

LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL KE-11

ALGORITMA GREEDY



Disusun Oleh:

Nama : Oktario Mufti Yudha
NPM : 2320506044
Kelas : 04 (Empat)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

I. Tujuan Praktikum

1. Mahasiswa mampu memahami Algoritma Greedy pada python
2. Mahasiswa mampu menerapkan Algoritma Greedy pada python


II. Dasar Teori

Algoritma greedy adalah pendekatan untuk memecahkan masalah dengan membuat keputusan secara bertahap, memilih opsi terbaik yang tersedia pada setiap langkah tanpa mempertimbangkan dampak jangka panjang. Inti dari algoritma greedy adalah selalu memilih pilihan yang tampak paling optimal pada saat itu. Karakteristik Utama dari algoritma greedy adalah sebagai berikut :

1. Pemilihan Lokal: Setiap keputusan didasarkan pada informasi lokal, tanpa memperhatikan keseluruhan struktur masalah.
2. Solusi Optimal Lokal: Algoritma selalu memilih solusi yang tampak paling optimal secara lokal di setiap langkah.
3. Kepastian: Algoritma greedy tidak selalu menjamin solusi optimal secara global untuk semua jenis masalah, tetapi dalam kasus tertentu (misalnya, masalah dengan properti optimal substruktur), algoritma ini dapat memberikan solusi optimal global.

III. Hasil dan Pembahasan

a. Algoritma Greedy



```
# Algoritma Greedy

def printMaxActivities(s, f):
    n = len(f)
    print("the following activities are selected")
    i = 0
    print(i)
    for j in range(n):
        if s[j] >= f[i]:
            print(j)
            i = j

s = [1, 3, 0, 5, 8, 5]
f = [2, 4, 6, 7, 9, 9]
printMaxActivities(s, f)

the following activities are selected
0
1
3
4
```

Gambar 3.1: Algoritma Greedy

1. Fungsi `printMaxActivities(s, f)`: Fungsi ini digunakan untuk mencetak indeks aktivitas yang dipilih berdasarkan algoritma greedy. Fungsi ini menerima dua parameter, yaitu `s` yang merupakan list berisi waktu mulai setiap aktivitas, dan `f` yang merupakan list berisi waktu selesai setiap aktivitas.
2. `n = len(f)`: Mendapatkan jumlah total aktivitas.
3. `print("the following activities are selected")`: Mencetak pesan bahwa aktivitas berikut akan dipilih.
4. `i = 0`: Mengatur indeks aktivitas pertama yang dipilih adalah 0.
5. `print(i)`: Mencetak indeks aktivitas pertama yang dipilih.
6. Loop `for j in range(n)`: Melakukan iterasi untuk setiap indeks aktivitas.
7. `if s[j] >= f[i]`: Jika waktu mulai aktivitas `j` lebih besar atau sama dengan waktu selesai aktivitas `i`, maka aktivitas `j` dapat dipilih.
8. `print(j)`: Mencetak indeks aktivitas `j` yang dipilih.
9. `i = j`: Mengatur indeks aktivitas yang baru dipilih adalah `j`.
10. `s = [1, 3, 0, 5, 8, 5]`, `f = [2, 4, 6, 7, 9, 9]`: Mendefinisikan waktu mulai dan waktu selesai setiap aktivitas.
11. `printMaxActivities(s, f)`: Menjalankan fungsi `printMaxActivities` dan mencetak indeks aktivitas yang dipilih.

b. Knapsack Pecahan dengan Algoritma Greedy

```
# Knapsack Pecahan dengan Algoritma Greedy
class Item:
    def __init__(self, weight, value):
        self.weight = weight
        self.value = value
        self.ratio = value / weight

def fractional_knapsack(items, capacity):
    items.sort(key = lambda x: x.ratio, reverse = True)
    total_value = 0
    remaining_capacity = capacity
    for item in items:
        if remaining_capacity <= 0:
            break
        fraction = min(1, remaining_capacity / item.weight)
        total_value += fraction * item.value
        remaining_capacity -= fraction * item.weight
    return round(total_value, 2)

items = [Item(10, 60), Item(20, 100), Item(30, 120)]
capacity = 50
print("maximum value that can be obtained is", fractional_knapsack(items, capacity))

maximum value that can be obtained is 240.0
```

Gambar 3.1: Knapsack Pecahan

1. Kelas Item: Kelas ini digunakan untuk merepresentasikan sebuah item dengan atribut weight (berat), value (nilai), dan ratio (rasio nilai per berat).
2. Fungsi fractional_knapsack(items, capacity): Fungsi ini digunakan untuk mencari nilai maksimum yang dapat diperoleh dari item-item yang dimasukkan ke dalam knapsack dengan kapasitas tertentu. Fungsi ini menerima dua parameter, yaitu items yang merupakan list berisi objek Item, dan capacity yang merupakan kapasitas knapsack.
3. items.sort(key = lambda x: x.ratio, reverse = True): Mengurutkan item-item berdasarkan rasio nilai per berat secara menurun.
4. total_value = 0, remaining_capacity = capacity: Menginisialisasi nilai total dan kapasitas yang tersisa.
5. Loop for item in items: Melakukan iterasi untuk setiap item.
6. if remaining_capacity <= 0: Jika kapasitas yang tersisa sudah habis, maka hentikan iterasi.
7. fraction = min(1, remaining_capacity / item.weight): Menghitung fraksi item yang akan dimasukkan ke dalam knapsack.

8. `total_value += fraction * item.value, remaining_capacity -= fraction * item.weight`: Menambahkan nilai item ke total nilai dan mengurangi berat item dari kapasitas yang tersisa.
9. `return round(total_value, 2)`: Mengembalikan nilai total yang telah dibulatkan hingga dua angka di belakang koma.
10. `items = [Item(10, 60), Item(20, 100), Item(30, 120)], capacity = 50`: Mendefinisikan item-item dan kapasitas knapsack.
11. `print("maximum value that can be obtained is", fractional_knapsack(items, capacity))`: Menjalankan fungsi `fractional_knapsack` dan mencetak nilai maksimum yang dapat diperoleh.

IV. Latihan

a. Latihan 1

Jelaskan apa saja perbedaan dari penerapan algoritma greedy untuk pemilihan aktivitas berikut ini dibandingkan dengan yang ada pada halaman 4-5 diatas !

```
# Latihan 1
'''
Pengurutan:
-Halaman 4-5: Tidak ada pengurutan.
-Kode pada Gambar: Mengurutkan aktivitas berdasarkan waktu selesai.

Data:
-Halaman 4-5: Menggunakan dua daftar (s dan f).
-Kode pada Gambar: Menggunakan dictionary (data).

Output:
-Halaman 4-5: Mencetak indeks aktivitas.
-Kode pada Gambar: Mencetak nama aktivitas.
'''
```

Gambar 4.1: Latihan1

b. Latihan 2

Diberikan n item dengan bobot berbeda dan masing-masing wadah berkapasitas c , masukkan setiap item ke dalam wadah sedemikian rupa sehingga jumlah total wadah yang digunakan dapat diminimalkan. Dapat diasumsikan bahwa semua item memiliki bobot lebih kecil dari kapasitas sampah.

```
# Latihan 2
def binPacking(berat, c):
    berat.sort(reverse=True)
    bins = [c] * len(berat)
    jumlah_bin = 0

    for i in range(len(berat)):
        j = 0

        while(j < jumlah_bin):
            if bins[j] >= berat[i]:
                bins[j] -= berat[i]
                break
            j += 1

        if j == jumlah_bin:
            bins[jumlah_bin] -= berat[i]
            jumlah_bin += 1

    return jumlah_bin

berat = [4, 8, 1, 4, 2, 1]
c = 10
print(binPacking(berat, c)) # Output: 2

berat = [9, 8, 2, 2, 5, 4]
c = 10
print(binPacking(berat, c)) # Output: 4
```



Gambar 4.3: Latihan 2

1. Fungsi `binPacking(berat, c)`: Fungsi ini digunakan untuk mencari jumlah bin minimum yang diperlukan untuk menyimpan item-item dengan berat tertentu. Fungsi ini menerima dua parameter, yaitu `berat` yang merupakan list berisi berat setiap item, dan `c` yang merupakan kapasitas setiap bin.
2. `berat.sort(reverse=True)`: Mengurutkan berat item secara menurun.
3. `bins = [c] * len(berat)`, `jumlah_bin = 0`: Menginisialisasi list `bins` yang berisi kapasitas setiap bin dan variabel `jumlah_bin` yang menyimpan jumlah bin yang digunakan.
4. Loop `for i in range(len(berat))`: Melakukan iterasi untuk setiap item.

5. Loop `while(j < jumlah_bin)`: Melakukan iterasi selama `j` kurang dari `jumlah_bin`.
6. `if bins[j] >= berat[i]`: Jika kapasitas bin `j` cukup untuk menyimpan item `i`, maka kurangi kapasitas bin `j` dengan berat item `i` dan hentikan iterasi.
7. `if j == jumlah_bin`: Jika tidak ada bin yang cukup untuk menyimpan item `i`, maka gunakan bin baru dan kurangi kapasitas bin tersebut dengan berat item `i`.
8. `return jumlah_bin`: Mengembalikan jumlah bin minimum yang diperlukan.
9. `berat = [4, 8, 1, 4, 2, 1], c = 10, print(binPacking(berat, c))`: Mendefinisikan berat item dan kapasitas bin, lalu menjalankan fungsi `binPacking` dan mencetak jumlah bin minimum yang diperlukan.
10. `berat = [9, 8, 2, 2, 5, 4], c = 10, print(binPacking(berat, c))`: Mendefinisikan berat item dan kapasitas bin, lalu menjalankan fungsi `binPacking` dan mencetak jumlah bin minimum yang diperlukan.

V. Kesimpulan

Praktikum ini memberikan wawasan yang mendalam tentang konsep dan penerapan Algoritma Greedy dalam bahasa Python. Algoritma Greedy menyelesaikan masalah dengan mengambil keputusan optimal pada setiap langkah tanpa mempertimbangkan dampak jangka panjang. Melalui implementasi fungsi seperti `printMaxActivities` untuk pemilihan aktivitas dan `fractional_knapsack` untuk masalah knapsack pecahan, praktikum ini menunjukkan bagaimana Algoritma Greedy dapat digunakan dalam berbagai konteks. Pemilihan aktivitas menunjukkan bagaimana keputusan lokal dapat mempengaruhi hasil akhir, sedangkan masalah knapsack pecahan menekankan pentingnya rasio nilai per berat untuk mencapai solusi optimal. Selain itu, latihan mengenai penempatan item dalam wadah dengan kapasitas terbatas menunjukkan penerapan Algoritma Greedy dalam mengoptimalkan penggunaan ruang. Keseluruhan praktikum ini memberikan pemahaman yang kuat tentang

prinsip-prinsip dasar Algoritma Greedy dan bagaimana algoritma ini dapat diterapkan untuk menyelesaikan berbagai jenis masalah secara efisien.