**LAPORAN TUGAS BESAR**

**STRATEGI ALGORITMA**



**JUDUL :**

**EFISIENSI ALGORITMA BRUTE FORCE, GREEDY, DAN DYNAMIC PROGRAMMING UNTUK KNAPSACK PROBLEM**

Disusun oleh:

Ayudia Maharani (21102050)

Sani Akhzam Prakistiyanto (21102057)

Jovi Inzagi (21102064)

Puspita Kartika Sari (21102075)

Nazar Yosa Wisnu Wardana (21102192)

**TEKNIK INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO**

**BANYUMAS, JAWA TENGAH**

**2024**

1. **Dasar Teori**

Sebuah perusahaan logistik memiliki sebuah truk dengan kapasitas maksimum 100 kg. Ada beberapa paket yang harus dikirim ke pelanggan, tetapi truk hanya dapat mengangkut paket hingga mencapai kapasitas maksimum.

Setiap paket memiliki:

1. Berat: Berat paket dalam kilogram.
2. Keuntungan: Nilai profit dari pengiriman paket tersebut.

Perusahaan ingin memaksimalkan keuntungan dengan memilih kombinasi paket yang bisa diangkut, tanpa melebihi kapasitas truk.

Permasalahan tersebut akan diselesaikan dengan beberapa algoritma dibawah ini:

* 1. **Algoritma Brute Force**

Algoritma brute force adalah metode sederhana untuk menyelesaikan masalah dengan mencoba semua kemungkinan solusi satu per satu hingga menemukan yang benar.

Proses ini dilakukan tanpa mempertimbangkan cara yang lebih efisien atau pola tertentu. Misalnya, dalam konteks pemecahan kata sandi atau enkripsi, algoritma ini akan mencoba setiap kombinasi karakter hingga kombinasi yang sesuai ditemukan.

Brute force memiliki tujuan utama untuk mengeksplorasi semua kemungkinan dalam ruang solusi secara menyeluruh, sehingga solusi terbaik atau jawaban pasti akan ditemukan, meskipun membutuhkan waktu yang lama dan memakan sumber daya komputasi yang besar.

* 1. **Algoritma Greedy**

Algoritma greedy adalah algoritma yang memiliki prinsip “take what you can get now!”. Algoritma greedy membentuk solusi langkah per Langkah, Pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan, Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah berikutnya.

Pendekatan yang digunakan di dalam algoritma greedy adalah membuat pilihan yang “tampaknya” memberikan perolehan terbaik.

Dalam kasus ini permasalahannya adalah knapsack 0/1 yaitu Diberikan n buah objek dan sebuah knapsack dengan kapasitas bobot W. Setiap objek memiliki properti bobot (weigth) wi dan keuntungan(profit) pi. persoalan adalah memilih objek-objek yang dimasukkan ke dalam knapsack sedemikian sehingga memaksimumkan keuntungan. Total bobot objek yang dimasukkan ke dalam knapsack tidak boleh melebihi kapasitas knapsack.

* + 1. **Greedy by Weight**

Pada setiap langkah, knapsack diisi dengan objek yang mempunyai berat paling ringan.  Strategi ini mencoba memaksimumkan keuntungan dengan memasukan sebanyak mungkin objek kedalam knapsack.

Pertama kali yang dilakukan adalah mengurutkan secara menaik objek-objek berdasarkan weight-nya. Kemudian obyek-obyek yang dapat ditampung oleh knapsack diambil satu persatu sampai knapsack penuh atau (sudah tidak ada obyek lagi yang bisa dimasukan).

* + 1. **Greedy by Profit**

Pada setiap langkah Knapsack diisi dengan obyek yang mempunyai keuntungan terbesar. Strategi ini mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan  terlebih dahulu.

Pertama kali dilakukan adalah menurutkan secara menurun obyek-obyek berdasarkan profitnya .  Kemudian obyek-obyek yang dapat ditampung oleh knapsack diambil satu persatu sampai knapsack penuh atau (sudah tidak ada obyek lagi yang bisa dimasukan).

* + 1. **Greedy by Density**

Pada setiap langkah, knapsack diisi dengan obyek yang mempunyai densitas terbesar (perbandingan nilai dan berat terbesar). Strategi ini mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar.

Pertama kali yang dilakukan adalah mencari nilai profit per unit/ density dari tiap-tiap objek. Kemudian obyek-obyek diurutkan berdasarkan densitasnya. Kemudian obyek-obyek yang dapat ditampung oleh knapsack diambil satu persatu sampai knapsack penuh atau (sudah tidak ada obyek lagi yang bisa dimasukan). Keuntungan per unit berat (densitas) dari objek .

* 1. **Algoritma Dynamic Programming**

Dynamic programming adalah suatu teknik pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan tahapan (stage). Dynamic programming membagi masalah menjadi beberapa sub-masalah, kemudian mencari solusi sub-masalah yang kecil terlebih dahulu, dan simpan hasilnya. Dynamic programming dapat digunakan untuk mencari solusi dari permasalahan optimasi.

Pendekatan dynamic programming terbagi menjadi 2 macam yaitu **pendekatan maju (forward/bottom-up)** dan **pendekatan mundur (backward/top-down).** Dimana pendekatam maju(Bottom up) adalah Perhitungan dilakukan dari tahap 1, dilanjutkan ke tahap 2, kemudian tahap 3, dan seterusnya hingga tahap n. Sedangkan pendekatan mundur(top-down) adalah Perhitungan dilakukan dari tahap n, dilanjutkan ke tahap n − 1, kemudian tahap n − 2, dan seterusnya hingga tahap 1.

Fungsi rekursif untuk knapsack 0/1 dengan menggunakan dynamic programming adalah sebagai berikut:

* Jika dan tidak sama dengan 0

1. **Spesifikasi *Hardware***

* RAM : 8 GB
* CPU : AMD Athlon 3000u
* OS : Windows 11
* Bahasa Pemrograman : Python
* IDE : Google Collabs

1. **Implementasi**

Dalam study kasus yang diambil dapat disimpulkan bahwa persoalan yang dihadapi adalah terkait dengan knapsack 0/1, Dimana ada sebuah truk yang memiliki kapasitas maksimum 100 kg dan ingin memuat paket yang masing-masing memiliki berat dan profit sehingga dibutuhkan cara agar mendapatkan kombinasi barang yang optimal yang cukup untuk dibawa dengan truk. Knapsack adalah rogramsebuah permasalahan untuk menentukan pilihan objek dari sekumpulan objek yang masing-masing mempunyai bobot(weight) dan profit(value) untuk dimuat dalam sebuah media penyimpanan tanpa melebihi kapasitas media penyimpanan tersebut sehingga mendapatkan hasil yang optimum.

Dalam study kasus ini dipilih 3 metode yang menungkinkan untuk menyelesaikan permaslaahan tersebut yaitu bruteforce, greedy dan dynamic programming. Greedy akan dibagi menjadi 3 yaitu by weight, by profit dan by density. Berikut adalah penjelasan dari program yang dibuat untuk mengatasi permaslaahan tersebut dengan menggunakan metode-metode diatas.

1. Pembuatan data

Pada bagian ini bertanggungjawab untuk membuat data acak berupa berat(weights) dan keuntungan (profit). Dimana data itu akan disimpan dan digunakan untuk diproses menggunakan algoritma-algoritma yang sudah ditentukan. Hal yang dilakukan pertama kali adalah menentukan ukuran datanya yaitu ada 10, 100, 1000, 10000, 100000. Kemudian menentukan value maksimum untuk weight dan profit yaitu 100 untuk weight dan 250 untuk profit. Kemudian melakaukan generate data berdasarkan parameter paramenter yang sudah dibuat tadi kemudian hasilnya disimpan dalam dictionary data. Pembuatan data dan penyimpanan data ini bertujuan agar data yang digunakan pada setiap metode sama.

1. Metode Brute Force

Algoritma itu berjalan dengan yang dilakukan pertama kali adalah melakukan inisialisasi variable untuk menampung nilai nilai yang diperlukan yaitu jumlah brang, nilai maksimum dan kombinasinya.

Kemudian melakukan looping ukuran subsetnyadari 0 hinggan n barang, lalu menggunakan fungsi kombinasi untunk menghasilkan semua subset dengan ukuran r..

Untuk setiap subset akakn dilakuka looping untunk menghitung berat(total\_weight) dengan menjumlahkan berat baranag di subset. Kemudian meghitung total nilai(total\_value) edngan menjumlahkan nilai barang di subset

Jika total beratnya kurang dari kapasitas dan total\_valuenya lebih besar dari max\_value maka max\_value akan di perbarui dengan total\_value dan menyimpan kombinasi terbaiknya di subset sebagai best\_combination.

Setelah semua subset di perika nilai best\_combination akan menampilkan kombinasi terbaiknya dan max\_valuenya menunjukan nilai terbaiknya.

1. Metode Greedy

Metode greedy akan menerima data berat, profit, dan kapasitas yang sudah disimpan sebelumnya. Kemudian program akan menentukan modenya, yaitu antara by weight dengan prioritas barang berat ternedah, by profit dengan barang profit tertinggi, dan by density prioritas barag dengan profit to weight ratioya yang tertinggi, Dimana profit to weight ratio tertinggi didapatkan dari pembagian antara weight dan profit. Kemudian selanjutnya barang akan diurutkan dari skor tertinggi ke skor terendah untuk memastikan prioritas tertinggi diproses terlebih dahulu.

Kemudian iterasi dimulai untuk barang yang sudah diurutkan tersebut. Untuk setiap barang jika tidak melebihi kapasiitas maka akan di tambahkan ke knapsack jika lebih tidak dimasukan.

1. Metode Dynamic Programming

Membuat metode dynamic programming. Dynamic programming berjalan pertama dengan mendefinisikan table 2D yaitu untuk indeks dari 1 hingga n dan w kapasitasnya dari 0 hngga kapasitas maksimalnya. Isi nilainya adalah profit maksimum yanag diperoleh dari barang 1 hingga kapasitas maksimumnya.

Selanjutnya melakukan inisialisasi, jika tidak ada barang i=0 atau kapasitas tas nol w = 0. Kemudian dilanjutkan dengan pengisian table yaitu untuk setiap barang dan setiap kapasitas tas, jika barang lebih besar dari kapasitas saat ini maka barang tidak dapat dimasukan. Namun jika barang lebih kecil daripada kapasitas saat ini dengan mengambil nilai maksimum antara dua opsi yaitu Tidak memasukkan barang: dp[i−1][w]. atau Memasukkan barang: profits[i−1]+dp[i−1][w−weights[i−1]]

1. Menjalankan Semua Algoritma

Melakukan iterasi untuk setiap ukuran data set yaitu sebanyak 10, 100, 1000, 10000, 100000 data dengan data datanya yang sudah dibuat sebelumnya.kemudian melakukan esekusi dan menghitung waktunya pada metode yang sudah dibuat, pada metode bruteforce kami membatasi untuk datasetnya yaitu jika diatas 20 tidak akan di eksekusi karena akan memakan waktu yang sangat lama.

Kemudian pada greedy dilakukan juga penghitungan waktunya berdasarkan weight, profit, dan density. Kemudian untuk metode dynamic programming juga sama djalanak dan diambil waktunya. Lalu hasilnya disimpan.

1. Visualisasi data
2. **Sumbu X (Ukuran Dataset)**: Menampilkan ukuran dataset pada skala logaritmik.
3. **Sumbu Y (Waktu Eksekusi)**: Menampilkan waktu eksekusi dalam detik pada skala logaritmik.
4. **Garis dan Titik**: Mewakili waktu eksekusi setiap algoritma untuk ukuran dataset tertentu.
5. **Interpretasi**:
   1. Brute Force biasanya meningkat eksponensial dan mungkin terputus untuk ukuran dataset besar.
   2. Metode Greedy menunjukkan waktu eksekusi yang cepat.
   3. Dynamic Programming umumnya berada di antara brute force dan greedy.

A screenshot of a computer

Description automatically generated

A graph with a line graph

Description automatically generated with medium confidence

1. **Pengujian**
   1. **Data yang digunakan**

Data yang digunakan untuk melakukan pengujian ini adalah data acak untuk weight dan profit yang dibuat dengan fungsi random. Kemudian data random weight dan ptofitnya disimpan dalam sebuah tuple yang nantinya akan digunakan di setiap metode. Data yang digunakan bervariasi mulai dari 10, 100, 1000, 10000, dan 100000 untuk setiap metodenya.

* 1. **Tabel waktu pengujian**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **n** | **Brute Force** | **Greedy by Weight** | **Greedy by Profit** | **Greedy by Density** | **Dynamic Programming** |
| **10** | 0.001529 | 0.000029 | 0.000014 | 0.000018 | 0.000380 |
| **100** | - | 0.000072 | 0.000057 | 0.000078 | 0.003447 |
| **1000** | - | 0.000656 | 0.000435 | 0.000512 | 0.042763 |
| **10000** | - | 0.010377 | 0.006252 | 0.008422 | 0.627069 |
| **100000** | - | 0.101644 | 0.104410 | 0.133403 | 4.176093 |

1. **Analisis Hasil Pengujian**
2. Brute Force:
   * Waktu eksekusi tertinggi (0.001529 detik) meskipun dataset kecil karena algoritma mengeksplorasi semua kombinasi kemungkinan untuk memastikan solusi optimal.
   * Eksekusi untuk dataset melebihi 20 tidak dihitung karena memakan waktu yang sangat lama.
   * Kompleksitas eksponensial membuat algoritma ini kurang efisien bahkan untuk dataset kecil.
3. Greedy (Weight, Profit, Density):
   * Metode Greedy by Profit adalah yang tercepat (0.000014 detik) dalam menyelesaikan masalah ini.
   * Greedy by Density memberikan solusi optimal dengan waktu yang hampir sama cepat (0.000018 detik).
4. Dynamic Programming:
   * Waktu eksekusi (0.000380 detik) lebih tinggi dibanding metode greedy tetapi jauh lebih efisien dibanding brute force.
   * Memberikan solusi optimal untuk dataset ini.
5. **Kesimpulan**
6. **Brute Force Knapsack**

**Kompleksitas Waktu:**

* + Terdapat subset (kombinasi) yang harus diperiksa.
  + Untuk setiap subset, menghitung total berat dan nilai membutuhkan .
  + Sangat tidak efisien untuk ukuran dataset besar karena pertumbuhan eksponensial.

1. **Greedy Knapsack**

**Kompleksitas Waktu:**

* + Tahap pengurutan berdasarkan prioritas (weight, profit, atau density) membutuhkan .
  + Seleksi barang setelah pengurutan dilakukan dalam
  + Sangat cepat untuk dataset besar karena hanya bergantung pada pengurutan.

1. **Dynamic Programming (Bottom-Up)**

**Kompleksitas Waktu:**

* + n: Jumlah barang.
  + W: Kapasitas knapsack.
  + Matriks DP berukuran dan setiap elemen dihitung dalam waktu konstan
  + Lebih efisien dibandingkan brute force, tetapi untuk kapasitas W besar, algoritma ini menjadi tidak praktis.

Dari hasil pengujian dan analisis kompleksitas waktu, Greedy Knapsack adalah algoritma yang lebih bagus dibandingkan dengan Brute Force dan Dynamic Programming dalam hal waktu eksekusi dan praktikalitas. Greedy memiliki kelebihan Waktu eksekusi cepat, algoritma ini memiliki kompleksitas yang memungkinkan eksekusi lebih cepat, bahkan untuk dataset besar. Kemudian Greedy bekerja dengan baik untuk dataset besar karena waktu eksekusinya sangat efisien, Algoritma greedy dapat disesuaikan dengan berbagai mode (berat, profit, atau densitas), memberikan fleksibilitas dalam solusi.

1. **Referensi**
   1. <https://www.lawencon.com/algoritma-brute-force/> - Brute Force
   2. <https://dosen.perbanas.id/menyelesaikan-knapsack-problem-dengan-menggunakan-algoritma-greedy/> - greedy
   3. <https://repository.upi.edu/100398/6/s_d505_045711_chapter1.pdf> - knapsack
   4. Materi perkuliahan bruteforce, greedy, dan dynamic programming.