

ANALISIS PERBANDINGAN ALGORITMA *BRUTE FORCE* DAN *GREEDY* UNTUK SOLUSI PEMILIHAN *FIGHTERS* PADA GAME RPG

Bagas Eko Tjahyono Putro, Faiz Maula Ahmad Edwin Putra, Muhammad Adnansyah

1301223279, 1301223017, 1301223460

sanfla@student.telkomuniversity.ac.id, faizmaula@student.telkomuniversity.ac.id,

adnansyah@student.telkomuniversity.ac.id

ABSTRAK

Knapsack problem merupakan masalah dimana orang dihadapkan pada persoalan optimasi pemilihan benda yang dapat ditampung ke dalam sebuah knapsack (karung) yang memiliki keterbatasan daya dan ruang tampung. Dalam penelitian ini, kami menggunakan *Knapsack Problem* pada aplikasi *Fighters Selector* dengan algoritma *Brute Force* dan algoritma *Greedy*. Tujuan dari dilakukan penelitian ini adalah untuk membandingkan algoritma yang memiliki solusi paling optimal antara algoritma *Brute Force* dan algoritma *Greedy*. Hasil yang didapatkan yaitu algoritma *Brute Force* memiliki solusi yang lebih optimal dibandingkan algoritma *Greedy*.

Kata kunci: Algoritma, *Fighters*, *Selector*, *Brute*, *Force*, *Greedy*

1. PENDAHULUAN

Pada game bergenre RPG atau Role-Playing Game, pemain biasa dihadapkan dengan suatu level yang memiliki *requirements* tertentu. Biasanya setiap level memiliki boss yang harus dikalahkan agar pemain dapat menyelesaikan level tersebut. Salah satu persyaratan untuk mengalahkan boss adalah susunan tim yang dimiliki harus mempunyai

total power lebih besar dari boss di level tersebut. Tetapi, setiap tim yang disusun harus sesuai dengan kapasitas *budget* yang dimiliki oleh pemain tersebut. *Fighters Selector* adalah program untuk menentukan *fighter* terbaik sesuai *budget* yang dimiliki.

1.1 METODE PENYELESAIAN

Pemilihan hero bisa disesuaikan dengan power, harga, atau *density*. *Fighter* yang dipilih merupakan pilihan *fighter* terbaik yang direkomendasikan oleh sistem berdasarkan dengan implementasi algoritma *Brute Force* ataupun algoritma *Greedy*.

Pada pengembangan aplikasi ini algoritma yang akan digunakan untuk penyelesaian masalah adalah algoritma *Brute Force* dan algoritma *Greedy*. Algoritma *Brute Force* adalah strategi pencarian yang sederhana dan komprehensif yang secara sistematis mengeksplorasi setiap opsi hingga jawaban dari suatu masalah ditemukan. Sedangkan algoritma *Greedy* adalah kelas algoritma yang membuat pilihan optimal secara lokal pada setiap langkah dengan harapan menemukan solusi optimal global.

Pemilihan dua algoritma di atas bertujuan untuk membandingkan hasil dari masing-masing algoritma. Penyusunan tim untuk tiap level dapat ditentukan dengan menerapkan prinsip 0/1 Knapsack Problem.

Pendekatan *Brute Force* dipilih untuk membandingkan hasil dari *Greedy*, untuk mengetahui apakah solusi yang didapatkan dari metode *Greedy* optimal atau tidak.

2. ANALISIS ALGORITMA

Pada penelitian ini kami melakukan analisis terhadap dua algoritma untuk menentukan *fighter* yang akan dipilih, yaitu algoritma *Brute Force* dan algoritma *Greedy*.

2.1 Algoritma *Brute Force*

Algoritma *Brute Force* adalah sebuah pendekatan yang langsung (straightforward) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (problem statement) dan definisi konsep yang melibatkan. Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (obvious way). Cara kerja algoritma *Brute Force* pada *knapsack problem* adalah sebagai berikut:

- 1.) Enumerasi semua kombinasi
- 2.) Evaluasi setiap kombinasi
- 3.) Memilih kombinasi terbaik

2.1.1 Implementasi Algoritma *Brute Force*

Algoritma berikut diimplementasi dengan bahasa pemrograman *Python*

```
import itertools

def bruteForce(fighters, hasil, hfighter, koin, tPower):
    n = len(fighters)
    best_power = 0
    best_combination = []

    for r in range(1, n + 1):
        for combination in itertools.combinations(fighters, r):
            total_harga = sum(fighter.harga for fighter in combination)
            total_power = sum(fighter.power for fighter in combination)
            if total_harga <= koin and total_power > best_power:
                best_power = total_power
                best_combination = combination

    for fighter in best_combination:
        hasil.append(fighter.img_url)
        hfighter.append(fighter.nama)
        koin -= fighter.harga

    tPower.append(best_power)
    return koin
```

Gambar 1. Algoritma *Brute Force*

2.1.2 Kompleksitas Algoritma *Brute Force*

- 1.) Untuk n buah objek, banyaknya himpunan bagian yang harus dievaluasi adalah 2^n
- 2.) Perhitungan total bobot setiap himpunan bagian dapat dikerjakan dalam $O(n)$
- 3.) Sehingga kompleksitas algoritma exhaustive search untuk permasalahan 0/1 knapsack adalah $O(n \cdot 2^n)$

2.2 Algoritma *Greedy*

Algoritma *Greedy* adalah kelas algoritma yang membuat pilihan optimal secara lokal pada setiap langkah dengan harapan menemukan solusi optimal global. Pendekatan yang digunakan dalam algoritma *Greedy* adalah membuat pilihan yang memberikan peroleh terbaik. Algoritma *Greedy* disusun oleh beberapa komponen, yaitu:

- Himpunan kandidat: himpunan objek yang merepresentasikan objek yang belum dipilih

- Fungsi solusi: tidak ada kandidat yang dapat dipilih karena beban yang melebihi sisa kapasitas *knapsack*
- Fungsi seleksi: memilih objek dari himpunan kandidat yang tersisa
 - by profit: memiliki keuntungan terbesar
 - by weight: memiliki berat terkecil
 - by density: memiliki densitas terbesar
- Fungsi kelayakan: total berat yang dipilih tidak lebih besar dari kapasitas *knapsack*
- Fungsi obyektif: memaksimumkan total keuntungan (profit) dari objek-objek yang dipilih.

Cara kerja algoritma *Greedy* pada *knapsack problem* ada sebagai berikut:

- 1) *By profit*
 - Mengurutkan obyek - obyek secara *descending*
 - Mengambil satu persatu obyek sampai *knapsack penuh*
- 2) *By weight*
 - Mengurutkan obyek - obyek secara *ascending*
 - Mengambil satu persatu obyek sampai *knapsack penuh*
- 3) *By density*
 - Mencari nilai *density* dari tiap obyek
 - Mengurutkan obyek - obyek berdasarkan densitasnya secara *descending*

- Mengambil satu persatu obyek sampai *knapsack* penuh

2.2.1 Implementasi Algoritma *Greedy*

Algoritma berikut diimplementasi dengan bahasa pemrograman *Python*

```
def greedyABC(fighters, hasil, hfighter, koin, tPower):
    best_power = 0
    for fighter in fighters:
        if fighter.harga <= koin:
            hasil.append(fighter.img_url)
            hfighter.append(fighter.nama)
            koin -= fighter.harga
            best_power += fighter.power

    tPower.append(best_power)

    return koin
```

Gambar 2. Algoritma *Greedy*

2.2.2 Kompleksitas Algoritma *Greedy*

- 1) Kita memilih barang keuntungan terbesar yang masih bisa dipilih (berdasarkan kapasitas *knapsack*) di antara semua barang yang belum dipilih
- 2) Untuk mengoptimalkan proses pemilihan objek berdasarkan keuntungan, maka objek perlu diurutkan berdasarkan besar keuntungan secara menurun
- 3) Jika waktu pengurutan tidak dihitung, maka kompleksitas algoritma *greedy by profit* untuk masalah 0/1 *knapsack* adalah $O(n)$

3. DATA DAN EKSPERIMEN

3.1 Deskripsi Data

Data dari tiap karakter dibuat berdasarkan game RPG yang ada. Masing-masing karakter pada kedua mempunyai atribut yang meliputi:

- Nama
- Harga
- Power

Dibutuhkan juga data Boss dan Koin Awal untuk tiap levelnya.

3.2 Jumlah Data

Terdapat sejumlah 5 data karakter berbeda untuk tiap level dan masing-masing karakter memiliki power dan harga yang berbeda.

3.3 Sampel Data

Pengujian algoritma Brute Force dan algoritma Greedy, dilakukan dengan data yang telah dimasukan kedalam file excel bernama data.xlsx, data yang telah disiapkan terdiri dari beberapa tingkatan level dan setiap level memiliki data yang berbeda, berikut datanya:

Tabel Data Level 1

Boss	750	
Koin Awal	305	
Data Fighter		
Nama	Koin	Power
Claart	95	235
Bumba	105	280
Peter	135	375
James	90	275
Wolf	75	195

Tabel Data Level 2

Boss	1200	
Koin Awal	625	
Data Fighter		
Nama	Koin	Power
Claart	125	255
Bumba	135	280
Peter	200	325
James	175	420
Wolf	165	375

Tabel Data Level 3

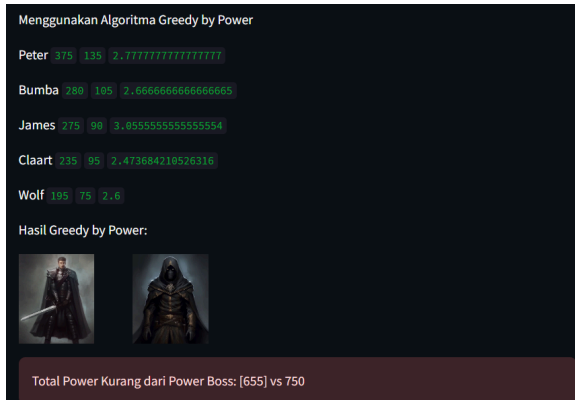
Boss	1375	
Koin Awal	795	
Data Fighter		
Nama	Koin	Power
Claart	198	375
Bumba	175	345
Peter	180	365
James	150	300
Wolf	190	350

Data yang telah tersedia kemudian akan diinputkan kedalam aplikasi, dan program akan menentukan urutan fighter berdasarkan koin awal yang dimiliki user dan power yang cukup untuk mengalahkan Boss.

3.2 Hasil

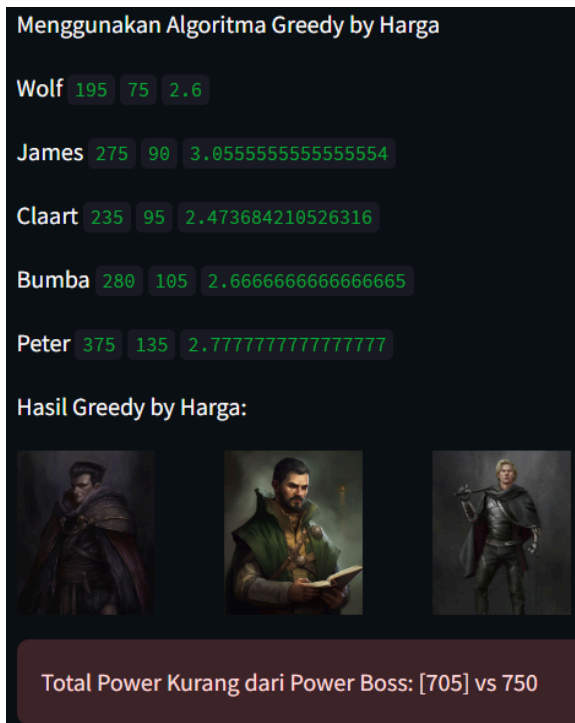
Uji Data Tabel Level 1

a. Greedy by Power

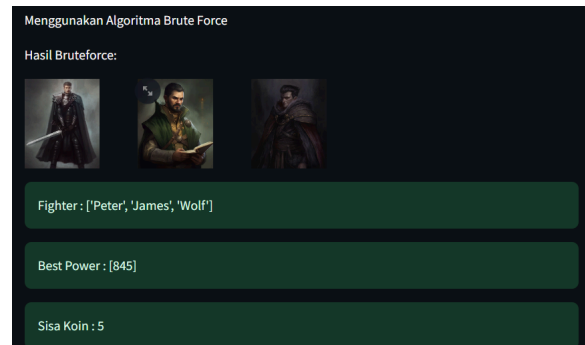


Dengan metode *Greedy by Power*, solusi yang didapatkan tidak memenuhi syarat, karena power kurang dari power boss. Sehingga metode *Greedy by Power* tidak bisa digunakan untuk data ini.

b. Greedy by Weight

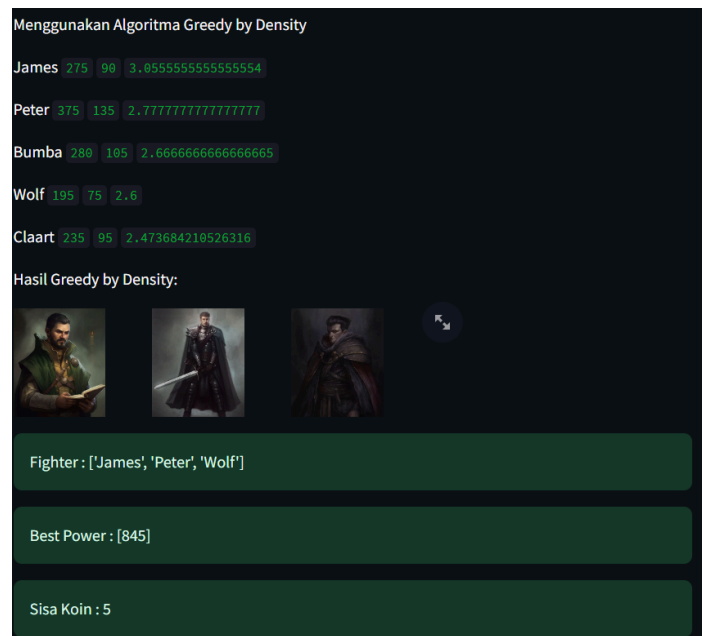


Hal yang sama didapatkan ketika solusi diselesaikan menggunakan metode *Greedy by Weight*.



Solusi optimal untuk data ini bisa didapatkan dengan metode *Brute Force*, susunan timnya adalah {Peter, James, Wolf} dengan power : $845 > 750$.

c. Greedy by Density



Solusi yang sama bisa didapatkan menggunakan metode *Greedy by Density*.

Perbandingan running time:

a. Brute Force

Running time: 0.003175973892211914 seconds

b. Greedy by Density

Running time: 0.004873752593994141 seconds

Running time *Brute Force* lebih kecil dibandingkan dengan *Greedy*, karena *Greedy* memerlukan data untuk disorting terlebih dahulu.

Uji Data Tabel Level 2

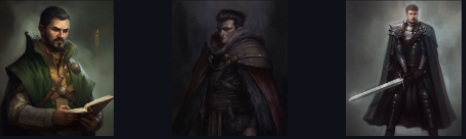
a. *Greedy By Power*

Urutkan data berdasarkan power secara *descending*.

Menggunakan Algoritma Greedy by Power

James	420	175	2.4
Wolf	375	165	2.272727272727273
Peter	325	200	1.625
Bumba	280	135	2.074074074074074
Claart	255	125	2.04

Hasil Greedy by Power:



Total Power Kurang dari Power Boss: [1120] vs 1200

Solusi yang didapatkan menggunakan metode *Greedy by Power* tidak memenuhi persyaratan.

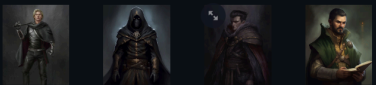
b. *Greedy by Weight*

Urutkan data berdasarkan berat/harga secara *ascending*.

Menggunakan Algoritma Greedy by Harga

Claart	255	125	2.04
Bumba	280	135	2.074074074074074
Wolf	375	165	2.272727272727273
James	420	175	2.4
Peter	325	200	1.625

Hasil Greedy by Harga:



Fighter: ['Claart', 'Bumba', 'Wolf', 'James']

Best Power: [1330]

Sisa Koin: 25

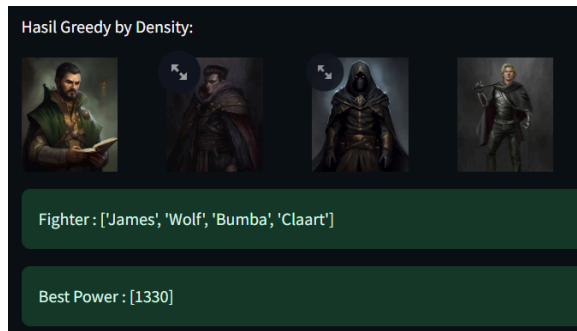
Solusi yang didapatkan memenuhi syarat. Susunan tim yang diberikan yaitu, {Claart, Bumba, Wolf, James}. Dengan power : 1330 > 1200.

c. *Greedy by Density*

Urutkan data secara *descending* berdasarkan *density*.

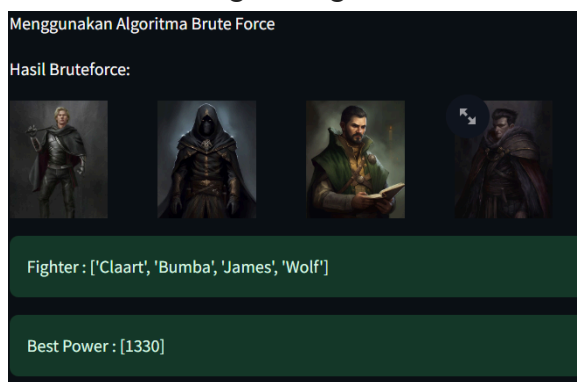
Menggunakan Algoritma Greedy by Density

James	420	175	2.4
Wolf	375	165	2.272727272727273
Bumba	280	135	2.074074074074074
Claart	255	125	2.04
Peter	325	200	1.625



Metode *Greedy by Density* memberikan solusi yang sama seperti *Greedy by Harga*.

d. Perbandingan dengan *Brute Force*



Solusi yang didapatkan dengan metode *Greedy by Density* dan *Greedy by Weight* = *Brute Force*. Sehingga untuk data ini dapat digunakan metode *Greedy* untuk mendapatkan solusi optimal.

a. Brute Force

Running time: 0.011190652847290039 secon

b. Greedy by Density

Running time: 0.005075931549072266 seconds

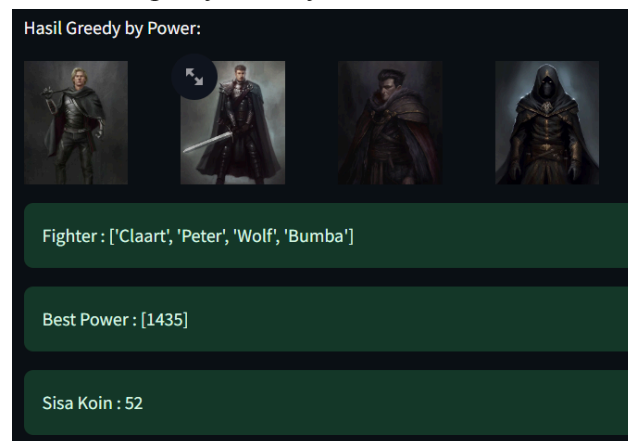
c. Greedy by Weight

Running time: 0.005167245864868164 seconds

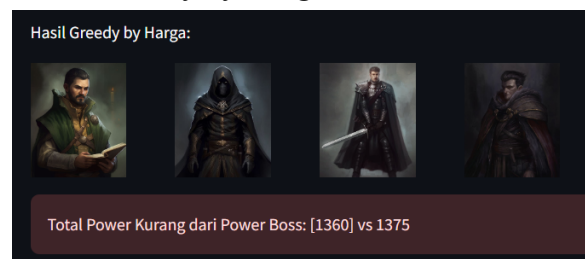
Uji Data Tabel Level 3

a. Greedy By Power

Menggunakan algoritma Greedy by Power didapatkan susunan tim yang memenuhi persyaratan yaitu,

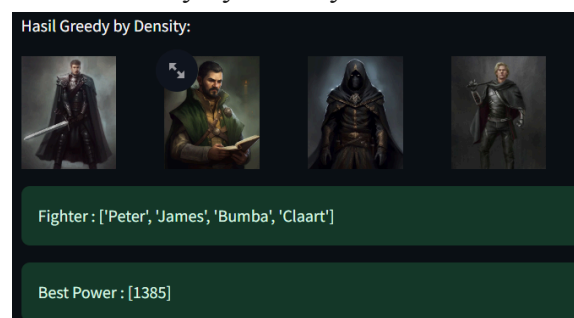


b. Greedy by Weight



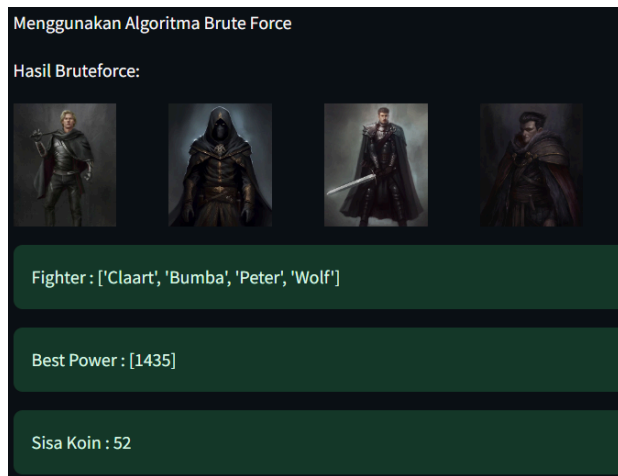
Dengan *Greedy by Weight*, solusi yang didapatkan tidak memenuhi persyaratan karena total power < power boss.

c. Greedy by Density



Dengan metode *Greedy by Density*, solusi yang didapatkan memiliki power > boss, tetapi bukan solusi terbaik.

d. Perbandingan dengan Brute Force
Setelah dibandingkan dengan Metode *Brute Force*. Susunan tim yang didapatkan adalah sama, sehingga untuk data ini metode *Greedy by Power* dapat digunakan untuk mendapatkan solusi yang optimal.



4. CONCLUSION

Berdasarkan eksperimen, algoritma Brute Force cenderung memiliki running time lebih cepat dibandingkan algoritma Greedy karena Greedy memerlukan sorting sebelumnya. Meskipun begitu, Brute Force dapat dijadikan pembandingan untuk solusi yang dihasilkan oleh Greedy karena selalu menghasilkan solusi optimal.

Namun, dalam praktiknya, terutama pada skala masalah besar, Brute Force tidak praktis karena kompleksitas waktu yang eksponensial. Algoritma ini mencoba setiap kemungkinan kombinasi item untuk mencari solusi optimal, sehingga waktu komputasinya sangat tinggi saat jumlah item meningkat.

Sebaliknya, algoritma Greedy, meskipun memerlukan langkah sorting yang

menambah overhead waktu, memiliki kompleksitas waktu yang lebih terukur dan dapat diandalkan dalam banyak kasus praktis. Meskipun tidak menjamin solusi optimal, Greedy sering memberikan solusi yang baik dalam waktu singkat, terutama pada masalah dengan skala besar.

REFERENSI

G.S. Wulandari, S. Saadah. Pengantar Strategi Algoritma, KBM Indonesia, 2021.

Riri Nada Devita and Aji Prasetya Wibawa, "Teknik-teknik optimasi knapsack problem," Sains, Aplikasi, Komputasi dan Teknologi Informasi., Vol 2, No 1, April 2020, pp. 35-40.

CII2K3 STRATEGI ALGORITMA - Topik 1 BRUTE FORCE

CII2K3_SA_EAR-Greedy-2022-2