Pemanfaatan Algoritma Greedy untuk Permainan Ular Tangga

Jimly Nur Arif - 13522123
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: 13522123@std.stei.itb.ac.id

***Abstract*—Permainan ular tangga adalah permainan papan klasik yang banyak dimainkan oleh anak-anak dan mengandalkan keberuntungan untuk menentukan pemenang. Namun, dengan pendekatan yang tepat, peluang kemenangan dapat dioptimalkan. Makalah ini mengeksplorasi penerapan algoritma greedy dalam permainan ular tangga untuk memaksimalkan keuntungan pemain pada setiap langkah. Algoritma greedy adalah metode pemecahan masalah yang mengandalkan pilihan lokal optimal pada setiap langkah dengan harapan mencapai solusi global optimal. Dalam konteks permainan ular tangga, algoritma ini membantu menentukan langkah terbaik setelah setiap lemparan dadu dengan mempertimbangkan keuntungan langsung dari setiap pilihan langkah.**

Keywords—ular tangga; greedy;

I. Introduction

Permainan ular tangga adalah salah satu permainan papan tradisional yang populer di berbagai belahan dunia, terutama di kalangan anak-anak. Permainan ini sederhana, tetapi menarik karena menggabungkan elemen keberuntungan dan strategi. Setiap pemain memulai dari kotak pertama dan berusaha mencapai kotak terakhir (kotak 100) dengan melempar dadu untuk menentukan langkah yang harus diambil. Dalam perjalanan menuju kotak 100, pemain dapat mendarat di kotak-kotak yang berisi tangga atau ular. Tangga memungkinkan pemain untuk naik ke kotak yang lebih tinggi, sementara ular membuat pemain turun ke kotak yang lebih rendah.

Secara umum, permainan ular tangga mengandalkan keberuntungan, karena hasil lemparan dadu tidak dapat diprediksi. Namun, dengan pendekatan algoritmik yang tepat, seperti algoritma greedy, pemain dapat meningkatkan peluang untuk mencapai kotak terakhir lebih cepat. Algoritma greedy adalah salah satu metode pemecahan masalah yang mengandalkan pilihan lokal optimal pada setiap langkah dengan harapan mencapai solusi global optimal. Dalam konteks permainan ular tangga, algoritma ini dapat membantu pemain menentukan langkah terbaik berdasarkan posisi saat ini dan hasil lemparan dadu.

Makalah ini bertujuan untuk mengeksplorasi penerapan algoritma greedy dalam permainan ular tangga. Kami akan membahas bagaimana algoritma ini dapat digunakan untuk memilih langkah optimal pada setiap giliran, serta mengilustrasikan implementasinya dalam kode Python. Pendekatan greedy fokus pada keuntungan jangka pendek dan tidak selalu menghasilkan solusi terbaik secara keseluruhan. Namun, dalam permainan yang mengandalkan keberuntungan seperti ular tangga, pendekatan ini dapat memberikan panduan yang bermanfaat untuk memaksimalkan peluang menang.

II. Landasan Teori

A. Algoritma Greedy

Algoritma greedy adalah salah satu metode dalam pemecahan masalah yang mengandalkan pilihan lokal optimal pada setiap langkah dengan harapan mencapai solusi global optimal. Algoritma ini bekerja dengan memilih langkah yang paling menguntungkan pada saat itu tanpa mempertimbangkan konsekuensi jangka panjang. Dalam banyak kasus, algoritma greedy dapat menghasilkan solusi yang mendekati optimal atau bahkan optimal, terutama untuk masalah tertentu yang memiliki struktur khusus.

B. Prinsip Dasar Algoritma Greedy

Pilih Langkah Terbaik Saat Ini: Pada setiap langkah, algoritma memilih opsi yang tampaknya memberikan manfaat terbesar saat itu. Langkah ini haruslah yang terbaik dari semua langkah yang tersedia saat ini.

Ulangi Sampai Selesai: Proses pemilihan langkah terbaik diulangi sampai mencapai tujuan akhir atau kondisi tertentu terpenuhi. Algoritma terus bekerja hingga semua elemen masalah telah diatasi atau solusi akhir tercapai.

C. Karakteristik Algoritma Greedy

Pilihan Lokal Optimal: Algoritma greedy selalu memilih langkah yang paling menguntungkan pada saat itu tanpa memikirkan akibat jangka panjangnya. Pilihan ini disebut pilihan lokal optimal.

Tidak bisa balik ke sebelumnya: Algoritma greedy tidak mengubah pilihan yang sudah dibuat. Sekali langkah diambil, algoritma tidak kembali untuk memperbaiki atau mengubah pilihan sebelumnya.

Efisiensi: Algoritma greedy biasanya lebih efisien dalam hal waktu dan sumber daya karena hanya mempertimbangkan langkah saat ini dan tidak perlu memproses seluruh ruang solusi.

D. *Penerapan Algoritma Greedy*

Penghitungan Koin Kembalian: Misalkan kita ingin memberikan kembalian dengan jumlah minimum koin. Algoritma greedy akan selalu memilih koin dengan denominasi terbesar yang kurang dari atau sama dengan jumlah yang tersisa, kemudian mengulang proses ini sampai semua kembalian diberikan.

Berikut adalah contoh penyelesaian *coin exchange*

```
function CoinExchange(C : himpunan_koin, A : integer) → himpunan_solusi
{ mengembalikan koin-koin yang total nilainya = A, tetapi jumlah koinnya minimum }
 Deklarasi
   S : himpunan_solusi
   x : koin
 Algoritma
 S ← {}
 while (∑(nilai semua koin di dalam S) ≠ A) and (C ≠ {} ) do
    x ← koin yang mempunyai nilai terbesar
    C ← C – {x}
    if (∑(nilai semua koin di dalam S) + nilai koin x ≤ A then
        S ← S ∪ {x}
    endif
 endwhile
 if (∑(nilai semua koin di dalam S) = A then
    return S
 else
   write('tidak ada solusi')
 endif
```

Pemilihan Aktivitas: Dalam masalah Pemilihan aktivitas, di mana kita ingin menjadwalkan sebanyak mungkin aktivitas yang tidak saling bertumpang tindih, algoritma greedy akan memilih aktivitas yang selesai paling cepat terlebih dahulu.

Berikut merupakan Solusi untuk pemilihan aktivitas

```
function Greedy-Activity-Selector(s1, s2, …, sn : integer, f1, f2, …, fn : integer) → set of integer
{ Asumsi: aktivitas sudah diurut terlebih dahulu berdasarkan waktu selesai:   f1 <= f2 <= ... <= fn }
Deklarasi
   i, j, n : integer
   A : set of integer
Algoritma:
  n ← length(s)
  A ← {1}     { aktivitas nomor 1 selalu terpilih }
  j ← 1
  for i ← 2 to n do
     if si ≥ fj then
        A ← A ∪ {i}
        j ← i
     endif
 endif
```

E. *Jenis Jenis Algoritma Greedy*

Contoh 7. Diberikan 4 buah objek sbb:

$(w_1, p_1) = (6, 12)$; $(w_2, p_2) = (5, 15)$; $(w_3, p_3) = (10, 50)$; $(w_4, p_4) = (5, 10)$;

dan sebuah *knapsack* dengan kapasitas $K = 16$.

Solusi dengan algoritma *greedy*:

Properti objek				*Greedy by*			Solusi
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	Optimal
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

- Solusi optimal: $X = (0, 1, 1, 0)$
- *Greedy by profit* dan *greedy by density* memberikan solusi optimal! Apakah untuk kasus lain juga optimal? Perhatikan Contoh 8 berikut.

1. Greedy by profit.
   - Pada setiap langkah, pilih objek yang mempunyai keuntungan terbesar.
   - Strategi ini mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan terlebih dahulu.
2. Greedy by weight.
   - Pada setiap langkah, pilih objek yang mempunyai berat teringan.
   - Mencoba memaksimumkan keuntungan dengan dengan memasukkan sebanyak mungkin objek ke dalam knapsack.
3. Greedy by density.
   - Pada setiap langkah, knapsack diisi dengan objek yang mempunyai pi/wi terbesar.
   - Mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar

F. Permainan Ular Tangga

Ular Tangga adalah permainan papan klasik yang dimainkan oleh dua atau lebih pemain. Permainan ini melibatkan papan yang terdiri dari petak-petak bernomor dari 1 hingga 100. Di antara petak-petak tersebut, terdapat beberapa tangga dan ular yang menghubungkan dua petak yang berbeda.

G. Tujuan Permainan:

Tujuan permainan ini adalah untuk menjadi pemain pertama yang mencapai petak terakhir (biasanya petak nomor 100).

H. Komponen Permainan:

Papan permainan: Papan dengan petak bernomor dari 1 hingga 100.
Bidak: Setiap pemain memiliki satu bidak yang mereka gerakkan di sepanjang papan.
Dadu: Sebuah dadu untuk menentukan jumlah langkah yang dapat diambil pemain pada giliran mereka.
Tangga: Menghubungkan satu petak ke petak lain yang lebih tinggi. Jika seorang pemain mendarat di petak dengan dasar tangga, mereka langsung naik ke petak di ujung atas tangga.
Ular: Menghubungkan satu petak ke petak lain yang lebih rendah. Jika seorang pemain mendarat di petak dengan kepala ular, mereka turun ke petak di ekor ular.

J. Cara Bermain:

Persiapan: Setiap pemain memilih satu bidak dan menempatkannya di petak awal (petak 1).
Giliran: Pemain melempar dadu secara bergiliran dan memindahkan bidak mereka sesuai dengan jumlah yang muncul di dadu.
Tangga: Jika pemain mendarat di dasar tangga, mereka segera naik ke petak yang terhubung dengan tangga tersebut.
Ular: Jika pemain mendarat di kepala ular, mereka segera turun ke petak yang terhubung dengan ekor ular.
Menang: Pemain yang pertama kali mencapai petak 100 adalah pemenangnya. Pemain harus mencapai petak 100 dengan tepat; jika mereka melempar angka yang lebih tinggi dari yang dibutuhkan untuk mencapai petak 100, mereka harus mundur.

K. Strategi:

Ular Tangga adalah permainan yang sebagian besar bergantung pada keberuntungan karena hasil lemparan dadu tidak dapat diprediksi. Namun, pemain bisa mengikuti beberapa strategi dasar:

1. Posisi Ular dan Tangga: Mengetahui di mana tangga dan ular berada dapat membantu pemain merencanakan langkah mereka, meskipun ini tidak selalu memungkinkan karena sifat acak dari dadu.

2. Meminimalkan Risiko: Jika memungkinkan, pemain dapat mencoba mendarat di petak yang aman (tidak ada kepala ular) untuk menghindari penurunan besar.

L. Varian Permainan:

1. Variasi Papan: Beberapa papan mungkin memiliki jumlah petak yang berbeda atau jumlah dan posisi tangga dan ular yang berbeda.
2. Aturan Tambahan: Beberapa varian memperkenalkan aturan khusus, seperti melempar dadu tambahan jika mendarat di petak tertentu, atau memiliki ular dan tangga yang hanya berlaku pada giliran tertentu.

III. Implementasi

A. Setup dan Inisialisasi

```python
import random

snakes_ladders = {
    3: 22, 5: 8, 11: 26, 20: 29,
    17: 4, 19: 7, 27: 1, 21: 9,
    16: 18, 25: 35, 32: 30, 34: 12,
    36: 6, 49: 11, 62: 19, 64: 60,
    87: 24, 93: 73, 95: 75, 99: 78,
}
```

B. Pendefinisian Greedy

```python
def roll_dice():
    return random.randint(1, 6)

def greedy_move(position):
    best_move = position
    for dice_roll in range(1, 7):
        new_position = position + dice_roll
        if new_position in snakes_ladders:
            new_position = snakes_ladders[new_position]
        if new_position > best_move:
            best_move = new_position
    return best_move
```

C. Main Program

```python
def play_game():
    position = 1
    moves = 0
    while position < 100:
        dice = roll_dice()
        print(f"Rolled a {dice}. Current position: {position}")
        new_position = position + dice
        if new_position in snakes_ladders:
            new_position = snakes_ladders[new_position]
        position = greedy_move(new_position)
        moves += 1
        print(f"Moved to position: {position}")
    print(f"Game won in {moves} moves!")

if __name__ == "__main__":
    play_game()
```

Fungsi ini menjalankan permainan:

1. Inisialisasi: Posisi awal di kotak 1 dan jumlah langkah (moves) diatur ke 0.
2. Permainan Berlangsung: Selama posisi pemain kurang dari 100:
   Pemain melempar dadu dan hasilnya disimpan di dice.
   Posisi baru dihitung dengan menambahkan hasil lemparan dadu ke posisi saat ini.
   Jika posisi baru adalah kotak dengan tangga atau ular, perbarui posisi baru berdasarkan peta snakes_ladders.
   Pilih langkah terbaik menggunakan fungsi greedy_move.
   Tingkatkan jumlah langkah.
   Cetak posisi baru pemain.
3. Permainan Selesai: Setelah mencapai atau melewati kotak 100, cetak jumlah langkah yang dibutuhkan untuk memenangkan permainan.

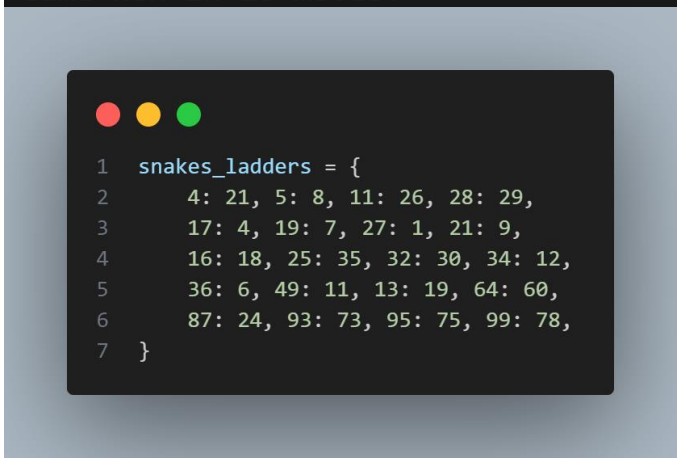
D. Contoh Output 1

```
Rolled a 6. Current position: 1
Moved to position: 26
Rolled a 3. Current position: 26
Moved to position: 35
Rolled a 5. Current position: 35
Moved to position: 46
Rolled a 2. Current position: 46
Moved to position: 54
Rolled a 3. Current position: 54
Moved to position: 63
Rolled a 6. Current position: 63
Moved to position: 75
Rolled a 2. Current position: 75
Moved to position: 83
Rolled a 6. Current position: 83
Moved to position: 94
Rolled a 1. Current position: 94
Moved to position: 81
Rolled a 4. Current position: 81
Moved to position: 91
Rolled a 4. Current position: 91
Moved to position: 81
Rolled a 3. Current position: 81
Moved to position: 90
Rolled a 6. Current position: 90
Moved to position: 102
Game won in 13 moves!
```

```python
snakes_ladders = {
    3: 22, 5: 8, 11: 26, 20: 29,
    17: 4, 19: 7, 27: 1, 21: 9,
    16: 18, 25: 35, 32: 30, 34: 12,
    36: 6, 49: 11, 62: 19, 64: 60,
    87: 24, 93: 73, 95: 75, 99: 78,
}
```

E. Contoh Output 2

```
Rolled a 4. Current position: 1
Moved to position: 26
Rolled a 3. Current position: 26
Moved to position: 35
Rolled a 3. Current position: 35
Moved to position: 44
Rolled a 4. Current position: 44
Moved to position: 54
Rolled a 4. Current position: 54
Moved to position: 63
Rolled a 2. Current position: 63
Moved to position: 71
Rolled a 5. Current position: 71
Moved to position: 82
Rolled a 2. Current position: 82
Moved to position: 90
Rolled a 2. Current position: 90
Moved to position: 98
Rolled a 4. Current position: 98
Moved to position: 108
Game won in 10 moves!
```



```
snakes_ladders = {
    4: 21, 5: 8, 11: 26, 28: 29,
    17: 4, 19: 7, 27: 1, 21: 9,
    16: 18, 25: 35, 32: 30, 34: 12,
    36: 6, 49: 11, 13: 19, 64: 60,
    87: 24, 93: 73, 95: 75, 99: 78,
}
```

IV. KESIMPULAN

Makalah ini mengeksplorasi penerapan algoritma greedy dalam permainan ular tangga, sebuah permainan papan yang sederhana namun sangat bergantung pada keberuntungan. Dengan memanfaatkan algoritma greedy, kita dapat meningkatkan strategi permainan untuk memaksimalkan peluang kemenangan.

Prinsip Algoritma Greedy: Algoritma greedy bekerja dengan memilih langkah yang paling menguntungkan pada setiap keputusan tanpa mempertimbangkan konsekuensi jangka panjang. Dalam konteks permainan ular tangga, ini berarti memilih langkah yang membawa pemain ke posisi tertinggi atau paling menguntungkan berdasarkan hasil lemparan dadu.

Implementasi pada Permainan Ular Tangga: Dalam permainan ular tangga, algoritma greedy digunakan untuk menentukan langkah terbaik setelah setiap lemparan dadu. Pemain mengevaluasi semua kemungkinan langkah dari hasil lemparan dadu dan memilih langkah yang memberikan keuntungan langsung terbesar, seperti naik tangga atau menghindari ular.

Keuntungan: Algoritma greedy mudah diimplementasikan dan cepat dalam mengambil keputusan karena hanya mempertimbangkan langkah saat ini. Ini membantu pemain untuk segera mengambil langkah terbaik berdasarkan posisi saat ini.

Keterbatasan: Algoritma greedy tidak selalu menjamin solusi optimal secara keseluruhan karena hanya fokus pada keuntungan jangka pendek. Ada situasi di mana pilihan lokal optimal tidak mengarah pada solusi global optimal.

Implementasi Kode Python: Implementasi kode dalam bahasa Python menunjukkan bagaimana algoritma greedy dapat digunakan untuk memilih langkah optimal berdasarkan kondisi saat ini dan hasil lemparan dadu. Kode ini mengilustrasikan langkah-langkah dalam permainan dan menunjukkan bagaimana strategi greedy dapat membantu pemain mencapai kotak 100 lebih efisien.

Kesimpulan Akhir

Penerapan algoritma greedy dalam permainan ular tangga menunjukkan bahwa meskipun permainan ini sangat dipengaruhi oleh keberuntungan, strategi yang tepat dapat memberikan keuntungan signifikan. Algoritma greedy membantu pemain untuk memaksimalkan peluang menang dengan mengambil langkah yang paling menguntungkan pada setiap giliran. Meskipun tidak selalu menjamin solusi terbaik secara keseluruhan, pendekatan ini tetap memberikan panduan yang berguna dalam permainan yang mengandalkan keberuntungan.

VIDEO LINK AT YOUTUBE

https://youtu.be/pBWwa5URN_8

ACKNOWLEDGMENT

Penulis ingin mengucapkan rasa syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat yang telah diberikan, sehingga dapat menyelesaikan karya tulis yang berjudul "Optimasi Portofolio Investasi dengan Algoritma Branch & Bound". Penulis juga mengucapkan terima kasih kepada keluarga atas dukungan dan semangat yang diberikan selama proses penulisan karya tulis ini. Terima kasih juga disampaikan kepada Ir. Rila Mandala, M.Eng., Ph.D., dan Monterico Adrian, S.T., M.T., dosen pengampu Mata Kuliah Strategi Algoritma ITB untuk Kelas 3, atas bimbingan dan ilmu yang telah diberikan.

DAFTAR PUSTAKA

[1] R. Munir, “Algoritma Greedy 2021 (Bagian 1)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

[2] R. Munir, “Algoritma Greedy 2021 (Bagian 2)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

[3] R. Munir, “Algoritma Greedy 2021 (Bagian 3)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf

Tertanda

Jimly Nur Arif 13522123

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.

andung, 12 Juni 2024