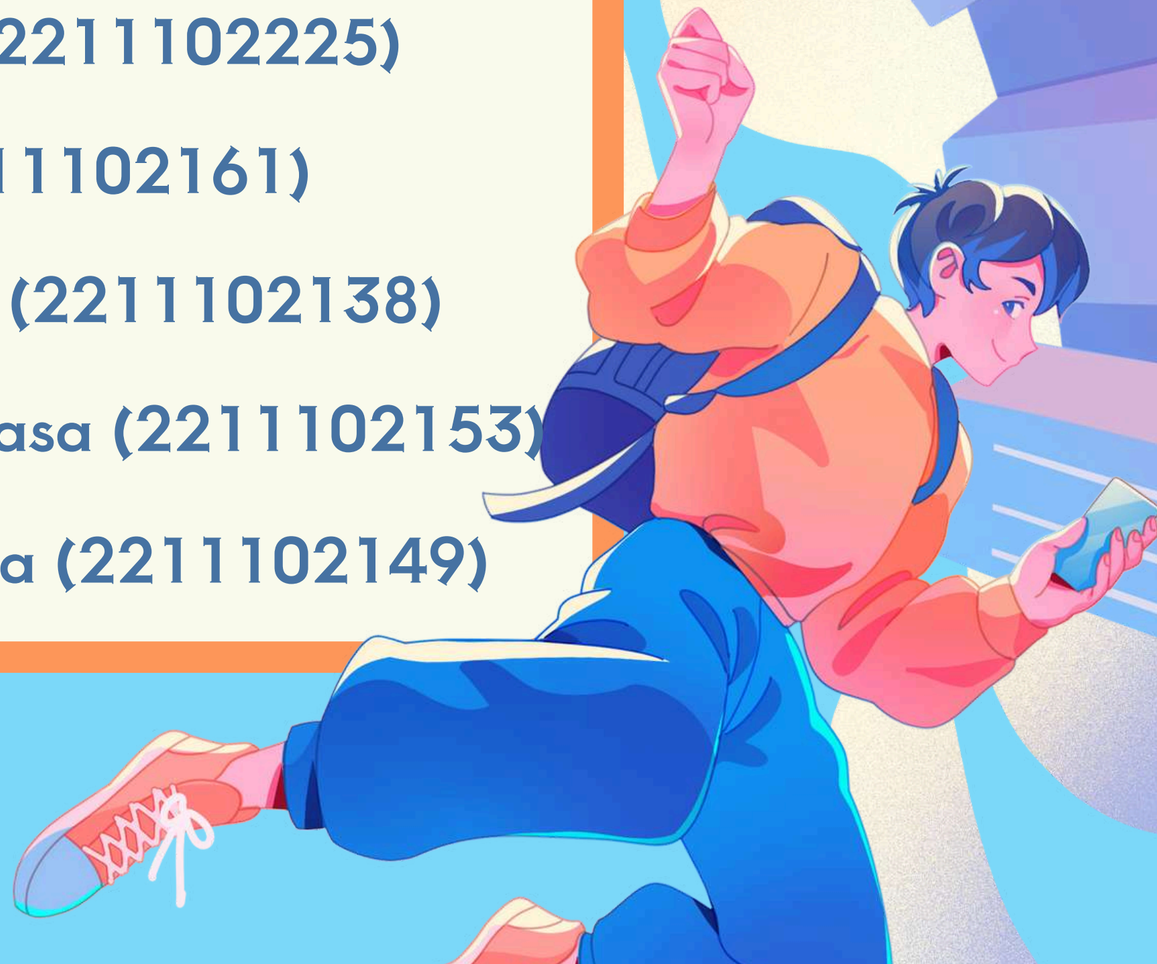
STRATEGI ALGORITMA

OPTIMASI PENGIRIMAN LOGISTIK PADA E-COMMERCE MENGGUNAKAN ALGORITMA DYNAMIC PROGRAMMING, BRANCH AND BOUND, DAN GREEDY

S1-IF-10-05

1. Kyla Azzahra Kinan (2211102225)
2. Hendrik Prayoga (2211102161)
3. Riftian Dimas Adriano (2211102138)
4. Muhammad Hatta Rajasa (2211102153)
4. Arif Pramudia Wardana (2211102149)



Dasar Teori

Optimasi pengiriman logistik dalam e-commerce bertujuan meningkatkan efisiensi operasional dan kepuasan pelanggan melalui penerapan algoritma seperti Dynamic Programming (DP), Branch and Bound, dan Greedy. DP membagi masalah menjadi sub-masalah kecil dengan menyimpan hasil perhitungan untuk efisiensi, cocok untuk kasus seperti Knapsack, guna memaksimalkan nilai barang dalam kapasitas kendaraan. Branch and Bound mengeksplorasi ruang solusi secara sistematis dengan mengeliminasi opsi yang tidak potensial, membantu menemukan kombinasi barang optimal. Sementara itu, algoritma Greedy memilih solusi lokal terbaik di setiap langkah, sering digunakan untuk masalah Fractional Knapsack. Dengan algoritma yang tepat, perusahaan dapat mengelola pengiriman lebih efektif, mengurangi biaya, dan meningkatkan kecepatan pengiriman.

Implementasi

Dynamic Programming

```
FUNCTION knapsack_dp(items, capacity):
    // Mulai pengukuran waktu
    START TIME
    n = LENGTH(items) // Jumlah barang
    W = capacity // Kapasitas knapsack
    // Buat tabel DP dengan ukuran (n+1) x (W+1) dan inisialisasi dengan 0
    CREATE dp TABLE with dimensions (n+1) x (W+1) initialized to 0

    // Mengisi tabel DP
    FOR i FROM 1 TO n: // Iterasi untuk setiap barang
        FOR w FROM 0 TO W: // Iterasi untuk setiap kapasitas
            // Jika berat barang i-1 kurang dari atau sama dengan kapasitas w
            IF weights[i-1] <= w:
                // Ambil maksimum antara tidak mengambil barang dan mengambil barang
                dp[i][w] = MAX(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1])
            ELSE:
                // Jika tidak, tetap gunakan nilai sebelumnya
                dp[i][w] = dp[i-1][w]

    // Menelusuri barang yang diambil
    w = W // Inisialisasi kapasitas
    selected_items = [] // Daftar barang yang dipilih
    total_weight = 0 // Inisialisasi total berat

    // Menelusuri tabel DP dari bawah ke atas
    FOR i FROM n DOWN TO 1:
        // Jika nilai saat ini berbeda dari nilai sebelumnya
        IF dp[i][w] != dp[i-1][w]:
            // Tambahkan barang ke daftar yang dipilih
            selected_items.APPEND(items[i-1].id)
            total_weight += weights[i-1] // Tambahkan berat barang
            w -= weights[i-1] // Kurangi kapasitas

    runtime = END TIME // Menghitung waktu eksekusi
    RETURN selected_items, total_weight, dp[n][W], runtime // Kembalikan hasil
```

Branch and Bound

```
CLASS Node:
    // Inisialisasi node dengan level, nilai, berat, batas, dan barang yang dipilih
    INITIALIZE(level, value, weight, bound, selected):
        SET self.level = level
        SET self.value = value
        SET self.weight = weight
        SET self.bound = bound
        SET self.selected = selected

FUNCTION knapsack_branch_and_bound(items, capacity):
    // Mulai pengukuran waktu
    START TIME
    n = LENGTH(items) // Jumlah barang
    W = capacity // Kapasitas knapsack
    // Buat antrian prioritas untuk menyimpan node
    CREATE priority queue

    // Buat node awal
    u = Node(-1, 0, 0, 0.0, [])
    u.bound = CALCULATE_BOUND(u, n, W, weights, values) // Hitung batas untuk node awal
    max_profit = 0 // Inisialisasi keuntungan maksimum
    best_items = [] // Daftar barang terbaik

    ENQUEUE(u) // Masukkan node awal ke dalam antrian

    // Selama antrian tidak kosong
    WHILE queue is not empty:
        u = DEQUEUE(queue) // Ambil node dengan batas tertinggi
        // Jika batas node lebih besar dari keuntungan maksimum
        IF u.bound > max_profit:
            // Buat node untuk level berikutnya
            v = Node(u.level + 1, u.value, u.weight, 0.0, u.selected[:])
            // Jika level masih dalam batas jumlah barang
            IF v.level < n:
                v.weight = u.weight + weights[v.level] // Tambahkan berat barang
                v.value = u.value + values[v.level] // Tambahkan nilai barang
                v.selected = u.selected + [ids[v.level]] // Tambahkan barang ke daftar yang dipilih
                // Jika berat tidak melebihi kapasitas dan nilai lebih besar dari maksimum
                IF v.weight <= W AND v.value > max_profit:
                    max_profit = v.value // Update keuntungan maksimum
                    best_items = v.selected // Simpan barang terbaik
                v.bound = CALCULATE_BOUND(v, n, W, weights, values) // Hitung batas untuk node baru
                // Jika batas node baru lebih besar dari keuntungan maksimum
                IF v.bound > max_profit:
                    ENQUEUE(queue, v) // Masukkan node baru ke dalam antrian

    runtime = END TIME // Menghitung waktu eksekusi
    total_weight = SUM(weights[item] FOR item IN best_items) // Hitung total berat barang yang dipilih
    RETURN best_items, total_weight, max_profit, runtime // Kembalikan hasil
```

Greedy

```
FUNCTION knapsack_greedy(items, capacity):
    // Mulai pengukuran waktu
    START TIME
    W = capacity // Kapasitas knapsack
    // Urutkan barang berdasarkan rasio nilai terhadap berat secara menurun
    SORT items BY value/weight ratio in descending order

    total_weight = 0 // Inisialisasi total berat
    total_value = 0 // Inisialisasi total nilai
    selected_items = [] // Daftar barang yang dipilih

    // Iterasi melalui barang yang sudah diurutkan
    FOR item IN sorted_items:
        // Jika menambahkan barang tidak melebihi kapasitas
        IF total_weight + item.weight <= W:
            selected_items.APPEND(item.id) // Tambahkan barang ke daftar yang dipilih
            total_weight += item.weight // Tambahkan berat barang
            total_value += item.value // Tambahkan nilai barang

    runtime = END TIME // Menghitung waktu eksekusi
    RETURN selected_items, total_weight, total_value, runtime // Kembalikan hasil
```

Hasil Output n =10

```
Ukuran input 10 data
Kapasitas bobot max : 5000 kg

Dynamic Programming:
Selected Items: BRG00010, BRG00009, BRG00008, BRG00007, BRG00006, BRG00005, BRG00004, BRG00003, BRG00002, BRG00001
Total Weight: 109
Total Value: 2631875
Runtime: 0.025598 seconds


Branch and Bound:
Selected Items: BRG00006, BRG00008, BRG00003, BRG00005, BRG00009, BRG00004, BRG00007, BRG00010, BRG00002, BRG00001
Total Weight: 112.94000000000001
Total Value: 2631875
Runtime: 0.000125 seconds


Greedy:
Selected Items: BRG00006, BRG00008, BRG00003, BRG00005, BRG00009, BRG00004, BRG00007, BRG00010, BRG00002, BRG00001
Total Weight: 112.94000000000001
Total Value: 2631875
Runtime: 0.000009 seconds
```

Hasil Output n = 100

```
Ukuran input 100 data
Kapasitas bobot max : 5000 kg

Dynamic Programming:
Selected Items: BRG00100, BRG00099, BRG00098, BRG00097, BRG00096, BRG00095, BRG00094, BRG00093, BRG00092, BRG00091, BRG00090, BRG00089, BRG00088, BRG00087, BRG00086, BRG00085, BRG00084, BRG0008
Total Weight: 953
Total Value: 23259305
Runtime: 0.265003 seconds


Branch and Bound:
Selected Items: BRG00072, BRG00053, BRG00077, BRG00043, BRG00082, BRG00023, BRG00070, BRG00052, BRG00093, BRG00034, BRG00100, BRG00085, BRG00089, BRG00040, BRG00015, BRG00018, BRG00055, BRG0007
Total Weight: 998.3799999999998
Total Value: 23259305
Runtime: 0.002870 seconds


Greedy:
Selected Items: BRG00072, BRG00053, BRG00077, BRG00043, BRG00082, BRG00023, BRG00070, BRG00052, BRG00093, BRG00034, BRG00100, BRG00085, BRG00089, BRG00040, BRG00015, BRG00018, BRG00055, BRG0007
Total Weight: 998.3799999999998
Total Value: 23259305
Runtime: 0.000049 seconds
```

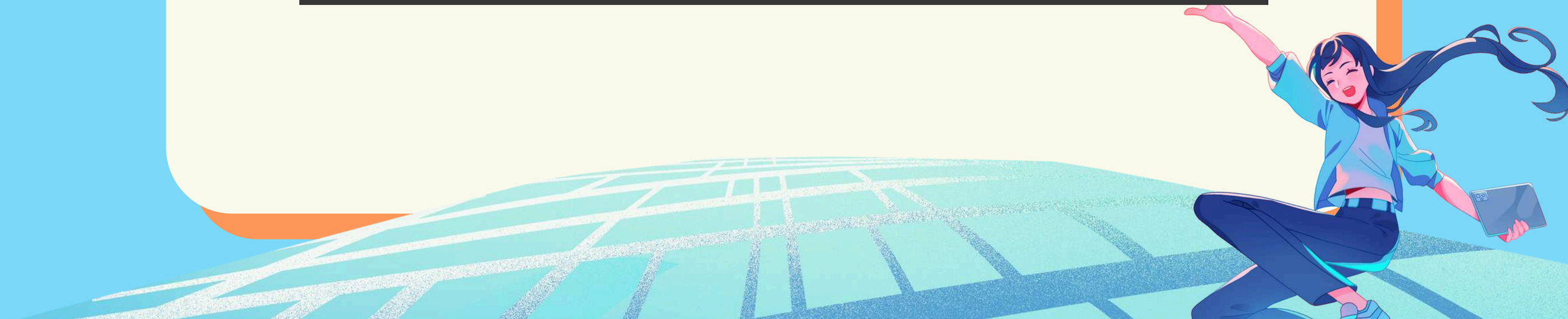
Hasil Output n = 1000

```
Ukuran input 1000 data
Kapasitas bobot max : 5000 kg

Dynamic Programming:
Selected Items: BRG01000, BRG00999, BRG00998, BRG00997, BRG00996, BRG00994, BRG00992, BRG00990, BRG00989, BRG00988, BRG00987, BRG00984, BRG00983, BRG00982, BRG00981, BRG00977, BRG00976, BRG00973
Total Weight: 5000
Total Value: 211071040
Runtime: 4.511589 seconds


Branch and Bound:
Selected Items: BRG00795, BRG00623, BRG00457, BRG00324, BRG00403, BRG00286, BRG00850, BRG00553, BRG00679, BRG00072, BRG00278, BRG00651, BRG00211, BRG00180, BRG00178, BRG00396, BRG00300, BRG00451
Total Weight: 4999.8099999999995
Total Value: 205369139
Runtime: 0.261823 seconds


Greedy:
Selected Items: BRG00795, BRG00623, BRG00457, BRG00324, BRG00403, BRG00286, BRG00850, BRG00553, BRG00679, BRG00072, BRG00278, BRG00651, BRG00211, BRG00180, BRG00178, BRG00396, BRG00300, BRG00451
Total Weight: 4999.499999999999
Total Value: 205354949
Runtime: 0.000900 seconds
```



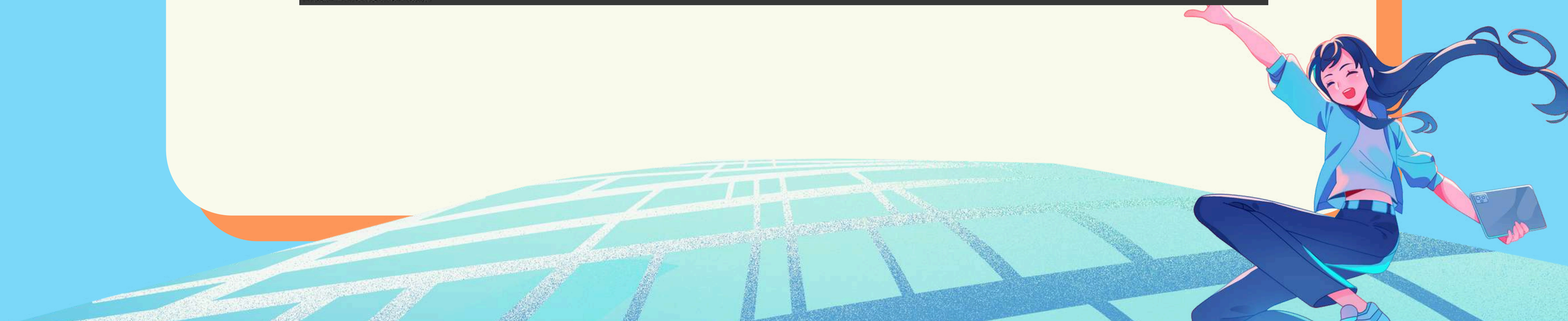
Hasil Output n = 5000

```
Ukuran input 5000 data
Kapasitas bobot max : 5000 kg

Dynamic Programming:
Selected Items: BRG05000, BRG04998, BRG04997, BRG04996, BRG04995, BRG04992, BRG04984, BRG04979, BRG04976, BRG04973, BRG04969, BRG04966, BRG04965, BRG04964, BRG04959, BRG04951, BRG04948, BRG04947
Total Weight: 5000
Total Value: 488564611
Runtime: 16.456330 seconds


Branch and Bound:
Selected Items: BRG04022, BRG01665, BRG01076, BRG02004, BRG00795, BRG00623, BRG03695, BRG00457, BRG02569, BRG01694, BRG02257, BRG02784, BRG00324, BRG03878, BRG02428, BRG00403, BRG00286, BRG04383
Total Weight: 4999.999999999997
Total Value: 455713216
Runtime: 1.107481 seconds


Greedy:
Selected Items: BRG04022, BRG01665, BRG01076, BRG02004, BRG00795, BRG00623, BRG03695, BRG00457, BRG02569, BRG01694, BRG02257, BRG02784, BRG00324, BRG03878, BRG02428, BRG00403, BRG00286, BRG04383
Total Weight: 4999.989999999998
Total Value: 455704515
Runtime: 0.004324 seconds
```



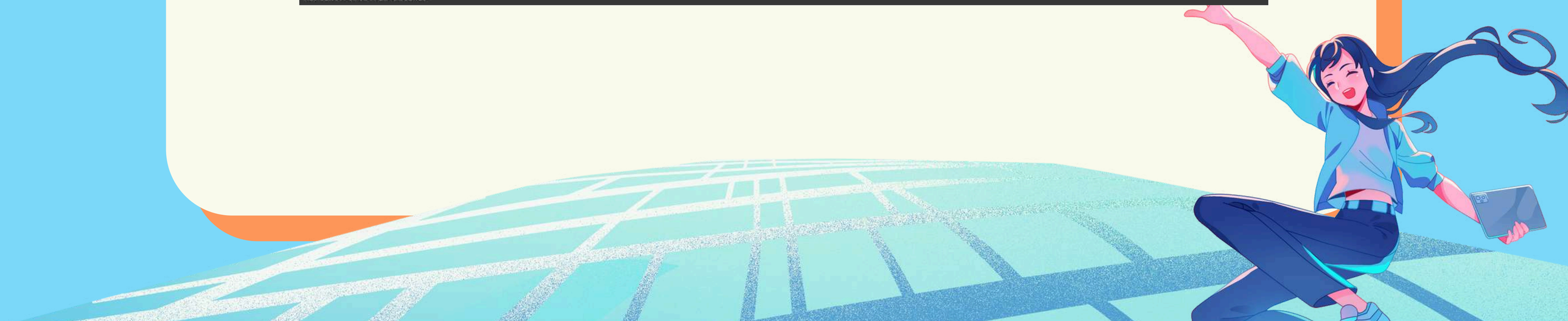
Hasil Output n = 10000

```
Ukuran input 10000 data
Kapasitas bobot max : 5000 kg

Dynamic Programming:
Selected Items: BRG10000, BRG09999, BRG09998, BRG09994, BRG09993, BRG09991, BRG09985, BRG09983, BRG09979, BRG09969, BRG09963, BRG09962, BRG09959, BRG09935, BRG09922, BRG09913, BRG09908, BRG09902
Total Weight: 5000
Total Value: 705975852
Runtime: 35.500866 seconds


Branch and Bound:
Selected Items: BRG04022, BRG05215, BRG01665, BRG07389, BRG01076, BRG09587, BRG02004, BRG00795, BRG07417, BRG08617, BRG00623, BRG03695, BRG09717, BRG06957, BRG06281, BRG00457, BRG02569, BRG05574
Total Weight: 4999.990000000003
Total Value: 640500242
Runtime: 1.117179 seconds


Greedy:
Selected Items: BRG04022, BRG05215, BRG01665, BRG07389, BRG01076, BRG09587, BRG02004, BRG00795, BRG07417, BRG08617, BRG00623, BRG03695, BRG09717, BRG06957, BRG06281, BRG00457, BRG02569, BRG05574
Total Weight: 4999.990000000000025
Total Value: 640484456
Runtime: 0.004783 seconds
```

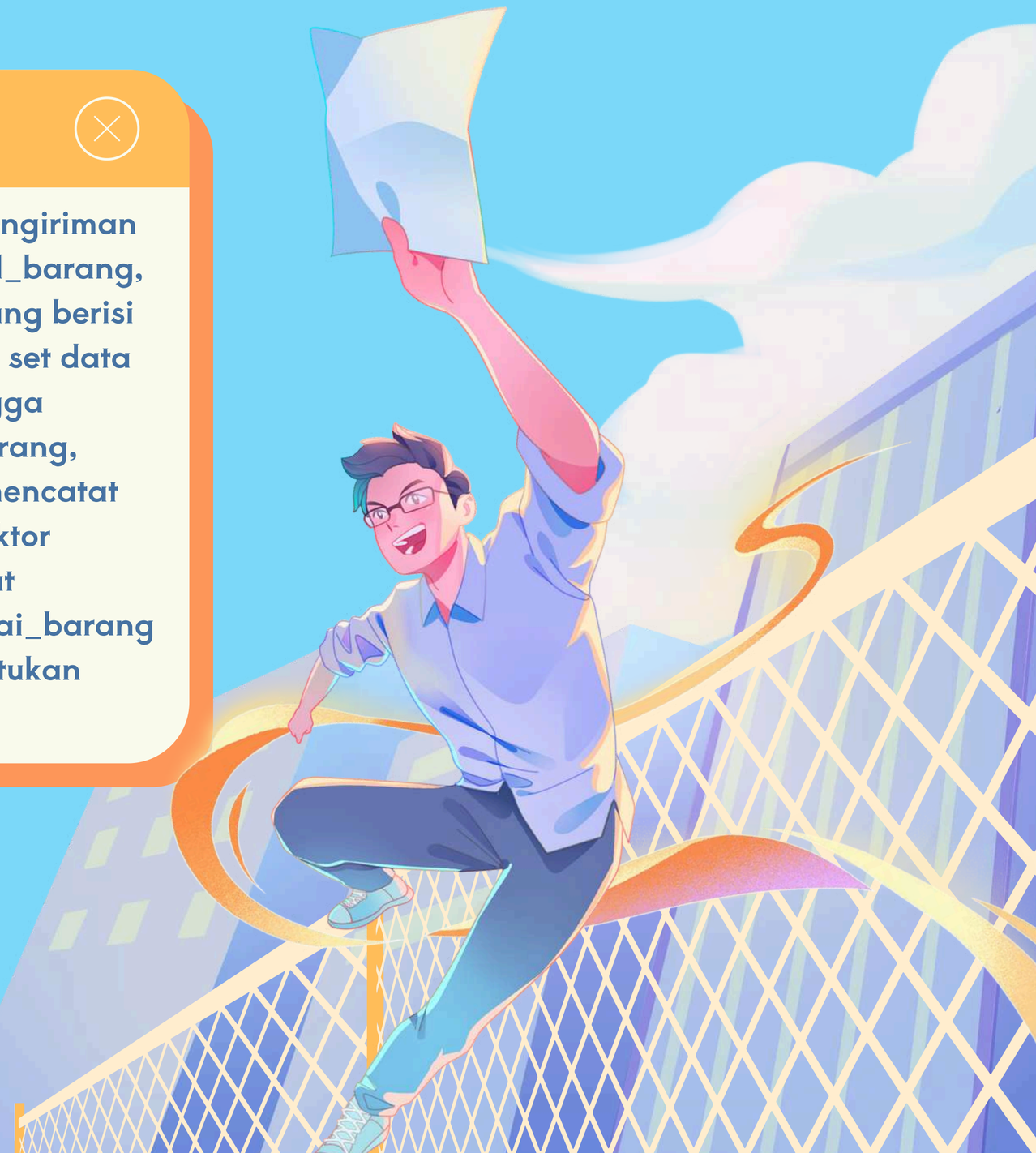


Pengujian

Data

Data yang digunakan untuk pengujian dalam optimasi pengiriman logistik pada e-commerce terdiri atas tiga kolom utama: id_barang, berat, dan nilai_barang. Data ini diambil dari file CSV yang berisi informasi tentang barang-barang yang akan diuji. Setiap set data memiliki jumlah barang yang berbeda, mulai dari 10 hingga 10.000.Kolom id_barang berisi kode unik untuk setiap barang, yang digunakan sebagai identitas barang. Kolom berat mencatat bobot masing-masing barang dalam kg, yang menjadi faktor penting dalam menentukan apakah barang tersebut dapat dimasukkan ke dalam knapsack. Sementara itu, kolom nilai_barang merepresentasikan nilai atau harga barang, untuk menentukan prioritas pengiriman.

	id_barang	berat	nilai_barang
0	BRG00001	17.26	109753
1	BRG00002	13.21	99982
2	BRG00003	8.57	437287
3	BRG00004	15.99	343799
4	BRG00005	10.26	324152

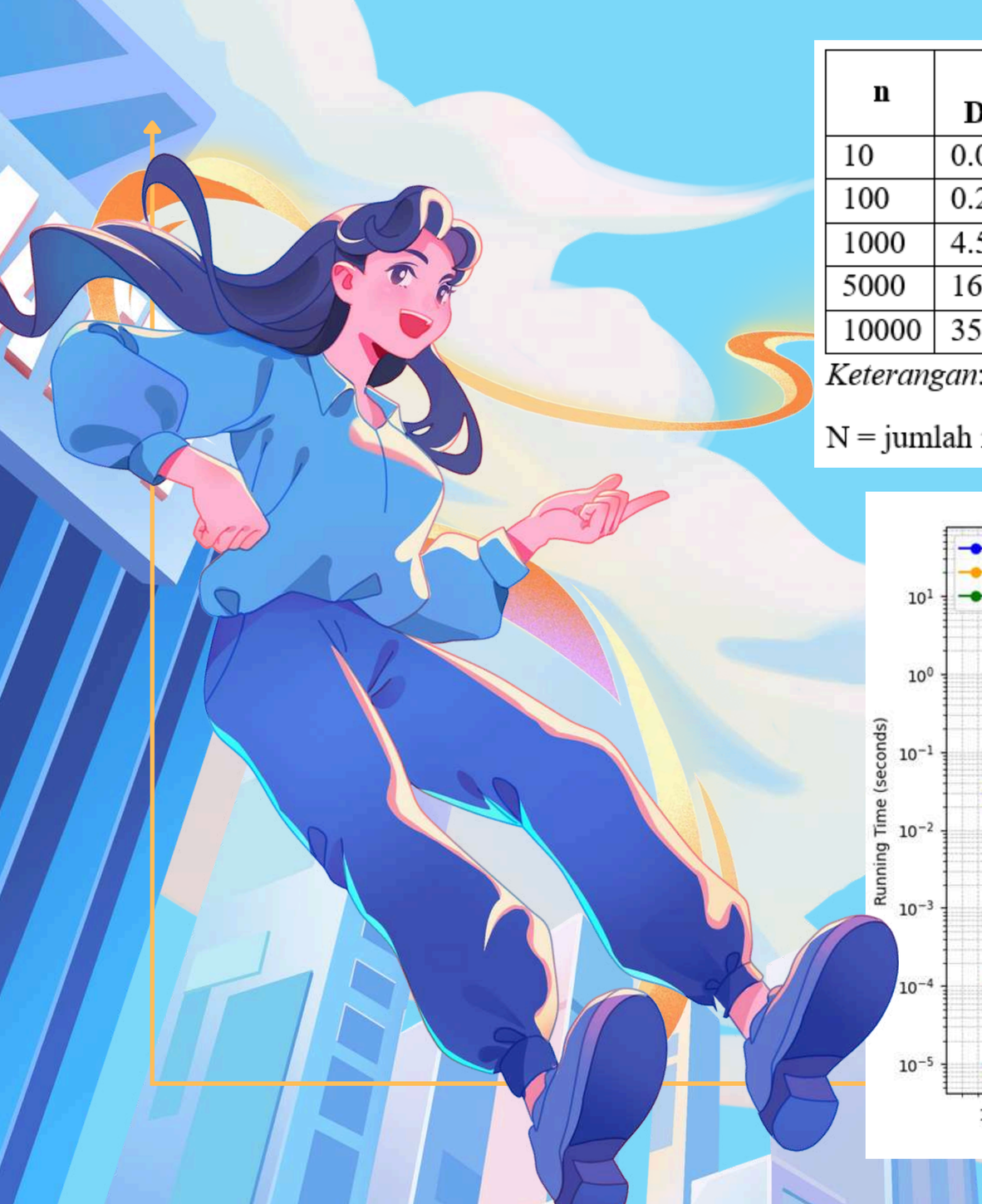


Data

Pengujian dilakukan dengan kapasitas knapsack yang sama, yaitu 5.000 kg, untuk semua ukuran input. Variasi ukuran input yang digunakan dalam pengujian adalah 10, 100, 1.000, 5.000, dan 10.000 barang. Setiap pengujian dilakukan beberapa kali untuk setiap ukuran input guna mendapatkan hasil yang konsisten dan akurat. Waktu eksekusi (running time) dicatat untuk setiap algoritma—Dynamic Programming, Branch and Bound, dan Greedy—untuk setiap ukuran input.

Hardware dan Software

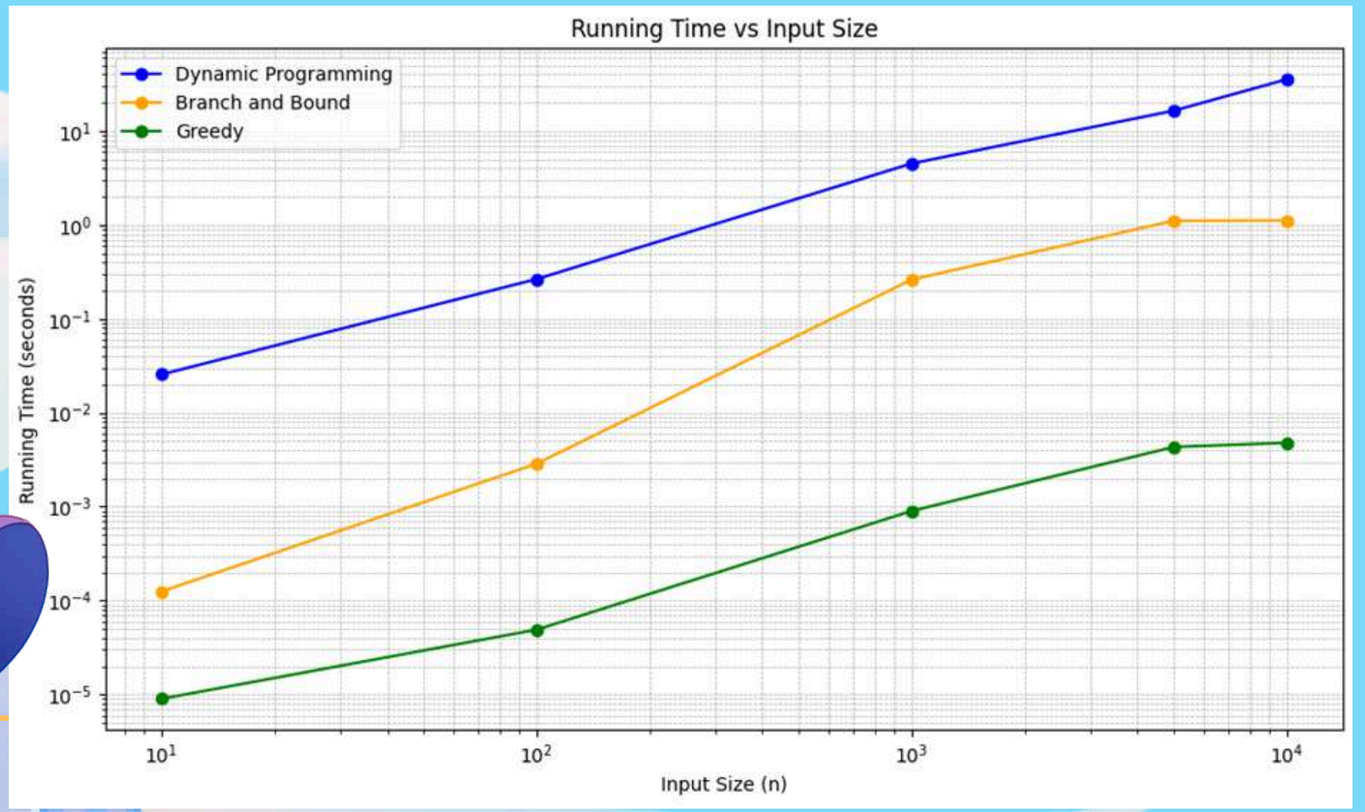
Pengujian dilakukan dengan kapasitas knapsack yang sama, yaitu 5.000 kg, untuk semua ukuran input. Variasi ukuran input yang digunakan dalam pengujian adalah 10, 100, 1.000, 5.000, dan 10.000 barang. Setiap pengujian dilakukan beberapa kali untuk setiap ukuran input guna mendapatkan hasil yang konsisten dan akurat. Waktu eksekusi (running time) dicatat untuk setiap algoritma—Dynamic Programming, Branch and Bound, dan Greedy—untuk setiap ukuran input.



n	Waktu Eksekusi Dynamic Programming	Waktu Eksekusi Branch and Bound	Waktu Eksekusi Greedy
10	0.025598	0.000125	0.000009
100	0.265003	0.002870	0.000049
1000	4.511589	0.261823	0.000900
5000	16.456330	1.107481	0.004324
10000	35.500866	1.117179	0.004783

Keterangan:

N = jumlah input



Hasil

Hasil pengujian menunjukkan bahwa waktu eksekusi untuk algoritma Dynamic Programming meningkat secara signifikan seiring dengan bertambahnya ukuran input, mencerminkan kompleksitas waktu yang lebih tinggi. Sebaliknya, algoritma Greedy menunjukkan waktu eksekusi yang sangat rendah di semua ukuran input, menjadikannya pilihan yang cepat meskipun tidak selalu optimal. Algoritma Branch and Bound menunjukkan waktu eksekusi yang lebih baik dibandingkan dengan Dynamic Programming, terutama pada ukuran input yang lebih besar, tetapi masih lebih lambat dibandingkan dengan Greedy.

Analisis Hasil Pengujian

Hasil Analisis Pengujian

Hasil pengujian menunjukkan bahwa pada jumlah input kecil (n = 10), semua algoritma memiliki waktu eksekusi cepat, dengan Dynamic Programming membutuhkan 0.025598 detik, Branch and Bound 0.000125 detik, dan Greedy hanya 0.000009 detik. Saat input meningkat ke n = 100, Dynamic Programming memakan waktu 0.265003 detik, Branch and Bound 0.002870 detik, dan Greedy tetap cepat di 0.000049 detik. Pada n = 1000, Dynamic Programming memakan 4.511589 detik, Branch and Bound 0.261823 detik, dan Greedy 0.000900 detik. Untuk n = 5000, Dynamic Programming naik signifikan ke 16.456330 detik, Branch and Bound 1.107481 detik, sementara Greedy hanya 0.004324 detik. Pada input terbesar (n = 10,000), Dynamic Programming membutuhkan 35.500866 detik, Branch and Bound 1.117179 detik, dan Greedy 0.004783 detik. Dengan demikian, algoritma Greedy unggul dalam efisiensi waktu di semua ukuran input data, sementara Branch and Bound menawarkan keseimbangan terbaik antara akurasi dan efisiensi pada skala besar, sedangkan Dynamic Programming kurang efisien untuk dataset besar.

Referensi

[1] M. N. Zein et al., “Penerapan Program Dinamis Untuk Menentukan Jalur Yang Optimum Dalam Pengiriman Benih Ikan Ceps Aquarium,” Bull. Appl. Ind. Eng. Theory, vol. 3, no. 1, pp. 34–39, 2022, [Online]. Available: http://www.jim.unindra.ac.id/index.php/baiet/article/view/6526%0Ahttp://www.jim.unindra.ac.id/index.php/baiet/article/viewFile/6526/880

[2] A. A. Prasha, C. O. Rachmadi, N. G. Raditya, and S. L. Mutiara, “Implementasi Algoritma Greedy dan Dynamic Programming untuk Masalah Implementasi Algoritma Greedy dan Dynamic Programming untuk Masalah Penjadwalan Interval dengan Model Knapsack,” FORMAT J. Ilm. Tek. Inform., vol. 13, no. 02, pp. 166–180, 2024, doi: 10.22441/format.2024.v13.i2.005.

[3] A. A. Salim and A. Alfian, “Perencanaan Produksi Untuk Mengoptimalkan Keuntungan Dengan Metode Branch And Bound Di UKM ‘X,’” SAINTEK J. Ilm. Sains dan Teknol. Ind., vol. 5, no. 1, pp. 30–38, 2021, doi: 10.32524/saintek.v5i1.250.

[4] F. D. Sianipar et al., “ESTIMASI RUTE TERDEKAT DARI UNIVERSITAS NEGERI MEDAN KE SPBU TERDEKAT MENGGUNAKAN ALGORITMA GREEDY,” JATI(Jurnal Mhs. Tek. Inform., vol. 8, no. 6, pp. 12218–12225, 2024.


Terima kasih
telah
Menonton!