Nama:Nesa Alfiarianti

Nim :D0424504

Jenis-Jenis Algoritma

Algoritma adalah sekumpulan intruksi atau langkah-langkah yang terstruktur untuk menyelesaikan suatu masalah.Didunia pemograman dan ilmu komputer,algoritma digunakan untuk memecahkan berbagi masalah mulai dari yang sederhana hingga yang sangat konteks.Ada berbagai jenis algoritma,masing-masing memilki karakteristik dan penggunaan yang berbeda.berikut adalah beberapa jenis algoritma beserta contohnya:

Algoritma Recursive

Penjelasan:

Algoritma recursive adalah metode pemrograman dimana suatu fungsi memanggil dirinya sendiri menyelesaikan sub-masalah yang lebih kecil,biasanya digunakan untuk masalah yang dapat dipecah menjadi begian-bagian yang serupa,seperti dalam penghitungan faktorial misalnya nilai n yang berkurang hingga mencapai 0,dimana ia mengembalikan 1.Setiap pengembalian n dengan hasil faktorial dari n-1 hingga mencapai kasus dasar.

Contoh:

Fungsi faktorial (n):

Jika n sama dengan 0,fungsi mengembalikan 1 sebagai kasus dasar.

Jika tidak,fungsi memanggil dirinya sendiri dengan n-1 dan mengembalikan hasilnya dengan n.

Pemanggilan fungsi:

Saat ini memanggil faktorial(5),fungsi akan menghitung:

5*faktorial(4)

4*faktorial(3)

3*faktorial(2)

2*faktorial(1)

1*faktorial(0) (kasus dasar,mengembalikan 1)

Hasil akhirnya adalah 5*4*3*2*1=120.

```python
def faktorial(n):
    if n < 0:
        return "Input harus bilangan bulat non-negatif"
    elif n == 0:
        return 1  # Kasus dasar
    else:
        return n * faktorial(n - 1)  # Rekursi

# Menggunakan fungsi
angka = 5
hasil = faktorial(angka)
print(f"Faktorial dari {angka} adalah {hasil}")  # Output: Faktorial dari 5 adalah 120

angka_negatif = -3
print(faktorial(angka_negatif))  # Output: Input harus bilangan bulat non-negatif
```

Kelebihan:

Menyederhanakan kode,terutama untuk masalah yang dapat dibagi menjadi sub-masalah (seperti pohon,grafik dan pemrograman dinamis.)

Kekurangan:

Penggunaan memori yang lebih tinggi karena setiap pemanggilan fungsi membutuhkan ruang di stack.

Potensi untuk menyebabkan stack overflow jika kedalaman rekursi terlalu dalam.

Kesimpulan:

Algoritma recursive untuk fungsi faktorial adalah menghitung faktorial suatu bilangan dengan terus menerus memanggil dirinya sendiri dengan bilangan yang lebih kecil hingga mencapai kasus dasar(n=0 atau n=1)lalu mengalikan hasil setiap panggilan untuk mendapatkan hasil akhir.Selain,dari faktorial ada beberapa contoh lain dari algoritma recursive.

Algoritma Sorting

Penjelasan:

Algoritma sorting adalah metode untuk mengurutkan data dalam urutan tertentu,seperti menaik (ascending)atau menurun(descending).Bubble sort adalah algoritma sederhana yang berfungsi dengan membandingkan setiap pasangan elemen bersebelahan dalam daftar,jika elemen lebih besar dari dua,keduanya ditukar.proses ini diulang hingga tidak ada lagi pertukaran yang dilakukan,yang berarti daftar sudah terurut.algoritma ini memiliki kompleksitas waktu $0(n^2)$,sehingga kurang efisien untuk daftar besar.

Contoh:

Fungsi bubble_sort(arr):

Fungsi ini menerima satu argumen ,yaitu arr,yang merupakan daftar yang ingin diurutkan.

Variabel n:

Menghitung panjang daftar untuk menentukan berapa banyak iterasi yang diperlukan.

Loop pertama(for i in range(n)):

Mengiterasi melalui setiap elemen dalam daftar,ini akan diulang sebanyak panjang daftar.

Flag swapped:

Variabel ini digunakan untuk memeriksa apakah ada pertukaran yang terjadi didalam iterasi.Jika tidak ada pertukaran ,ini berarti daftar sudah terurut dan proses bisa dihentikan lebih awal.

Loop kedua(for j in range (0,n-i-1)):

Loop ini membandingkan elemen bersebelahan.perbandingan dilakukan hingga elemen yang sudah terurut.

Kondisi pertukaran(if arr[j] > arr[j+1]):

Jika elemen yang lebih besar ditemukan sebelum elemen yang lebih kecil,kedua elemen tersebut akan ditukar.

Penghentian dini:

Jika tidak ada pertukaran yang terjadi diseluruh iterasi,fungsi akan keluar dari loop lebih awal,meningkatkan efisiensi.

```
def bubble_sort(arr):
    n = len(arr)
    # Iterasi melalui semua elemen
dalam array
    for i in range(n):
        # Inisialisasi flag untuk
mendeteksi pertukaran
        swapped = False
        # Perbandingan elemen
bersebelahan
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Tukar jika elemen
tidak terurut
                arr[j], arr[j+1] =
arr[j+1], arr[j]
                swapped = True
        # Jika tidak ada pertukaran,
daftar sudah terurut
        if not swapped:
            break
    return arr

data = [64, 34, 25, 12, 22, 11, 90]
print(bubble_sort(data))
```

Kelebihan:

Sederhana:mudah dipahami dan diimplementasikan.

In-place:Tidak memerlukan ruang tambahan.

Stabil:Mempertahankan urutan elemen yang sama.

Berhenti dini:Dapat keluar lebih awal jika sudah terurut.

Kekurangan:

Inefisiensi:Kompleksitas 0(n^2) membuatnya lambat untuk dataset besar.

Performa buruk:tidak cocok untuk pengurutan data besar

banyak pertukaran:melakuakan banyak pertukaran yang tidak efisien

waktu eksekusi lama:pengecekan berulang memperlambat proses.

Kesimpulan

Bubble sort dapat menjadi pilihan yang baik untuk pengajaran dan situasi di mana daftar kecil perlu diurutkan.Namun,untuk aplikasi praktis dengan dataset yang lebih besar,algoritma sorting yang lebih efisien harus dipertimbangkan.

Algoritma Searching

Penjelasan:

Algoritma searching adalah metode yang digunakan untuk menemukan elemen tertentu dalam struktur data.algoritma ini sangat penting dalam pemrograman dan pengolahan data karena sering digunakan dalam berbagai aplikasi,mulai dari pencarian dalam berbasis data hingga pengolahan array dan daftar.

Contoh:

Fungsi linear_search(arr,target):

Mencari dua parameter:

Arr:daftar(array)tempat pencarian dilakukan.

Target:elemen yang ingin dicari dalam daftar.

For indekx,value in enumerate(arr):

Menggunakan anumerate untuk mendapatkan indeks dan nilai dari setiap elemen dalam daftar.Ini memungkinkan kita untuk melacak posisi elemen dalam daftar.

For value == target:

Memeriksa apakah elemen saat ini (value) sama dengan elemen yang dicari (target).

Jika sama,fungsi akan mengembalikan indekx,yang merupakan posisi elemen yang ditemukan

Return -1:

Jika pencarian selesai tanpa menemukan elemen yang dicari,fungsi akan mengembalikan -1 sebagai tanda bahwa elemen tidak akan dalam daftar.

- Daftar:[10,20,30,40,50]
- Target:30
- Proses:

  Iterasi 1:indekx = 0,value = 10 (tidak cocok)

  Iterasi 2:indekx = 1,value = 20 (tidak cocok)

  Iterasi 3:indekx = 2,value = 30 (cocok,mengembalikan 2)

```
def linear_search(arr, target):
    for index, value in
enumerate(arr):
        if value == target:
            return index
    return -1

# Contoh penggunaan
arr = [10, 20, 30, 40, 50]
target = 30
print(linear_search(arr, target))  #
Output: 2
```

Kelebihan:

Sederhana dan mudah dipahami.

Tidak memerlukan data terurut.

Kekurangan:

Tidak efisien untuk daftar besar,karena memeriksa setiap elemen satu per satu.

Kompleksitas waktu:0(n),dimana n adalah jumlah elemen dalam daftar.

Kesimpulan:

Pencarian liner adalah metode yang efektif untuk daftar kecil atau tidak berurut ,tetapi dapat menjadi lambat ketika jumlah data meningkat.

Algoritma Greedy

Penjelasan:

Agoritma greedy bekerja dengan memilih pilihan terbaik yang tersedia pada setiap angkah tanpa memikirkan konsekuensi jangka panjang,pendekatan ini mungkin tidak menghasilkan solusi optimal,tetapi sering digunakan karena cepat dan mudah diimplentasikan.

Contoh:

Input Data:

Dapatkan daftar koin dan jumlah yang diinginkan (amount).

Sort Koin:

Urutkan daftar koin dari yang terbesar ke yang terkecil. Ini untuk memastikan kita selalu mencoba menggunakan koin dengan nilai tertinggi terlebih dahulu.

Inisialisasi Variabel:

Buat variabel count untuk menghitung jumlah koin yang digunakan, dan inisialisasi dengan 0.

Iterasi Koin:

Untuk setiap koin dalam daftar yang telah diurutkan:

Selama jumlah yang tersisa (amount) lebih besar atau sama dengan nilai koin saat ini:

Kurangi amount dengan nilai koin.

Tambahkan 1 ke count untuk setiap koin yang digunakan.

Cek Sisa Jumlah:

Setelah iterasi, periksa apakah amount telah menjadi 0.

Jika ya, kembalikan nilai count (jumlah koin yang digunakan).

Jika tidak, kembalikan -1 (menandakan tidak mungkin mencapai jumlah yang diinginkan dengan koin yang tersedia)

Penerapan:

Input: Koin = [1, 5, 10, 25], Amount = 63.

Sort Koin: [25, 10, 5, 1].

Inisialisasi: count = 0.

Iterasi:

Gunakan 2 koin 25: amount = 63 - 50 = 13, count = 2.

Gunakan 1 koin 10: amount = 13 - 10 = 3, count = 3.

Gunakan 3 koin 1: amount = 3 - 3 = 0, count = 6.

5. Cek Sisa: amount = 0, kembalikan 6

Output:

Jumlah minimum koin yang dibutuhkan adalah 6.

```
def coin_change(coins, amount):
    coins.sort(reverse=True)  #
Urutkan koin dari yang terbesar
    count = 0
    for coin in coins:
        while amount >= coin:  # Ambil
koin sebanyak mungkin
            amount -= coin
            count += 1
    return count if amount == 0 else
-1  # Kembalikan jumlah koin atau -1
jika tidak bisa

# Contoh penggunaan
coins = [1, 5, 10, 25]  # Jenis koin
amount = 63  # Nilai yang diinginkan
print(coin_change(coins, amount))  #
Output: 6 (2x25 + 1x10 + 3x1)
```

Kelebihan:

Sederhana: Langkah-langkahnya mudah dipahami.

Efisien: Memiliki kompleksitas waktu.

Mudah Diimplementasikan: Kode singkat dan jelas.

Kekurangan:

idak Selalu Optimal: Solusi mungkin tidak terbaik untuk semua kombinasi koin.

Kasus Khusus: Beberapa jenis koin dapat menghasilkan lebih banyak koin daripada yang diperlukan.

Kesimpulan:

Algoritma Coin Change efektif untuk banyak kasus, tetapi tidak selalu memberikan solusi optimal. Untuk masalah lebih kompleks, algoritma lain mungkin diperlukan.

Algoritma Backtracking

Penjelasan:

Algoritman Backtracking adalah teknik algotitma yang digunakn untuk menyelesaikan masalah dengan mencari solusi secara betahap, tetapi mundur jika menemui jalan buntu (dead end). Algoritma ini

digunakan dalam masalah yang membutuhkan pencarian solusi optimal melalui percobaan dan kesalahan.

Contoh:

Inisialisasi:

Siapkan papan catur dan tentukan ukuran N.

Fungsi Backtrack:

Cek apakah penempatan ratu di posisi tertentu valid (tidak saling menyerang).

Jika valid, tempatkan ratu dan lanjutkan ke kolom berikutnya secara rekursif.

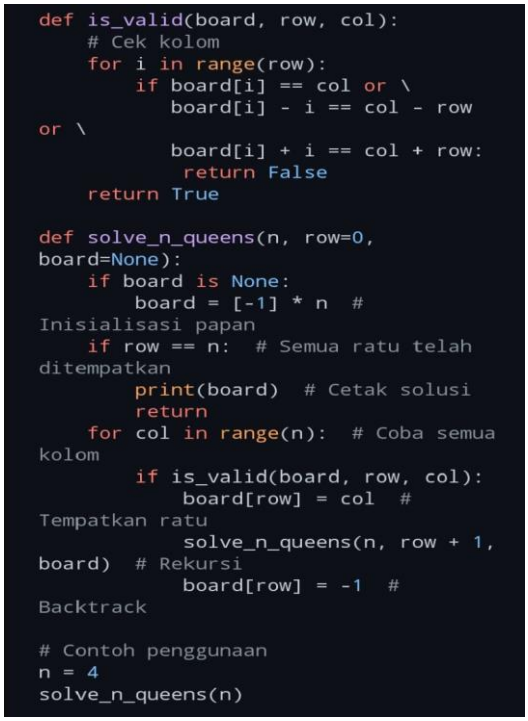
Jika semua ratu telah ditempatkan, simpan solusi.

Backtrack:

Jika penempatan tidak valid, hapus ratu dari posisi tersebut (backtrack) dan coba posisi lain.

Ulangi:

Lanjutkan proses sampai semua kemungkinan dicoba.



```
def is_valid(board, row, col):
    # Cek kolom
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row
or \
           board[i] + i == col + row:
            return False
    return True

def solve_n_queens(n, row=0,
board=None):
    if board is None:
        board = [-1] * n  #
Inisialisasi papan
    if row == n:  # Semua ratu telah
ditempatkan
        print(board)  # Cetak solusi
        return
    for col in range(n):  # Coba semua
kolom
        if is_valid(board, row, col):
            board[row] = col  #
Tempatkan ratu
            solve_n_queens(n, row + 1,
board)  # Rekursi
            board[row] = -1  #
Backtrack

# Contoh penggunaan
n = 4
solve_n_queens(n)
```

Kelebihan:

Fleksibel dan dapat digunakan untuk berbagai masalah pencarian dan mengoptimasikan.

Mampu menemukan semua solusi atau solusi terbaik.

Kekurangan:

Dapat menjadi tidak efisien untuk masalah besar karena eksplorasi semua kemungkinan.

Memerlukan pengelolaan memori yang baik untuk menjaga state dari pilihan yang telah dicaba.

Kesimpulan:

Algoritma Backtracking digunakan disini untuk mencari solusi dalam penempatan ratu.Dengan menguji setiap kemungkinan dan kembali saat solusi tidak valid, kita dapat menemukan semua konfigurasi yang memenuhi syarat.

Algoritama randomized

Penjelasan:

Algoritma Randomized adalah metode yang menggunakan elemen acak dalam prosesnya untuk mencapai hasil. Pendekatan ini sering digunakan untuk mempercepat algoritma, mengurangi kompleksitas, atau menemukan solusi dalam kasus-kasus yang sulit. Algoritma ini dapat dibagi menjadi dua kategori: algoritma yang menghasilkan solusi yang tepat dan algoritma yang menghasilkan solusi perkiraan.

Contoh:

Pilih Pivot: Pilih elemen acak dari array sebagai pivot.

Partisi: Bagi array menjadi dua bagian:

Elemen yang lebih kecil dari pivot.

Elemen yang lebih besar dari pivot.

Rekursi: Terapkan QuickSort secara rekursif pada kedua bagian.

Gabungkan: Gabungkan hasil dari kedua bagian.

Pivot: Elemen pivot dipilih secara acak dari array.

Partisi: Array dibagi menjadi dua bagian berdasarkan pivot.

Rekursi: QuickSort diterapkan pada masing-masing bagian.

Gabungan: Hasilnya digabungkan untuk membentuk array yang terurut.

```python
import random

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = random.choice(arr)  # Memilih pivot secara acak
    less_than_pivot = [x for x in arr if x < pivot]
    greater_than_pivot = [x for x in arr if x > pivot]
    return quicksort(less_than_pivot) + [pivot] + quicksort(greater_than_pivot)

# Contoh penggunaan
arr = [10, 7, 8, 9, 1, 5]
sorted_arr = quicksort(arr)
print(sorted_arr)  # Output: [1, 5, 7, 8, 9, 10]
```

kelebihan:

Kecepatan: Dalam banyak kasus, algoritma randomized seperti QuickSort dapat lebih cepat dibandingkan algoritma deterministik.

Sederhana: Algoritma seringkali lebih sederhana dan lebih mudah diimplementasikan.

Mengurangi Risiko Worst Case: Mengurangi kemungkinan terjebak dalam kasus terburuk dengan distribusi acak.

Kekurangan:

Ketidakpastian: Hasil dapat bervariasi pada setiap eksekusi, sehingga mungkin memerlukan lebih banyak pengujian.

Tidak selalu optimal: Dalam beberapa kasus, solusi yang diperoleh mungkin tidak optimal.

Kesimpulan:

Algoritma randomized, seperti QuickSort, memanfaatkan elemen acak untuk meningkatkan efisiensi dan kesederhanaan. Meskipun hasilnya mungkin tidak selalu konsisten, pendekatan ini sangat berguna dalam banyak konteks, terutama ketika menghadapi masalah besar dan kompleks.

Kesimpulan

Algoritma merupakan inti dari pemrograman dan komputasi modern.masing-masing jenis algoritma memiliki kekuatan dan kelemahan tergantung pada masalah yang dihadapi.dan memahami berbagi jenis algoritma seperti,Algoritma Recursive,Algoritma Sorting,Algoritma Searching,Algoritma Greedy,Algoritma Backtracking,Algoritma Randomized,kita dapat memilih algoritma yang paling sesuai untuk menyelesaikan masalah secara efektif dan efisien.