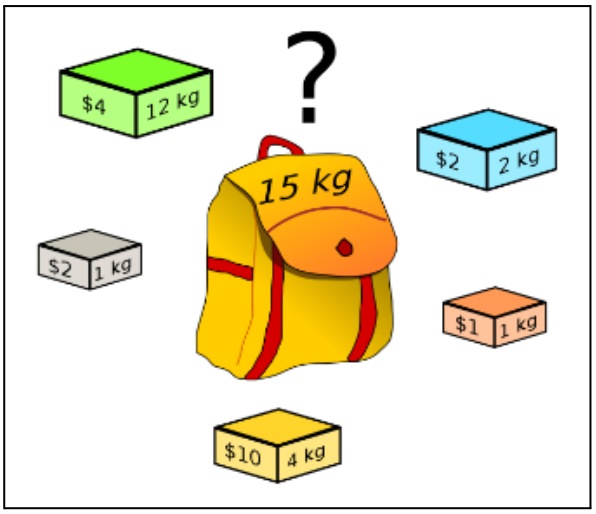
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik /2020-2021/Program-Dinamis-2020-Bagian2.pdf, diakses pada 12 Juni 2024)

Dalam pemilihan suatu simpul, diperhatikan urutan dari simpul-simpul yang dipilih karena status dari suatu simpul pada graf dapat berbeda-beda tergantung dengan rute yang telah dipilih sebelumnya. Oleh karena itu, pencarian rute optimal pada masalah ini dapat diselesaikan menggunakan algoritma Program Dinamis

II. Landasan Teori

A. Persoalan Optimasi

Persoalan optimasi adalah suatu persoalan yang membutuhkan solusi yang paling banyak atau paling sedikit untuk suatu parameter optimasi tertentu. Terdapat beberapa contoh persoalan optimasi yang sudah umum seperti *Integer Knapsack Problem* yang memiliki parameter optimasi berupa keuntungan yang didapat paling maksimal dan *Travelling Salesman Problem* yang memiliki parameter optimasi berupa jarak yang dilalui paling minimum.



***Gambar 2**:* Ilustrasi *Integer Knapsack Problem*

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf, diakses pada 12 Juni 2024)

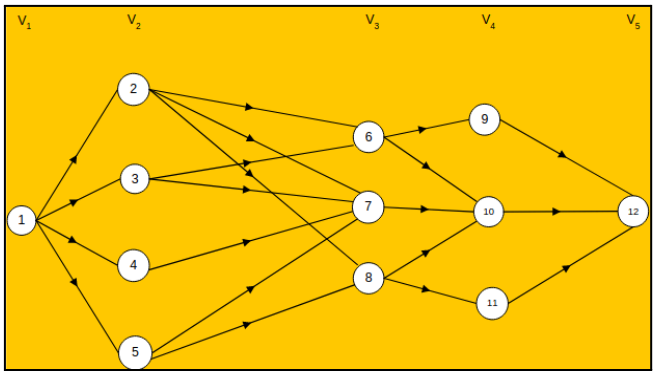
Persoalan optimasi dapat diselesaikan menggunakan beberapa algoritma yang menjamin suatu keoptimalan dari solusi yang dihasilkannya. Contohnya adalah algoritma A* yang memastikan keoptimalan hasilnya dengan terus melakukan pengecekan terhadap simpul atau jalur yang memiliki nilai dari parameter optimasi sementara yang paling optimal dan algoritma Program Dinamis yang memastikan keoptimalan hasilnya dengan memilih pilihan tahap selanjutnya dengan cara memilih pilihan paling optimal pada tahap yang sekarang.

B. Program Dinamis

Berdasarkan pendapat Rinaldi Munir, Program Dinamis adalah algoritma pemecahan persoalan optimasi dengan cara menguraikan persoalan menjadi sekumpulan tahap sedemikian sehingga solusi persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan.

Solusi yang dihasilkan pada Program Dinamis dibuat berdasarkan Prinsip Optimalitas dari solusi persoalan tersebut. Prinsip Optimalitas adalah prinsip yang memastikan bahwa jika solusi total optimal, maka bagian solusi sampai tahap ke-*k* juga optimal. Oleh karena itu, dengan berdasarkan Prinsip Optimalitas, dapat dihasilkan solusi dari tahap ke-(*k+1*) dengan berdasarkan pada solusi optimal yang dihasilkan pada tahap ke-*k*.

Persoalan pada program dinamis dapat digambarkan kedalam suatu bentuk graf berarah bernama graf multitahap yang setiap simpul pada tahap ke-*k* pasti akan selalu mengarah ke simpul pada tahap ke-(*k+1*).



***Gambar 3**:* Contoh Graf Multitahap pada Program Dinamis

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian2.pdf, diakses pada 12 Juni 2024)

Persoalan optimasi dapat diselesaikan menggunakan algoritma program dinamis dengan langkah-langkah sebagai berikut.

1. Menganalisis struktur solusi optimal dengan cara melihat banyaknya tahap yang perlu dilalui untuk mencapai solusi optimal dan simpul-simpul yang berada pada tiap solusinya.
2. Mendefinisikan solusi optimal dalam suatu fungsi atau hubungan rekursif sehingga terlihat hubungan antar tahap.
3. Menghitung solusi optimal pada setiap tahap dengan berdasarkan pada fungsi atau hubungan rekursif yang telah dibuat sebelumnya.
4. Melakukan rekonstruksi solusi optimal dengan menggabungkan setiap simpul yang paling optimal pada tahapnya masing-masing.

Penghitungan solusi optimal dapat dilakukan dengan 2 buah cara, yaitu penghitungan solusi program dinamis secara maju dengan mulai bergerak dari tahap 1 hingga ke tahap *n*, dan penghitungan solusi program dinamis secara mundur dengan mulai bergerak dari tahap ke *n* hingga ke tahap 1.

III. Pembahasan

A. Analisis Komponen Multitasking Problem

Multitasking Problem adalah salah satu persoalan optimasi rute dimana seseorang perlu mencari cara paling efektif untuk menyelesaikan beberapa pekerjaan ‘otomatis’ yang perlu diselesaikan pada tempat tertentu dan memiliki waktu penyelesaiannya masing-masing sehingga seluruh pekerjaan dapat diselesaikan dengan waktu paling minimum. Pekerjaan ‘otomatis’ merujuk kepada suatu pekerjaan yang hanya memerlukan suatu pemicu diawal pekerjaan sehingga pekerjaan tersebut dapat selesai dengan sendirinya pada waktu tertentu dan perlu dikunjungi kembali untuk mendapatkan hasilnya.

Multitasking Problem dapat digambarkan dalam bentuk suatu graf berarah dengan sisi yang menunjuk dari simpul A ke simpul B mewakili waktu yang diperlukan untuk berpindah dari simpul A ke simpul B. Setiap simpul memiliki sebuah urutan dari waktu penyelesaian pekerjaan pada simpul tersebut dan juga waktu yang tersisa hingga pekerjaan yang sedang dilakukan pada simpul tersebut selesai.

Terdapat beberapa aturan atau asumsi yang perlu diperhatikan pada *Multitasking Problem* sebagai berikut.

1. Waktu yang diperlukan dari simpul A ke simpul B secara langsung pasti minimal atau lebih kecil daripada total waktu dari simpul A ke simpul C lalu ke simpul B untuk sembarang simpul.
2. Pengunjungan pada simpul yang masih memiliki waktu tersisa untuk melakukan pekerjaan dianggap akan menunggu pekerjaan tersebut selesai pada simpul yang dikunjungi dan akan memicu pekerjaan selanjutnya untuk mulai.
3. Pengunjungan simpul hanya dapat dilakukan kepada simpul yang masih memiliki pekerjaan yang harus dilakukan atau setidaknya simpul yang hasil

pekerjaannya masih perlu diambil sehingga perlu dikunjungi.

4. Waktu yang diperlukan dari simpul A ke simpul B pasti memiliki waktu yang valid atau lebih dari sama dengan 0 sehingga selalu dapat dilakukan pengunjungan dari simpul A ke simpul B untuk sembarang simpul selama memenuhi aturan atau asumsi ke-3.

5. Setiap simpul yang ada pada minimal memiliki 1 buah pekerjaan untuk diselesaikan.

Adapun aturan atau asumsi yang dijelaskan diatas dibuat hanya untuk menyederhanakan persoalan yang secara umum sehingga program hanya akan fokus dalam penerapan algoritma program dinamis terhadap komponen utama dari *Multitasking Problem*.

Akan dibuktikan bahwa untuk setiap aturan atau asumsi diatas pasti akan dapat selalu berlaku untuk sembarang kasus yang ada.

1. Jika waktu yang diperlukan dari simpul A ke simpul B tidak minimal, maka nilai dari waktu yang diperlukan dari simpul A ke simpul B dapat diperbaharui dengan waktu minimumnya dan dilakukan pencatatan terhadap jalur dari simpul A ke simpul B yang minimum sehingga setiap dilakukan perpindahan dari simpul A ke simpul B akan dianggap melakukan perpindahan melalui jalur dari simpul A ke simpul B yang minimum.

2. Dikarenakan oleh aturan atau asumsi pertama berlaku, maka pengunjungan suatu simpul tanpa melakukan apapun pasti akan menghasilkan solusi yang tidak efektif. Oleh karena itu, boleh dianggap bahwa aturan atau asumsi ke-2 berlaku untuk mengefisiensikan jumlah pengunjungan simpul yang ada.

3. Sama seperti alasan pada poin ke-2, pengunjungan simpul dan tidak melakukan apapun akan menghasilkan solusi tidak efektif, sehingga pengunjungan simpul yang tidak memiliki pekerjaan lagi pasti menghasilkan solusi yang tidak efektif.

4. Jika tidak ditemukan jalur yang menghubungkan dari simpul A ke simpul B, maka akan dilakukan modifikasi seperti yang dilakukan pada alasan poin ke-1 sehingga didapatkan waktu tempuh dari simpul A ke simpul B minimum dengan melalui simpul yang lainnya.

5. Dikarenakan aturan atau asumsi ke-3 berlaku, maka suatu simpul yang tidak memiliki pekerjaan pasti tidak akan pernah dikunjungi sehingga dapat dianggap menjadi tidak ada.

Jadi, untuk sembarang kasus *Multitasking Problem* yang ada, perlu dilakukan *preprocessing* terlebih dahulu sehingga dihasilkan kasus *Multitasking Problem* yang memenuhi aturan atau asumsi diatas.

Untuk setiap *Multitasking Problem* yang dibicarakan pada makalah ini selanjutnya adalah *Multitasking Problem* yang merujuk pada kasus yang telah dilakukan *preprocessing* atau memenuhi aturan atau asumsi yang disebutkan pada poin sebelumnya.

B. Penerapan Program Dinamis

Dalam melakukan konversi dari suatu permasalahan optimasi kedalam bentuk algoritma Program Dinamis, perlu dilakukan beberapa tahap analisis terlebih dahulu seperti yang dijelaskan pada bagian Landasan Teori.

Langkah pertama yang perlu dilakukan adalah menganalisis struktur solusi pada *Multitasking Problem*, yaitu penentuan total tahap yang ada dan juga simpul-simpul yang ada pada setiap tahapnya.

Pada *Multitasking Problem*, pengunjungan simpul pada graf berarah yang merepresentasikan *Multitasking Problem* terjadi sebanyak $(k+n)$ kali pengunjungan, dengan k adalah banyaknya pekerjaan yang perlu diselesaikan dan n adalah banyaknya simpul yang ada. Nilai tersebut didasarkan pada setiap pengunjungan yang pasti akan memulai satu pekerjaan yang lain dan untuk setiap simpul perlu dilakukan pengambilan hasil pekerjaan untuk pekerjaan terakhir pada setiap simpul tersebut.

Untuk simpul pada setiap tahapnya adalah simpul-simpul yang dapat dikunjungi dari simpul-simpul yang ada pada tahap sebelumnya dengan simpul pada tahap pertama adalah simpul awal tempat mulai, sehingga memungkinkan bagi suatu simpul untuk berada pada 2 tahap atau lebih sekaligus di graf multitahap. Adapun simpul yang dapat dikunjungi adalah seluruh simpul yang masih memiliki pekerjaan yang harus dilakukan.

Langkah selanjutnya adalah menentukan solusi optimal dalam suatu fungsi atau hubungan rekursif. Pada *Multitasking Problem*, solusi optimal dapat ditentukan dengan fungsi rekursif berikut.

$$f(MT,\ MV,\ i)\ =\ min\{\ C_{ij} + V_j + f(MT_j\ ,\ MV_j\ ,\ j)\ \},\ j\ \in\ S$$

$$f([0,\ ...,\ 0],\ [[],\ ...,\ []],\ i)\ =\ 0 \quad \textit{... (basis)}$$

, dengan MT adalah kondisi pekerjaan yang sedang dilakukan pada setiap simpul, MT_j adalah nilai MT terbaru yang telah dikurangi dengan waktu yang telah berlangsung setelah mengunjungi simpul j, MV adalah urutan pekerjaan tersisa pada setiap simpul, MV_j adalah nilai MV terbaru dengan menghilangkan pekerjaan yang telah diselesaikan saat mengunjungi simpul j, C_{ij} adalah waktu tempuh dari simpul i ke simpul j, V_j adalah waktu yang tersisa untuk menyelesaikan pekerjaan yang sedang berlangsung pada simpul j, dan S adalah himpunan simpul yang dapat dikunjungi dari simpul i.

Setelah ditentukan fungsi atau hubungan rekursif antar tahap pada *Multitasking Problem*, dapat dilanjutkan dengan melakukan perhitungan berdasarkan fungsi rekursif yang telah dirumuskan. Penghitungan dapat dilakukan secara maju maupun mundur. Setelah penghitungan telah mencapai kondisi basis, maka akan dilakukan *backtracking* untuk merekonstruksi ulang solusi yang telah ditemukan.

C. *Implementasi pada Kode Program*

```python
# fungsi rekursif me-return nilai minimum dan path yang harus dilalui
def rekursif(n:int, mt: List[int], mv: List[List[int]], nodeNow: int) -> Tuple[int, List[int]]:
    listNode = availNode(n, mv)

    if (listNode == []): # Basis: saat seluruh pekerjaan telah selesai
        return [0, [nodeNow]]

    min_time = math.inf
    min_path = []
    for nodeDikunjungi in listNode: # mengecek setiap jalur yang bisa dikunjungi
        time = 0
        time += mk[nodeNow][nodeDikunjungi] # menambah waktu perjalanan dari nodeNow ke nodeDikunjungi

        tempMT = copy.deepcopy(mt)
        tempMV = copy.deepcopy(mv)

        tempMT[nodeDikunjungi] -= time
        if (tempMT[nodeDikunjungi] > 0): # jika pada saat sampai nodeDikunjungi, masih ada waktu tersisa
            time += tempMT[nodeDikunjungi] # menambah waktu tersisa pada nodeDikunjungi
        tempMT[nodeDikunjungi] = 0
        tempMV[nodeDikunjungi].pop()

        for y in range(len(tempMT)): # mengurangi setiap timer dengan waktu yang telah dilalui
            tempMT[y] -= time
            if (tempMT[y] < 0):
                tempMT[y] = 0

        if (tempMV[nodeDikunjungi] != []): # mengupdate timer nodeDikunjungi dengan proses yg dilakukan
            tempMT[nodeDikunjungi] = tempMV[nodeDikunjungi][-1]

        hasil = rekursif(n, tempMT, tempMV, nodeDikunjungi)
        time += hasil[0]

        if (time < min_time): # mencari waktu paling minimum
            min_time = time
            min_path = [nodeNow] + hasil[1]

    return [min_time, min_path]
```

Gambar 4: Implementasi Fungsi Rekursif Program Dinamis dari *Multitasking Problem*

(Sumber: (dokumen pribadi penulis), diakses pada 12 Juni 2024)

Fungsi rekursif diatas diawali dengan pencarian simpul-simpul yang dapat dikunjungi dari simpul ‘nodeNow’ dengan kondisi setiap simpul yang ditunjukkan oleh mv dan mt dengan mv adalah kumpulan list pekerjaan yang belum selesai pada setiap simpul dan mt adalah waktu tersisa pada setiap simpul hingga pekerjaan yang sedang dikerjakan pada simpul tersebut selesai.

```python
def availNode(n: int, mv: List[List[int]]) -> List[int]:
    listAvail = []
    for i in range(n):
        if mv[i] != []:
            listAvail.append(i)

    return listAvail
```

Gambar 5: Implementasi Fungsi Pencarian Simpul Tujuan

(Sumber: (dokumen pribadi penulis), diakses pada 12 Juni 2024)

Jika tidak ada simpul yang dapat dikunjungi, maka akan memasuki kondisi basis. Lalu, jika ada simpul yang dapat dikunjungi, maka akan dicari nilai minimum diantara proses pengunjungan simpul-simpul yang bisa dikunjungi tersebut. Pada setiap pengunjungan simpul tersebut, dilakukan terlebih dahulu penyesuaian nilai mv dan mt yang ada sesuai dengan definisi pada fungsi rekursif yang telah dirumuskan sebelumnya.

Selain hal tersebut, terdapat juga bagian program yang dilakukan beberapa penyesuaian terhadap algoritma program dinamis yang telah dijelaskan pada bagian sebelumnya.

Pada kode program, proses rekonstruksi solusi tidak dilakukan secara backtracking saat ditemukannya kondisi basis, atau setidaknya tidak dilakukan secara eksplisit. Proses rekonstruksi ini dilakukan langsung untuk setiap kali pemanggilan rekursif sehingga jalur yang dilalui akan terus dicatat dan dikembalikan kepada tahap sebelumnya karena algoritma diimplementasikan dengan metode mundur.

Penyesuaian lain yang dilakukan adalah menambahkan pekerjaan dengan waktu 0 pada awal list pekerjaan setiap simpul yang ada. Hal ini dilakukan karena pada awalnya setiap simpul belum memiliki pekerjaan yang sedang berlangsung, sehingga penambahan pekerjaan dengan waktu 0 pada setiap simpul dianggap sebagai pekerjaan untuk memulai suatu pekerjaan pada simpul tersebut.

```python
def read_file() -> Tuple[int, List[List[int]], List[List[int]], int]:
    file_name = input("Masukkan nama file: ")

    try:
        mk = []; mv = []
        with open("../test/" + file_name, 'r') as file:
            n = int(file.readline())
            startNode = int(file.readline())
            if (startNode > n or startNode <= 0):
                print("Start Node invalid!")
                return None

            for _ in range(n):
                line = file.readline()
                line = list(map(int, line.split(' ')))
                if (len(line) != n):
                    print("Jumlah node tidak sesuai pada matriks ketetanggan!")
                    return None
                mk.append(line)

            for _ in range(n):
                line = file.readline()
                line = list(map(int, line.split(' '))) + [0]
                mv.append(line)

        return [n, mv, mk, startNode-1]

    except FileNotFoundError:
        print(f"File `{file_name}` tidak ditemukan pada folder `test`!")
        return None
```

Gambar 6: Fungsi *Parsing* Komponen pada *Multitasking Problem*

(Sumber: (dokumen pribadi penulis), diakses pada 12 Juni 2024)

IV. Hasil Pengujian

Pada pengujian kode program, terdapat beberapa kondisi khusus yang ingin diuji. Kondisi-kondisi pengujian yang dilakukan adalah sebagai berikut.

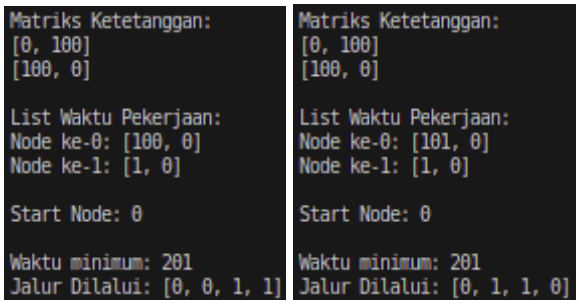
1. Kasus 1

```
Matriks Ketetanggan:
[0, 100]
[100, 0]

List Waktu Pekerjaan:
Node ke-0: [1, 0]
Node ke-1: [1, 0]

Start Node: 0

Waktu minimum: 102
Jalur Dilalui: [0, 0, 1, 1]
```

```
Matriks Ketetanggan:
[0, 100]
[100, 0]

List Waktu Pekerjaan:
Node ke-0: [1, 0]
Node ke-1: [1, 0]

Start Node: 1

Waktu minimum: 102
Jalur Dilalui: [1, 1, 0, 0]
```

Pada Kasus 1, waktu yang diperlukan untuk berpindah simpul jauh lebih besar daripada waktu yang diperlukan untuk menyelesaikan pekerjaan pada simpul. Oleh karena itu, solusi paling optimal adalah dengan melakukan perpindahan simpul sedikit mungkin sehingga langkah paling optimal adalah menyelesaikan seluruh pekerjaan pada simpul awal terlebih dahulu.

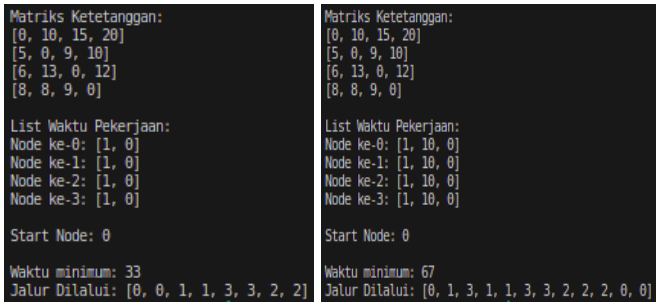
2. Kasus 2:



Pada Kasus 2, jika dilakukan analisis lebih lanjut maka akan terlihat bahwa untuk kasus yang kiri, waktu minimum yang didapat jika melalui jalur [0,0,1,1] dan [0,1,1,0] adalah sama yaitu 201. Maka dapat disimpulkan bahwa pada kasus yang kanan, jalur [0,0,1,1] akan menghasilkan waktu minimum yang lebih besar dikarenakan jalur tersebut tidak ditampilkan sebagai jalur minimum.

Untuk jalur [0,0,1,1], waktu minimum bersifat linear terhadap waktu pekerjaan pada simpul 0, dikarenakan jalur tersebut menyelesaikan pekerjaan pada simpul 0 terlebih dahulu. Sedangkan pada jalur [0,1,1,0], waktu minimum tidak bergantung pada waktu pekerjaan pada simpul 0 atau simpul awal, dikarenakan jalur tersebut meninggalkan pekerjaan tersebut untuk selesai dengan sendirinya. Akan tetapi, jalur ini akan memiliki waktu minimal 201 dan akan meningkat secara linear pada saat waktu pada simpul 0 lebih dari 201.

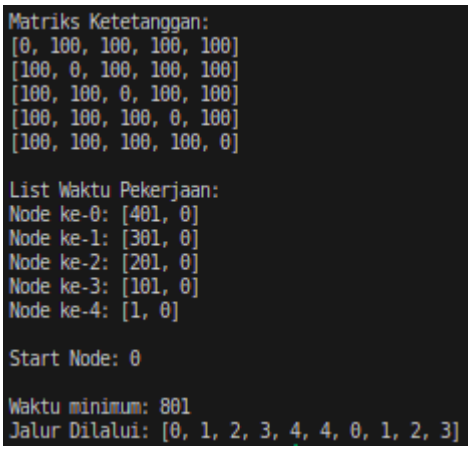
3. Kasus 3



Pada Kasus 3, jika seluruh nilai pekerjaan pada setiap simpul adalah sama dan hanya memiliki satu buah elemen, maka penyelesaian *Multitasking Problem* akan memiliki solusi yang sama dengan permasalahan optimasi TSP dengan tidak perlu kembali ke simpul awalnya.

Akan tetapi jika pekerjaan di setiap simpulnya berjumlah lebih dari 1 buah pekerjaan, maka pernyataan sebelumnya tidak lagi berlaku. Dikarenakan tidak dijaminnya bahwa setiap langkahnya hanya akan menyelesaikan seluruh pekerjaan pada simpul sekarang terlebih dahulu, melainkan bisa jadi berpindah terlebih dahulu sebelum pekerjaan pada simpul sekarang selesai seperti yang terlihat pada kasus sebelah kanan.

4. Kasus 4



Kasus 4 adalah salah satu contoh ekspansi dari Kasus 2 yang dimana setiap simpulnya berselisih 100 sesuai dengan waktu yang diperlukan untuk berpindah antar simpul. Oleh karena itu, Kasus 4 menunjukkan perilaku yang mirip, yaitu mendahulukan simpul yang memiliki pekerjaan lebih cepat yaitu simpul 4. Lalu setelah menyelesaikan simpul 4, jalur selanjutnya adalah menuju simpul yang paling lama telah dijalankan, yaitu simpul 0 dan selanjutnya. Namun pada kasus sekarang, sebenarnya setelah menyelesaikan semua pekerjaan di simpul 4, simpul apapun yang kita pilih selanjutnya tetap akan menghasilkan waktu minimum yaitu 801.

Pranala Github

Source code program dapat dilihat pada pranala berikut:

https://github.com/Bana-man/MultitaskingProblem_Solver

Pranala Youtube

Video penjelasan mengenai makalah dapat dilihat pada pranala berikut:
https://youtube.com/playlist?list=PLZHjGSNAZck6A5dnAtwiXysHNBP3tg_5h&si=G7734oWOoo6RU0yi

Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa yang telah memberikan karunia, sehingga penulis dapat Makalah IF2211 Strategi Algoritma – Sem. II Tahun 2023/2024 menyelesaikan makalah yang berjudul “Penerapan Algoritma Program Dinamis dalam Penyelesaian Multitasking Problem” yang dapat selesai tepat pada waktunya. Tak lupa juga penulis mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi, M.T yang telah membimbing kami pada mata kuliah

IF2211 Strategi Algoritma. Terakhir, penulis mengucapkan terima kasih kepada orang tua, keluarga, dan seluruh pihak yang membantu penulis dalam menyelesaikan makalah ini.

References

[1] Rinaldi Munir, "Program Dinamis (*Dynamic Programming*) Bagian 1", https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf , diakses 12 Juni 2024.

[2] Rinaldi Munir, "Program Dinamis (*Dynamic Programming*) Bagian 2", https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian2.pdf , diakses 12 Juni 2024.

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, 12 Juni 2024

Ahmad Hasan Albana, 13522041