Penerapan Algoritma Dynamic Programming untuk Menentukan Langkah Terbaik pada Permainan Ular Tangga

Muhammad Fakhry Zaki - 13524087
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: mfakhry.zaki@gmail.com , 13524087@std.stei.itb.ac.id

Abstract*—Snakes & Ladders is a simple boardgame that is usually played by 2 – 4 players. This game is a game based entirely on luck because the only action the players can do throughout the game is rolling a dice. The player's playing piece would move forward depending on the number of the dice rolled in their turn. This game is equipped with obstacles such as snakes and ladders. Ladders can make a player's playing piece to immediately move to the tile above it, whereas snakes can make a player's playing piece move to the tile below it. This paper explores the minimum steps, order of dice, and order of tile passed along the way to the finish tile using the dynamic programming strategy. The implementation is being done using c++ language. The implementation and testing result shows that the solution given by the dynamic programming algorithm is the optimal solution for all tiles within a relatively short amount of time.

***Keywords*—Dynamic programming, Snakes & Ladders, Memoization, Optimal**

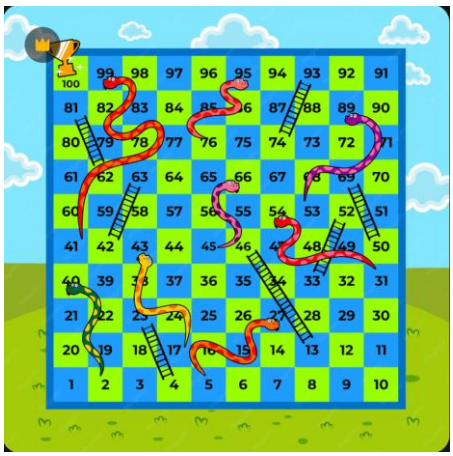
Abstrak*— Ular Tangga adalah permainan sederhana yang kerap dimainkan oleh 2 – 4 orang. Permainan ini adalah permainan yang sepenuhnya mengandalkan keberuntungan karena aksi yang bisa dilakukan pemain hanyalah melempar dadu. Pion dari seorang pemain dimajukan sejumlah hasil dadu yang didapat dari lemparannya pada gilirannya. Permainan ini dilengkapi dengan rintangan berupa ular dan tangga. Tangga dapat membuat pion pemain langsung naik ke petak atas, sedangkan ular dapat membuat pion pemain turun ke petak bawah. Makalah ini mengeksplorasi langkah minimal, urutan dadu, serta urutan petak untuk mencapai petak akhir dengan menggunakan strategi program dinamis. Implmentasi dilakukan menggunakan bahasa c++. Hasil implementasi dan pengujian yang dilakukan menunjukkan bahwa penyelesaian menggunakan strategi program dinamis ini berhasil memberikan hasil yang optimal untuk seluruh petak dalam waktu yang singkat.

***Kata kunci*—Program dinamis, Ular Tangga, Memoisasi, Optimal**

I. PENDAHULUAN

Ular Tangga adalah suatu permainan papan yang sangat sederhana. Permainan ini kerap dimainkan oleh 2 hingga 4 orang. Permainan ini sangatlah sederhana karena satu-satunya aksi yang bisa dilakukan oleh seorang pemain hanyalah melempar dadu saja. Semua pemain memiliki sebuah pion yang sama-sama mulai dari petak pertama pada awal permainan. Kemudian, secara bergantian masing-masing pemain melemparkan satu buah dadu dan pion miliknya dapat bergerak ke depan sejauh jumlah mata dadu yang didapatnya. Pemenang dari permainan ini adalah pemain yang pionnya pertama sampai ke petak akhir papan permainan.

Namun, di antara petak-petak yang dilalui oleh para pemain, terdapat dua jenis rintangan, yakni *ular* dan *tangga*. *Tangga* adalah rintangan yang bermanfaat bagi para pemain karena apabila pion dari seorang pemain berhenti pada petak yang ada *tangga*-nya, maka pion dari pemain tersebut bisa langsung memanjat ke petak di ujung atas tangga tersebut. Di sisi lain, *ular* adalah rintangan yang berbahaya bagi para pemain karena apabila pion dari seorang pemain berhenti pada petak yang ada *ular*-nya, maka pion tersebut akan terjatuh hingga ke ujung ekor ular. Kedua tantangan inilah yang membuat permainan ular tangga menjadi menegangkan. Tidak jarang momen *comeback* bisa terjadi karena pemain yang memimpin bisa tiba-tiba jatuh sangat jauh karena berhenti di petak *ular*.



Gambar 1.1. Contoh Papan Permainan Ular Tangga

Sumber : https://www.magnific.com/idn/vektor-premium/vektor-permainan-papan-ular-tangga-untuk-anak-anak_149554192.htm

Pada makalah ini, peneliti menggunakan strategi *dynamic programming* untuk mencari jumlah langkah minimal, urutan dadu, serta urutan petak yang dikunjungi untuk mencapai petak akhir dari suatu petak tertentu. Tentunya, langkah minimal ini akan sangat sulit didapatkan pada permainan sesungguhnya karena membutuhkan keberuntungan yang sangat besar. Namun, pada makalah ini pembahasan hanya difokuskan kepada pencarian langkah optimal tanpa mempertimbangkan peluang terjadinya di permainan sesungguhnya.

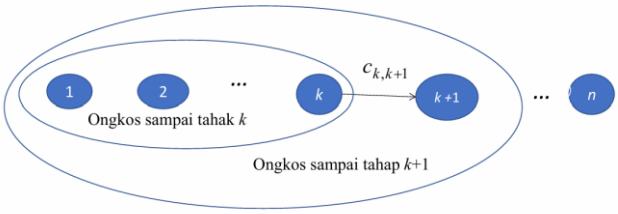
II. Dasar Teori

A. Dynamic Programming

Dynamic programming, atau dalam bahasa Indonesia disebut program dinamis, adalah salah satu strategi algoritma untuk menyelesaikan permasalahan. Strategi ini memecah masalah yang ada menjadi sekumpulan tahapan sedemikian sehingga solusi dari permasalahan tersebut bisa dipandang sebagai serangkaian keputusan atau pemilihan yang saling berkaitan.

Kata "program" pada program dinamis bukan berarti pemrograman, melainkan kata "program" tersebut memiliki makna perencanaan. Sebagaimana dalam KBBI, program artinya rancangan mengenai asas serta usaha (dalam ketatanegaraan, perekonomian, dan sebagainya) yang akan dijalankan. Lalu, kata "dinamis" merujuk kepada pencarian atau perhitungan solusi yang berkembang selama penyelesaian masalah.

Strategi program dinamis ini digunakan untuk menyelesaikan permasalahan-permasalahan optimasi, yakni minimasi atau maksimasi. Strategi ini juga menggunakan prinsip optimatilas, yakni apabila solusi dari keseluruhan permsalahan adalah optimal, maka solusi untuk sebagian permasalahan tersebut dipastikan juga optimal. Penggunaan prinsip optimalitas juga berarti dalam penyelesaian suatu permasalahan, apabila solusi untuk tahap ke-*k* sudah didapatkan dan solusi untuk tahap ke-*k+1* ingin dicari, maka pencarian solusinya bisa langsung dilanjutkan dari tahap ke-*k* tanpa harus kembali ke tahap pertama lagi.



Gambar 2.1. Ilustrasi Perpindahan Tahap pada Program Dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/25-Program-Dinamis-(2026)-Bagian1.pdf

Salah satu ide yang membedakan antara program dinamis dengan strategi-strategi yang lain adalah memoisasi. Memoisasi adalah menyimpan solusi dari permasalahan untuk suatu tahap ke dalam tabel yang disimpan di dalam memori. Penggunaan memoisasi ini lah yang membuat program dinamis bisa bekerja dengan waktu yang relatif cepat. Karena adanya memoisaasi, apabila solusi untuk tahap ke-*k* sudah ditemukan, maka solusi tersebut disimpan ke dalam tabel sehingga ketika ingin digunakan kembali bisa langsung mengambil dari tabel tersebut.

B. Persoalan Coin Change

Ada banyak sekali contoh persoalan-persoalan yang bisa diselesaikan menggunakan strategi program dinamis. Lebih lagi, program dinamis ini bisa dibilang salah satu strategi algoritma yang terbaik untuk menyelesaikan permasalahan optimasi. Program dinamis jauh lebih cepat dibandingkan *bruteforce* karena tidak perlu mencacah satu per satu keseluruhan solusi. Ditambah, penggunaan memoisasi juga mempercepat pencarian solusi secara signifikan. Program dinamis juga menjamin bahwa solusi yang didapatkan adalah benar-benar solusi yang paling optimal, berbeda dengan *greedy* yang hanya bisa menjamin mendapatkan solusi yang *pseudo*-optimal.

Salah satu contoh persoalan optimasi yang bisa diselesaikan dengan program dinamis adalah persoalan *coin change*, dalam bahasa Indonesia disebut persoalan penukaran uang. Persoalan penukaran uang berbunyi seperti ini : "Diberikan M jenis koin, masing-masing jenis bernilai $a_1,a_2,...,a_M$ rupiah. Asumsikan banyaknya koin untuk setiap nominal yang ada tak terbatas. Tentukan banyaknya koin paling sedikit untuk membayar tepat sebesar N rupiah!". Persoalan penukaran uang ini adalah persoalan opimasi minimasi karena tujuan yang ingin dicapai adalah mendapatkan jumlah koin tersedikit untuk membayarkan suatu nilai tertentu.

Untuk menyelesaikan persoalan ini, dapat digunakan dua jenis pendekatan penyelesaian program dinamis, yaitu *top-down* dan *bottom-up*. Pendekatan yang digunakan untuk penyelesaian *top-down* adalah dengan rekursi dan menyimpan hasil yang sudah dikunjungi pada memo yang biasanya disimpan sebagai variabel global. Sedangkan penyelesaian *bottom-up* biasanya menggunakan iterasi dengan menggunakan tabel dinamis yang di-*update* terus-menerus hingga mendapatkan solusi yang dicari.

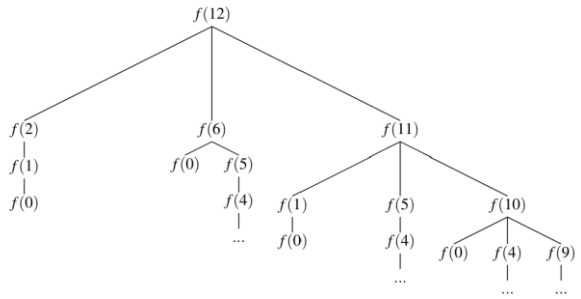
Untuk suatu persoalan yang sama, biasanya, baik pendekatan *top-down* maupun *bottom-up*, memiliki kesamaan di *basecase* persoalan dan fungsi solusi yang digunakan. Untuk permasalahan penukaran uang *basecase* nya adalah nilai dari *f(0)* sama dengan 0 karena untuk membayar uang sebesar 0 dibutuhkan 0 buah koin. Fungsi solusi untuk permasalahan penukaran uang ditunjukkan pada gambar di bawah ini.

$$f(x) = \begin{cases} 0, & x = 0 \\ \min\limits_{\substack{1 \le k \le M \\ a_k \le x}} f(x - a_k) + 1, & x > 0 \end{cases}$$

Gambar 2.2. Fungsi Solusi Permasalahan Penukaran Uang

Sumber : https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf

Pada pendekatak *top-down*, pencarian solusi dimulai dari pemanggilang fungsi untuk nilai yang dicari (*f(x)*). Kemudian, ketika fungsi tersebut membutuhkan fungsi-fungsi pada tahap sebelumnya yang belum diketahui hasilnya, maka fungsi-fungsi tahap sebelumnya tersebut diselesaiakan terlebih dahulu. Untuk penyelesaian *bottom-up*, pencarian solusi menggunakan iterasi dilakukan dari tahap satu hingga tahap *x* (nilai yang ingin dicari solusinya) sehingga ketika suatu fungsi pada tahap *k* membutuhkan informasi dari fungsi pada tahap sebelumnya, nilainya pasti sudah ada dan bisa langsung digunakan.



Gambar 2.3. Ilustrasi Pohon Rekursi untuk Pendekatan *Top-Down*

Sumber : https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf

x	0	1	2	3	4	5	6	7	8	9	10	11	12
$f(x)$	0	1	2	3	4	5							

$$\begin{aligned} f(6) &= \min(f(6-1)+1, f(6-6)+1) \\ &= \min(f(5)+1, f(0)+1) \\ &= \min(5+1, 0+1) \\ &= \min(6, 1) \\ &= 1 \end{aligned}$$

Gambar 2.4. Ilustrasi Pengisian Tabel untuk Pendekatak *Bottom-Up*

Sumber : https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf

Pada makalah ini, permasalahan yang ingin diselesaikan adalah mencari jumlah langkah minimal, urutan dadu, serta urutan petak yang dikunjungi untuk mencapai petak akhir dari suatu petak tertentu. Persoalan ini mirip dengan persoalan penukaran uang di atas. Nilai hasil lemparan dadu dari 1—6 bisa diibaratkan sebagai koin-koin yang bernilai $a_1 ... a_M$ rupiah dan jarak awal dari petak yang dicari ke petak akhir bisa diibaratkan sebagai jumlah uang yang ingin dibayarkan. Perbedaannya dari persoalan penukaran uang di atas adalah adanya rintangan berupa *ular* dan *tangga* yang dapat menyebabkan perpindahan posisi pion pemain dengan instan.

III. Implementasi dan Pengujian

Untuk implementasi, saya menggunakan bahasa c++ untuk menyelesaikan persoalan tersebut. Saya membagi program menjadi 4 buah file, yakni main (.cpp), io (.hpp/.cpp), solver (.hpp/.cpp), dan ConfigException (.hpp).

File ConfigException berisi deklarasi kelas ConfigException yang digunakan untuk error apabila konfigurasi yang dimasukkan pengguna ada yang salah.

File io digunakan untuk menerima input konfigurasi, kemudian memastikan file konfigurasi tersebut sesuai dengan format. Apabila konfigurasi yang diberikan oleh pengguna tidak sesuai, program akan men-*throw* ConfigException dan akan meminta pengguna untuk memasukan konfigurasi lain yang sesuai. Apabila konfigurasinya sudah sesuai, maka program akan menyimpan data-data dari konfigurasi ke dalam memori program. Kegunaan lain dari file ini adalah untuk menampilkan hasil program ke layar terminal.

File solver adalah file utama yang mengimplementasikan strategi program dinamis untuk menyelesaikan persoalan makalah ini. Terdapat dua fungsi utama, yakni solve() dan startSolve(). Fungsi solve() adalah fungsi rekursif untuk mencari langkah terbaik untuk setiap petak. Fungsi startSolve() adalah *entrypoint* untuk menjalankan fungsi solve() pada setiap petak.

Terakhir, file main adalah file yang menjadi *entrypoint* utama untuk keseluruhan program. Program dimulai dari penggunaan fungsi readAndVerifyFile() dari file io untuk menerima masukan konfigurasi dari pengguna. Selanjutnya program memanggil startSolve() untuk mengomputasi hasil dari setiap petak. Terakhir, program kembali memanggil fungsi dari file io, yakni printPapan() dan tampilkanPanel() untuk menampilkan rangkuman penyelesaian program dan menampilkan CLI interaktif untuk bisa digunakan oleh pengguna.

Dari keempat file tersebut, berikut adalah beberapa komponen utama dalam program ini.

A. *Kelas ConfigException*

Kelas ConfigException ini didefinisikan pada file ConfigException. Kelas ini adalah turunan dari kelas exception di c++ dan bertanggungjawab untuk menerima error konfigurasi dari fungsi readAndVerifyFile() pada file io. Konstruktor kelas ini menerima pesan error dan baris terjadinya error tersebut.

```
class ConfigException : public exception {
protected:
    string message;
    int line;

public:
    ConfigException(const string& message, const int& line) : message(message), line(line) {}
    string toString () {
        return "Config error pada line " + to_string(line) + ": " + message;
    }
};
```

Gambar 3.1. Definisi Kelas ConfigException

B. *Kelas PetakInfo*

Karena pada makalah ini, ada tiga persoalan yang ingin diselesaikan, yakni pencarian jumlah langkah minimal, urutan dadu hingga ke petak akhir, serta urutan petak yang dikunjungi, maka ketiga informasi tersebut dienkapsulasi dalam satu kelas PetakInfo.

```cpp
class PetakInfo {
public:
    int minStep;
    string urutanDadu;
    string rute;
    PetakInfo()
        : minStep(NILAI_MAXIMAL), urutanDadu(""), rute("") {}

    PetakInfo (int step, string dadu, string rut)
        : minStep(step), urutanDadu(dadu), rute(rut) {}
};

extern PetakInfo petak[1000000];
```

Gambar 3.2. Definisi Kelas PetakInfo

Variabel global, array PetakInfo, petak masing-masing menyimpan informasi pada petak tersebut di papan permainan ular tangga. Array petak inilah yang akan diolah pada fungsi di file solver nantinya.

C. *Konfigurasi Permainan Ular Tangga*

Untuk memudahkan program, format konfigurasi untuk papan ular tangga dibuat sesederhana mungkin. Informasi yang dimuat dalam konfigurasi antara lain ukuran panjang dan lebar papan, jumlah tangga dan petak-petak asal dan tujuan dari tiap tangga, serta jumlah ular dan petak-petak asal dan tujuan dari tiap ular.

```
10 10
7
4 23
13 46
33 52
42 63
50 69
62 81
74 93
8
27 5
40 2
43 17
54 31
66 45
89 53
95 76
99 41
```

Gambar 3.3. Contoh Konfigurasi Papan Ular Tangga

Baris pertama pada contoh konfigurasi tersebut menunjukkan ukuran tinggi dan lebar dari papan permainan. Baris selanjutnya menunjukkan jumlah tangga yang ada pada papan disimpan pada variabel *tanggaCount*. Kemudian, sebanyak *tanggaCount* baris berikutnya mendefinisikan masing-masing tangga dengan nilai yang lebih kecil menunjukkan pangkal (asal) tangga dan nilai yang lebih besar menunjukkan ujung (akhir) tangga. Selanjutnya, baris selanjutnya menunjukkan jumlah ular yang ada pada papan disimpan dalam variabel *ularCount*. Terakhir, sebanyak *ularCount* baris terakhir menunjukkan masing-masing ular yang ada dengan nilai yang lebih besar menunjukkan mulut ular dan nilai yang lebih kecil menunjukkan ekor ular.

D. *Fungsi Solve()*

Fungsi solve() ini didefinisikan di file solver dan digunakan untuk mencari penyelesaian dari permasalahan pada makalah ini. Di sinilah strategi program dinamis diterapkan. Strategi program dinamis yang digunakan adalah *top-down* dengan menggunakan rekursi untuk mencari solusi optimal dari petak tersebut. Memoisasi disimpan di variabel petak yang didefinisikan di bagian PetakInfo sebelumnya.

```cpp
PetakInfo solve(int idxPetak) {
    if (idxPetak > UkuranPetak) {
        return PetakInfo(NILAI_MAXIMAL, "", "");
    }

    if (idxPetak == UkuranPetak) {
        return PetakInfo(0, "", " " + to_string(idxPetak));
    }

    if (petak[idxPetak].minStep != NILAI_MAXIMAL) {
        return petak[idxPetak];
    }

    if (visiting[idxPetak]) {
        return PetakInfo();
    }

    visiting[idxPetak] = true;

    int petakSekarang = idxPetak;

    if (Tangga.count(idxPetak)) {
        petakSekarang = Tangga[idxPetak];
    }
    else if (Ular.count(idxPetak)) {
        petakSekarang = Ular[idxPetak];
    }

    string prefixRute = petakSekarang != idxPetak
        ? " (" + to_string(idxPetak) + "=>" + to_string(petakSekarang) + ")"
        : " " + to_string(petakSekarang);

    PetakInfo best = PetakInfo();

    for (int dadu = 6; dadu >= 1; dadu--) {
        PetakInfo hasil = solve(petakSekarang + dadu);
        if (hasil.minStep == NILAI_MAXIMAL) continue;

        PetakInfo kandidat(
            1 + hasil.minStep,
            " " + to_string(dadu) + hasil.urutanDadu,
            prefixRute + hasil.rute
        );
        best = minimal(best, kandidat);
    }

    visiting[idxPetak] = false;
    return petak[idxPetak] = best;
}
```

Gambar 3.4. Implementasi Fungsi Solve

Baris-baris atas pada fungsi digunakan untuk *basecase* berbagai kasus, ketika mencapai suatu *basecase* maka fungsi akan langsung di-*return*. Bagian untuk mencari solusi optimal ditunjukkan oleh blok *for-loop*. Pada blok tersebut, setiap kemungkinan hasil dadu dicoba dan hasil yang terbaik disimpan dalam variabel *best*. Setelah semua kemungkinan selesai dicoba, nilai *best* adalah nilai yang paling optimal untuk petak tersebut dan nilai tersebut di-*assign* ke dalam memo petak. Ketika sedang mengunjungi (mengomputasi nilai) dari suatu petak, indeks dari petak tersebut diberi *flag visiting* agar nilai yang dihasilkan sesuai dan tidak *overlap*. *Flag visiting* tersebut kemudian dihapus (nilainya dijadikan *false*) ketika hasil optimal pada petak tersebut sudah didapatkan.

E. Ringkasan Papan dan Program

Setelah program main dijalankan, ringkasan program dan papan permainan akan ditampilkan ke layar terminal.

```
SELAMAT DATANG DI PENGHITUNG ULAR TANGGA
Masukkan file input: config
------------ Papan Permainan ------------
100     99      98      97      96      95      94      93      92      91
81      82      83      84      85      86      87      88      89      90
80      79      78      77      76      75      74      73      72      71
61      62      63      64      65      66      67      68      69      70
60      59      58      57      56      55      54      53      52      51
41      42      43      44      45      46      47      48      49      50
40      39      38      37      36      35      34      33      32      31
21      22      23      24      25      26      27      28      29      30
20      19      18      17      16      15      14      13      12      11
1       2       3       4       5       6       7       8       9       10
List Tangga:
4 -> 23
13 -> 46
33 -> 52
42 -> 63
50 -> 69
62 -> 81
74 -> 93
List Ular:
27 -> 5
40 -> 2
43 -> 17
54 -> 31
66 -> 45
89 -> 53
95 -> 76
99 -> 41
Waktu komputasi: 4517 microseconds
-----------------------------------------
```

Gambar 3.5. Tampilan Papan Permainan

Selain menampilkan papan permainan, program juga menampilkan panel interaktif yang bisa digunakan oleh pengguna.

```
List perintah :
1. Melihat detail satu petak (cmd: 1-100)
2. Melihat summary dari semua petak (cmd: summary)
3. Melihat detail dari semua petak (cmd: detail)
4. Keluar dari program (cmd: quit/exit)
>>
```

Gambar 3.6. Perintah Interaktif Program

Pengguna dapat memasukkan perintah-perintah tersebut berulang kali untuk melihat hasil penyelesaian permasalahan oleh program. Untuk menutup panel dan menyelesaikan program, pengguna bisa memasukkan perintah quit/exit.

F. Summary Keseluruhan Petak

Pada panel interaktif, apabila pengguna menggunakan perintah *summary* program akan menampilkan list langkah minimal setiap petak.

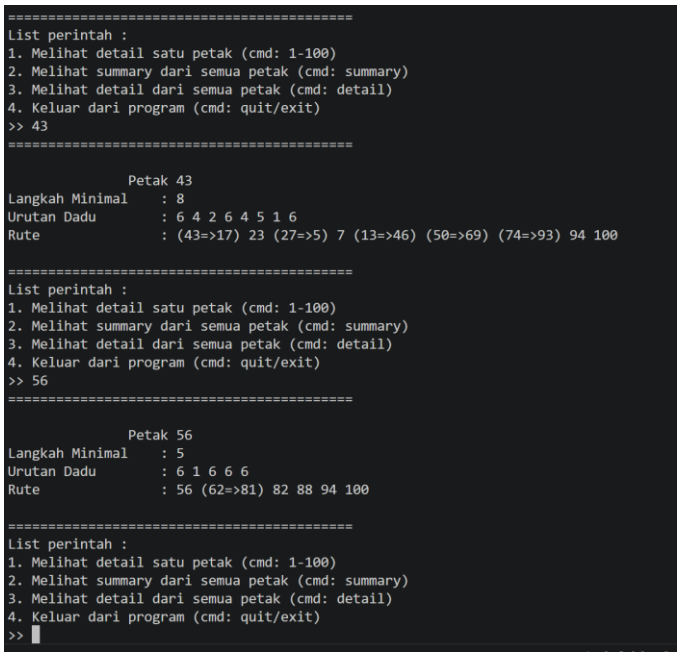
```
Petak[1] : 6
Petak[2] : 6
Petak[3] : 6
Petak[4] : 7
Petak[5] : 6
Petak[6] : 6
Petak[7] : 5
Petak[8] : 5
Petak[9] : 5
Petak[10] : 5
Petak[11] : 5
Petak[12] : 5
Petak[13] : 4
Petak[14] : 9
Petak[15] : 9
Petak[16] : 9
Petak[17] : 9
Petak[18] : 8
Petak[19] : 8
Petak[20] : 8
```

Gambar 3.7. Tampilan Langkah Optimal Menuju ke Akhir Petak dari Petak ke 1 – 20

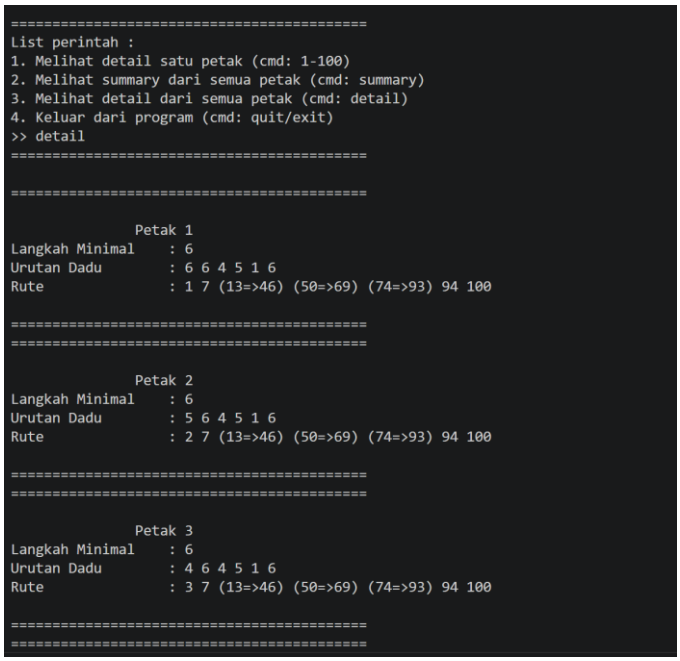
Dari hasil *summary* tersebut ada beberapa *insight* yang bisa didapat. Pertama, langkah optimal untuk petak 4, lebih besar dibanding petak 1 – 6 yang lain. Hal ini disebabkan karena petak 4 adalah pangkal tangga yang membuatnya langsung naik ke petak 23. Petak 1 – 6 bisa menaiki tangga yang lebih baik pada petak 13 (hingga ke petak 46) sehingga total langkah optimalnya lebih pendek. Kemudian, untuk alasan yang sama, petak 14 dan seterusnya membutuhkan lebih banyak langkah untuk bisa mencapai petak akhir karena sudah melewatkan tangga pada petak 13.

G. Detail Petak

Pengguna juga bisa melihat detail suatu petak (hasil dari ketiga permasalahan yang diangkat pada materi ini) dengan menggunakan perintah detail. Perintah detail akan memberikan detail untuk setiap petak dari petak awal hingga petak akhir, sedangkan apabila pengguna hanya ingin melihat detail dari salah satu petak, maka bisa langsung memasukkan angka dari petak yang ingin dilihat.



Gambar 3.8. Tampilan Detail Petak 43 dan 56 dengan Langsung Memasukkan Angka



Gambar 3.9. Tampilan Detail Petak 1 – 3 dengan Perintah Detail

IV. Kesimpulan

Dari hasil implementasi dan pengujian, bisa disimpulkan bahwa bahwa strategi program dinamis dapat mencari jumlah langkah minimal, urutan dadu, serta urutan petak yang dikunjungi untuk mencapai petak akhir dari seluruh petak pada papan permainan dengan mangkus dan sangkil. Hasil yang didapat pun menunjukkan hasil yang optimal global. Hasil ini membuktikan bahwa program dinamis adalah salah satu strategi yang sangat baik dalam penyelesaian persoalan optimasi karena menyeimbangkan antara keoptimalan absolut dengan waktu komputasi yang sangkil.

Meskipun demikian, tidak bisa dipungkiri bahwa dalam permainan ular tangga sungguhan urutan langkah terbaik ini akan sangat jarang terjadi. Ditambah lagi, dalam permainan, kita juga tidak bisa memilih dadu yang terbaik sesuai yang diinginkan karena pada permainan sesungguhnya pelemparan dadu dilakukan secara benar-benar acak.

V. Lampiran

Tautan Github :

https://github.com/MFakhryzaKi/MakalahStima

VI. Ucapan Terima Kasih

Saya ingin mengucapkan terima kasih sebesar-besarnya terutama kepada Tuhan Yang Maha Esa, Allah Swt, yang telah menganugrahi saya kemampuan untuk bisa menyelesaikan makalah ini. Selanjutnya ucapan terima kasih juga ingin saya sampaikan kepada dosen saya Bapak Prof. Dr. Ir. Rinaldi, M.T. yang telah mengajarkan saya mata kuliah Strategi Algoritma. Terakhir, saya juga ingin mengucapkan terima kasih kepada orang tua dan teman-teman saya yang senantiasa memberikan saya dukungan dalam bentuk apa pun.

References

[1] A. F. Aji & W. Gozali, “PEMROGRAMAN KOMPETITIF DASAR Panduan Memulai OSN Informatika, ACM-ICPC, dan Sederajat”. https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf (Diakses pada 14 juni 2026)

[2] Munir, Rinaldi, 2026, “Program Dinamis (Dynamic Programming) Bagian 1”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/25-Program-Dinamis-(2026)-Bagian1.pdf (Diakses pada 13 Juni 2026)

[3] Munir, Rinaldi, 2026, “Program Dinamis (Dynamic Programming) Bagian 2”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/26-Program-Dinamis-(2026)-Bagian2.pdf (Diakses pada 13 Juni 2026)

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, 15 Juni 2026

Ttd

Muhammad Fakhry Zaki

13524087