

LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL KE-11

ALGORITMA GREEDY



Disusun Oleh:

Nama	Restu Wibisono
NPM	2340506061
Kelas	03 (Tiga)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

I. Tujuan Praktikum

Adapun tujuan praktikum ini sebagai berikut :

1. Mahasiswa mampu menerapkan algoritma greedy pada kasus knapsack menggunakan bahasa pemrograman python

II. Dasar Teori

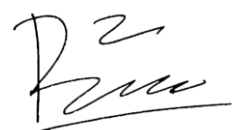
Masalah Knapsack adalah salah satu masalah optimasi klasik dalam ilmu komputer dan teori keputusan. Masalah Knapsack sering digunakan dalam konteks seperti alokasi sumber daya, pemilihan portofolio investasi, dan pengemasan barang. Algoritma dinamis dan teknik pemrograman lainnya digunakan untuk menemukan solusi optimal atau mendekati optimal untuk masalah ini, menjadikannya topik yang menarik dan penting dalam studi algoritma dan optimasi.

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

- Pemilihan Lokal Setiap keputusan didasarkan pada informasi lokal, tanpa memperhatikan keseluruhan struktur masalah.
- Solusi Optimal Lokal Algoritma selalu memilih solusi yang tampak paling optimal secara lokal di setiap langkah.
- 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.

Algoritma Dijkstra menggunakan prinsip greedy untuk memastikan bahwa setiap langkah yang diambil adalah optimal secara lokal, yang pada akhirnya memberikan solusi optimal global untuk masalah jarak terpendek pada graf dengan bobot non-negatif. Walaupun tidak semua masalah yang

Tanda Tangan



menggunakan algoritma greedy akan memberikan solusi optimal global, algoritma Dijkstra adalah contoh di mana prinsip greedy bekerja dengan baik dan memberikan solusi optimal global.

III. Hasil dan Pembahasan

1. Algoritma Greedy 1

```
# Algoritma Greedy

def coin_change_greedy(n):
    coins = [20, 10, 5, 1]
    i = 0

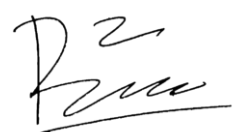
    while n > 0:
        if (coins[i] > n):
            i += 1
        else:
            print(coins[i])
            n -= coins[i];
        print("\n\n")
    coin_change_greedy(57)

✓ 0.0s Python
```

20
20
10
5
1
1

- ‘def coin_change_greedy(n):’ Mendefinisikan fungsi ‘coin_change_greedy’ dengan parameter ‘n’ (jumlah uang).
- ‘coins = [20, 10, 5, 1]’ Daftar nilai koin, diurutkan dari terbesar ke terkecil.
- ‘i = 0’ Indeks untuk mengakses elemen dalam daftar ‘coins’.
- ‘while n > 0:’ Loop berjalan selama ‘n’ lebih besar dari 0.
- ‘if (coins[i] > n):’ Memeriksa apakah nilai koin pada indeks ‘i’ lebih besar dari ‘n’.
- ‘i += 1’ Jika nilai koin lebih besar dari ‘n’, pindah ke koin lebih kecil.
- ‘else:’ Jika nilai koin tidak lebih besar dari ‘n’, jalankan blok ini.
- ‘print(coins[i])’ Mencetak nilai koin yang dipilih.
- ‘n -= coins[i]’ Kurangi ‘n’ dengan nilai koin yang dipilih.

Tanda Tangan



- ‘print("\n\n")’ Mencetak dua baris baru sebagai pemisah output.
- ‘coin_change_greedy(57)’ Memanggil fungsi dengan parameter ‘57’..

2. Algoritma Greedy 2

```
# n → Jumlah total kegiatan
# s[] → Array yang berisi waktu mulai semua aktivitas
# f[] → Array yang berisi waktu selesai semua aktivitas

def printMaxActivites(s, f):
    n = len(f)
    print("The following activites are selected")

    # aktivitas pertama selalu terpilih duluan
    i = 0
    print(i),

    # Pertimbangkan aktivitas lainnya
    for j in range(n):
        # Jika aktivitas ini memiliki waktu mulai lebih besar
        # atau sama dengan waktu selesai aktivitas sebelumnya,
        # maka pilih aktivitas tersebut
        if s[j] ≥ f[i]:
            print(j),
            i = j

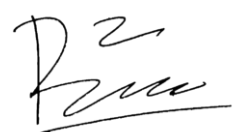
# Program untuk cek pemanggilan fungsi di atas
s = [1, 3, 0, 5, 8, 5]
f = [2, 4, 6, 7, 9, 9]
printMaxActivites(s, f)
```

✓ 0.0s Python

```
The following activites are selected
0
1
3
4
```

- ‘def printMaxActivites(s, f):’ Mendefinisikan fungsi ‘printMaxActivites’ yang menerima dua parameter ‘s’ (waktu mulai) dan ‘f’ (waktu selesai).
- ‘n = len(f)’ Menghitung jumlah total kegiatan (‘n’).
- ‘print("The following activites are selected")’ Mencetak pesan untuk menandakan kegiatan yang dipilih.
- ‘i = 0’ Memilih aktivitas pertama (indeks 0) sebagai aktivitas yang selalu terpilih.
- ‘print(i),’ Mencetak indeks aktivitas pertama yang terpilih.

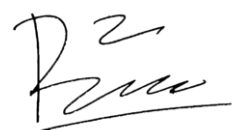
Tanda Tangan



- 'for j in range(n):' Memulai loop untuk memeriksa setiap aktivitas dari indeks 0 hingga 'n-1'.
- 'if s[j] >= f[i]:' Memeriksa apakah waktu mulai aktivitas 'j' lebih besar atau sama dengan waktu selesai aktivitas 'i'.
- 'print(j),' Jika kondisi terpenuhi, mencetak indeks aktivitas 'j'.
- 'i = j' Memperbarui 'i' ke 'j' untuk mempertimbangkan aktivitas berikutnya.
-
- ### Contoh Penggunaan
- - 's = [1, 3, 0, 5, 8, 5]' Daftar waktu mulai kegiatan.
- - 'f = [2, 4, 6, 7, 9, 9]' Daftar waktu selesai kegiatan.
- - 'printMaxActivites(s, f)' Memanggil fungsi untuk mencetak indeks kegiatan yang dipilih berdasarkan algoritma greedy.

3. Knapsack (0/1) dengan pemrograman dinamis:

Tanda Tangan



```
# Knapsack (0/1) dengan pemrograman dinamis

def knapSack(W, wt, val, n):

    # Kondisi dasar
    if n == 0 or W == 0:
        return 0

    if t[n][W] != -1:
        return t[n][W]

    # Diagram pilihan
    if wt[n-1] <= W:
        t[n][W] = max(
            val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
            knapSack(W, wt, val, n-1)
        )
    else:
        t[n][W] = knapSack(W, wt, val, n-1)
    return t[n][W]

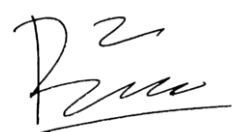
# Driver code
if __name__ == "__main__":
    val = [60, 100, 120]
    wt = [10, 20, 30]
    W = 50
    n = len(val)

    # We initialize the matrix with -1 at first
    t = [[-1 for _ in range(W + 1)] for _ in range(n + 1)]
    print("Maximum Value that can be Obtained is", knapSack(W, wt, val, n))

✓ 0.0s Python
Maximum Value that can be Obtained is 220
```

- ‘def knapSack(W, wt, val, n):’ Mendefinisikan fungsi ‘knapSack’ untuk menyelesaikan masalah Knapsack (0/1) dengan pemrograman dinamis.
- ‘if n == 0 or W == 0 return 0’ Kondisi dasar, jika tidak ada item atau kapasitas knapsack adalah 0, maka nilai maksimum adalah 0.
- ‘if t[n][W] != -1 return t[n][W]’ Jika nilai sudah dihitung sebelumnya, kembalikan nilai yang sudah disimpan di ‘t[n][W]’.
- ‘if wt[n-1] <= W:’ Memeriksa apakah berat item ke-n dapat dimasukkan ke dalam knapsack.
- ‘t[n][W] = max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1), knapSack(W, wt, val, n-1))’ Menghitung nilai maksimum dengan dua pilihan memasukkan item ke-n atau tidak memasukkannya.

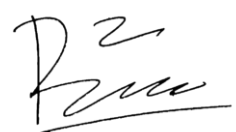
Tanda Tangan



- `'return t[n][W]'` Mengembalikan nilai maksimum yang dihitung dan menyimpannya di `'t[n][W]'`.
- `'else t[n][W] = knapSack(W, wt, val, n-1)'` Jika berat item ke-n lebih besar dari kapasitas knapsack, lewati item tersebut.
- `'return t[n][W]'` Mengembalikan nilai maksimum yang dihitung dan menyimpannya di `'t[n][W]'`.
- `'if __name__ == "__main__":'` Memeriksa apakah skrip dijalankan secara langsung.
- `'val = [60, 100, 120]'` Daftar nilai dari setiap item.
- `'wt = [10, 20, 30]'` Daftar berat dari setiap item.
- `'W = 50'` Kapasitas maksimum knapsack.
- `'n = len(val)'` Jumlah item.
- `'t = [[-1 for _ in range(W + 1)] for _ in range(n + 1)]'` Menginisialisasi matriks `'t'` dengan nilai -1 untuk menyimpan nilai maksimum yang sudah dihitung.
- `'print("Maximum Value that can be Obtained is", knapSack(W, wt, val, n))'` Memanggil fungsi `'knapSack'` dan mencetak nilai maksimum yang dapat diperoleh.

4. Knapsack Pecahan dengan Algoritma Greedy

Tanda Tangan



```

# Knapsack Pecahan dengan Algoritma Greedy

class Item:
    def __init__(self, weight, value):
        self.weight = weight
        self.value = value
        # Hitung rasio nilai terhadap berat untuk setiap item
        self.ratio = value / weight

def fractional_knapsack(items, capacity):
    # Urutkan item berdasarkan rasio nilai terhadap berat dalam urutan men
    items.sort(key=lambda x: x.ratio, reverse=True)

    total_value = 0
    # Inisialisasi kapasitas tersisa dari knapsack
    remaining_capacity = capacity

    # Iterasi melalui daftar item yang telah diurutkan
    for item in items:
        if remaining_capacity <= 0:
            break

        # Hitung fraksi item yang dapat dimasukkan ke dalam knapsack
        fraction = min(1, remaining_capacity / item.weight)

        # Perbarui nilai total dan kapasitas tersisa
        total_value += fraction * item.value
        remaining_capacity -= fraction * item.weight

    # Kembalikan nilai total maksimum yang diperoleh
    return round(total_value, 2)

# Contoh penggunaan
items = [Item(10, 60), Item(20, 100), Item(30, 120)]
capacity = 50
print("Maximum value we can obtain =", fractional_knapsack(items, capacity))

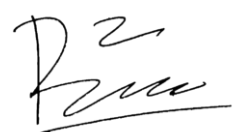
```

✓ 0.0s Python

Maximum value we can obtain = 240.0

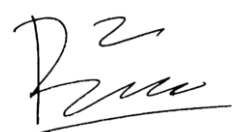
- 'class Item:' Mendefinisikan kelas 'Item' untuk merepresentasikan barang dengan berat dan nilai.
- 'def __init__(self, weight, value):' Konstruktor untuk menginisialisasi objek 'Item' dengan berat dan nilai.
- 'self.weight = weight' Menetapkan berat item.
- 'self.value = value' Menetapkan nilai item.
- 'self.ratio = value / weight' Menghitung dan menetapkan rasio nilai terhadap berat untuk setiap item.

Tanda Tangan



- `'def fractional_knapsack(items, capacity):'` Mendefinisikan fungsi `'fractional_knapsack'` yang menerima daftar item dan kapasitas knapsack.
- `'items.sort(key=lambda x: x.ratio, reverse=True)'` Mengurutkan item berdasarkan rasio nilai terhadap berat dalam urutan menurun.
- `'total_value = 0'` Menginisialisasi total nilai maksimum.
- `'remaining_capacity = capacity'` Menginisialisasi kapasitas tersisa dari knapsack.
- `'for item in items:'` Iterasi melalui daftar item yang telah diurutkan.
- `'if remaining_capacity <= 0: break'` Jika kapasitas knapsack sudah habis, keluar dari loop.
- `'fraction = min(1, remaining_capacity / item.weight)'` Menghitung fraksi item yang dapat dimasukkan ke dalam knapsack.
- `'total_value += fraction * item.value'` Menambahkan nilai fraksi item ke total nilai maksimum.
- `'remaining_capacity -= fraction * item.weight'` Mengurangi kapasitas tersisa dari knapsack.
- `'return round(total_value, 2)'` Mengembalikan total nilai maksimum yang diperoleh, dibulatkan ke dua desimal.
- `'items = [Item(10, 60), Item(20, 100), Item(30, 120)]'` Membuat daftar item dengan berat dan nilai masing-masing.
- `'capacity = 50'` Menetapkan kapasitas maksimum knapsack.
- `'print("Maximum value we can obtain =", fractional_knapsack(items, capacity))'` Memanggil fungsi `'fractional_knapsack'` dan mencetak nilai maksimum yang dapat diperoleh.

Tanda Tangan



IV. Latihan

1. Latihan 1

```
# 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.
'''
```

Python

2. Latihan 2

```
# Latihan 2

def minimize_bins(berat, c):
    berat.sort(reverse=True)
    bins = 0
    remaining_capacity = [c] * len(berat)

    for item in berat:
        for i in range(bins):
            if remaining_capacity[i] >= item:
                remaining_capacity[i] -= item
                break
        else:
            remaining_capacity[bins] -= item
            bins += 1

    return bins

# Example usage
berat = [4, 8, 1, 4, 2, 1]
c = 10
print("Minimum number of bins required:", minimize_bins(berat, c))

berat = [9, 8, 2, 2, 5, 4]
c = 10
print("Minimum number of bins required:", minimize_bins(berat, c))
```

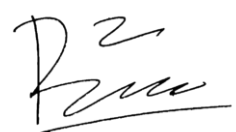
✓ 0.0s

Python

```
Minimum number of bins required: 2
Minimum number of bins required: 4
```

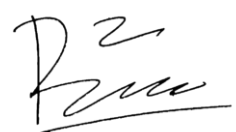
- ‘def minimize_bins(berat, c):’ Mendefinisikan fungsi ‘minimize_bins’ yang menerima dua parameter ‘berat’ (daftar berat item) dan ‘c’ (kapasitas maksimum setiap bin).

Tanda Tangan



- `'berat.sort(reverse=True)'` Mengurutkan daftar `'berat'` dalam urutan menurun.
- `'bins = 0'` Menginisialisasi jumlah bin yang digunakan.
- `'remaining_capacity = [c] * len(berat)'` Membuat daftar `'remaining_capacity'` untuk melacak kapasitas tersisa di setiap bin, diinisialisasi dengan kapasitas `'c'` untuk setiap bin potensial.
-
- `'for item in berat:'` Iterasi melalui setiap item dalam daftar `'berat'`.
- `'for i in range(bins):'` Iterasi melalui bin yang sudah digunakan.
- `'if remaining_capacity[i] >= item:'` Memeriksa apakah kapasitas bin ke-i cukup untuk menampung item.
- `'remaining_capacity[i] -= item'` Jika cukup, kurangi kapasitas tersisa bin ke-i dengan berat item.
- `'break'` Keluar dari loop jika item sudah dimasukkan ke dalam bin.
- `'else:'` Jika item tidak muat di bin yang sudah ada.
- `'remaining_capacity[bins] -= item'` Kurangi kapasitas bin baru dengan berat item.
- `'bins += 1'` Tambah jumlah bin yang digunakan.
- `'return bins'` Mengembalikan jumlah minimum bin yang digunakan.
- `'berat = [4, 8, 1, 4, 2, 1]'` Membuat daftar berat item pertama.
- `'c = 10'` Menetapkan kapasitas maksimum setiap bin.
- `'print("Minimum number of bins required:", minimize_bins(berat, c))'` Memanggil fungsi `'minimize_bins'` dan mencetak jumlah minimum bin yang dibutuhkan.
- `'berat = [9, 8, 2, 2, 5, 4]'` Membuat daftar berat item kedua.
- `'c = 10'` Menetapkan kapasitas maksimum setiap bin.
- `'print("Minimum number of bins required:", minimize_bins(berat, c))'` Memanggil fungsi `'minimize_bins'` dan mencetak jumlah minimum bin yang dibutuhkan.

Tanda Tangan



3. Latihan 3

```
# Latihan 3

def max_caught_thieves(arr, k):
    n = len(arr)
    police_positions = []
    caught_thieves = 0

    for i in range(n):
        if arr[i] == 'P':
            police_positions.append(i)
        elif arr[i] == 'T':
            for j in range(len(police_positions)):
                if abs(police_positions[j] - i) <= k:
                    caught_thieves += 1
                    del police_positions[j]
                    break

    return caught_thieves

# Example usage
arr = ['P', 'T', 'T', 'P', 'T']
k = 1
print("Maximum number of caught thieves:", max_caught_thieves(arr, k))

arr = ['T', 'T', 'P', 'P', 'T', 'P']
k = 2
print("Maximum number of caught thieves:", max_caught_thieves(arr, k))

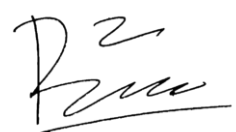
arr = ['P', 'T', 'P', 'T', 'T', 'P']
k = 3
print("Maximum number of caught thieves:", max_caught_thieves(arr, k))
```

✓ 0.0s Python

Maximum number of caught thieves: 2
Maximum number of caught thieves: 1
Maximum number of caught thieves: 2

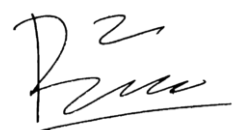
- ‘def max_caught_thieves(arr, k):’ Mendefinisikan fungsi ‘max_caught_thieves’ yang menerima dua parameter ‘arr’ (array yang merepresentasikan posisi polisi dan pencuri) dan ‘k’ (jarak maksimum yang dapat dilipat oleh polisi).
- ‘n = len(arr)’ Menghitung panjang array ‘arr’.
- ‘police_positions = []’ Membuat daftar untuk menyimpan posisi polisi.
- ‘caught_thieves = 0’ Inisialisasi jumlah pencuri yang tertangkap.
- ‘for i in range(n):’ Iterasi melalui setiap elemen dalam array ‘arr’.
- ‘if arr[i] == ‘P’:’ Jika elemen saat ini adalah polisi, tambahkan posisinya ke dalam daftar ‘police_positions’.

Tanda Tangan



- `'elif arr[i] == "T":'` Jika elemen saat ini adalah pencuri:
- `'for j in range(len(police_positions)):'` Iterasi melalui setiap posisi polisi.
- `'if abs(police_positions[j] - i) <= k:'` Jika jarak antara polisi dan pencuri kurang dari atau sama dengan `'k'`:
- `'caught_thieves += 1'` Tambahkan satu pencuri yang tertangkap.
- `'del police_positions[j]'` Hapus posisi polisi dari daftar.
- `'break'` Keluar dari loop pencarian posisi polisi.
- `'return caught_thieves'` Kembalikan jumlah pencuri yang tertangkap.
- `'arr = ["P", "T", "T", "P", "T"]'` Array yang merepresentasikan posisi polisi dan pencuri pertama.
- `'k = 1'` Jarak maksimum yang dapat dilipat oleh polisi.
- `'print("Maximum number of caught thieves:", max_caught_thieves(arr, k))'` Memanggil fungsi `'max_caught_thieves'` dan mencetak jumlah maksimum pencuri yang tertangkap.

Tanda Tangan



V. Kesimpulan

Dari praktikum di atas, dapat menyimpulkan beberapa hal penting.. Pertama, algoritma serakah telah terbukti efektif dalam menyelesaikan beberapa masalah optimasi, seperti penggantian suku cadang dan penempatan barang ke dalam kontainer dengan cepat. Namun, perlu dicatat bahwa pendekatan ini tidak selalu menghasilkan solusi optimal, karena algoritma serakah tidak mempertimbangkan konteks keseluruhan masalah, tetapi hanya solusi terbaik di setiap tahap.

Pemrograman dinamis, sebaliknya, memberikan pendekatan yang lebih sistematis dengan membagi masalah menjadi submasalah yang lebih kecil dan mempertahankan hasil perhitungan sebelumnya untuk menghindari redundansi. Meskipun terkadang lebih lambat dibandingkan algoritma serakah, pemrograman dinamis menjanjikan solusi optimal melalui pemodelan masalah yang tepat.

Penting untuk memilih struktur data yang sesuai, seperti daftar, larik, atau kelas, bergantung pada kebutuhan untuk merepresentasikan data dan hasil penghitungan dengan jelas. Terakhir, analisis kompleksitas kinerja dan waktu sangat penting untuk menilai efektivitas algoritme yang digunakan dan memastikan bahwa solusi yang dihasilkan tidak hanya efisien secara komputasi, namun juga akurat dan andal.

Tanda Tangan

