TUGAS MANDIRI

PERANCANGAN DAN ANALISIS ALGORITMA

”Optimal Binary Search Trees”

202323430048

DOSEN PENGAMPU:

Randi Proska Sandra, S.Pd, M.Sc

OLEH:

Manja Fani Oktavia

22343056

Informatika (NK)

PROGRAM STUDI INFORMATIKA

DEPARTEMEN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

A. PENJELASAN PROGRAM/ALGORITMA

Pohon pencarian biner adalah struktur data penting dalam ilmu komputer, terutama untuk mengimplementasikan kamus dengan operasi pencarian, penyisipan, dan penghapusan. Jika kita memiliki informasi tentang probabilitas pencarian elemen tertentu dalam himpunan, kita dapat mencari pohon pencarian biner yang optimal. Tujuan utamanya adalah meminimalkan jumlah rata-rata perbandingan dalam pencarian.

Secara sederhana, pohon pencarian biner optimal adalah pohon di mana setiap tingkat memiliki jumlah node yang sama atau mendekati sama, dan probabilitas pencarian di setiap node adalah proporsional dengan jumlah elemen di bawahnya. Dengan kata lain, node yang lebih sering dicari ditempatkan lebih dekat ke akar, dan probabilitas pencarian di setiap tingkat sebanding dengan jumlah node di bawahnya.

Untuk membuat Pohon pencarian biner , kita memulai dengan daftar kunci yang diurutkan dan probabilitasnya. Kemudian, kita membuat tabel yang berisi perkiraan biaya pencarian untuk semua kemungkinan subpohon dari daftar asli menggunakan pemrograman dinamis. Terakhir, kita menggunakan tabel ini untuk membuat Pohon pencarian biner.

Kompleksitas waktu pembuatan Pohon pencarian biner adalah O(n^3), yang dapat dioptimalkan menjadi O(n^2) dengan beberapa teknik. Setelah Pohon pencarian biner dibangun, kompleksitas waktu pencarian kunci adalah O(log n), sama dengan pohon pencarian biner biasa. Pohon pencarian biner digunakan dalam aplikasi di mana kunci memiliki probabilitas pencarian yang berbeda, seperti database, kompiler, dan program komputer lainnya.

Pohon pencarian biner optimal ini memungkinkan pencarian yang efisien dengan jumlah rata-rata perbandingan terkecil mungkin. Dengan demikian, dalam pencarian, algoritma dapat mengarahkan pencarian dengan cepat ke bagian pohon yang paling mungkin berisi elemen yang dicari, mengurangi jumlah perbandingan yang diperlukan untuk menemukan kunci yang dicari. Hal ini menghasilkan efisiensi yang signifikan dalam operasi pencarian, yang sangat penting dalam berbagai aplikasi, termasuk basis data, sistem kompilasi, dan pemrosesan teks.

B. PSEUDUCODE

```
Fungsi pohonPencarianBiner(F, P, n):
    Untuk setiap j dalam rentang(n):
        F[j][j] = P[j]
        Untuk setiap i dalam rentang(j - 1, -1, -1):
            F[i][j] = maksimum_penggerak
            Untuk setiap k dalam rentang(i, j + 1):
                biaya = jumlah(P[i : j + 1]) jika k == j else
F[i][k - 1] + F[k + 1][j] + jumlah(P[i : j + 1])
                F[i][j] = minimum(F[i][j], biaya)
    Kembalikan F[0][n - 1]

Fungsi Main():
    # inisialisasi data pemesanan
    baju = ["kaos", "kemeja", "dress", "batik"]
    probabilitas = [0.1, 0.2, 0.3, 0.4]
    n = panjang(baju)
```

```
    # membuat tabel F untuk menyimpan biaya minimal
    F = array 2D dengan ukuran n x n diisi dengan nilai 0

    # menghitung biaya minimal
    biayaMinimal = pohonPencarianBiner(F, probabilitas, n)

    # tampilkan biaya minimal
    Tampilkan "Biaya minimal untuk pemesanan baju adalah:",
biayaMinimal

Panggil Main()
```

C. SOURCE CODE

```
import sys

def pohonPencarianBiner(F, P, n):
    for j in range(n):
        F[j][j] = P[j]
        for i in range(j - 1, -1, -1):
            F[i][j] = sys.maxsize
            for k in range(i, j + 1):
                biaya = (
                    sum(P[i : j + 1])
                    if k == j
                    else F[i][k - 1] + F[k + 1][j] + sum(P[i :
j + 1]))

                F[i][j] = min(F[i][j], biaya)

    return F[0][n - 1]


def Main():
    # inisialisasi data pemesanan
    baju = ["kaos", "kemeja", "dress", "batik"]
    probabilitas = [0.1, 0.2, 0.3, 0.4]
    n = len(baju)

    # membuat tabel F untuk menyimpan biaya minimal
    F = [[0] * n for _ in range(n)]

    # menghitung biaya minimal
    biayaMinimal = pohonPencarianBiner(F, probabilitas, n)

    # tampilkan biaya minimal
    print("Biaya minimal untuk pemesanan baju adalah:",
biayaMinimal)

Main()
```

D. ANALISIS KEBUTUHAN WAKTU

1. Analisis Melalui Operasi/Instruksi

Pada algoritma di atas, terdapat beberapa operasi/instruksi yang dieksekusi secara berulang dalam loop. Kebutuhan waktu algoritma dapat dianalisis dengan memperhatikan operasi-instruksi yang dijalankan.

- Pohon pencarian binerOperator Penugasan (=): Terjadi dalam inisialisasi tabel 'F' dengan nilai awal 0, dan dalam pengisian nilai-nilai di dalam tabel tersebut.
- Pohon pencarian binerOperator Aritmatika (+): Digunakan untuk menghitung biaya dengan menjumlahkan elemen-elemen dalam array 'P'.
- Pohon pencarian binerPerulangan: Terdapat tiga perulangan nested (bersarang) dalam algoritma, yaitu untuk 'j', 'i', dan 'k'. Setiap perulangan memiliki batas tertentu yang bergantung pada ukuran input 'n'.
- Pohon pencarian binerFungsi Minimum (min()): Digunakan untuk mencari nilai minimum antara dua nilai.

2. Analisis Berdasarkan Jumlah Operasi Abstrak

   Kebutuhan waktu algoritma juga dapat dianalisis berdasarkan jumlah operasi abstrak yang dilakukan. Operasi-operasi ini tidak spesifik tergantung pada bahasa pemrograman tertentu, melainkan berdasarkan pada langkah-langkah algoritma itu sendiri.

   Misalnya, untuk algoritma di atas:
   - Jumlah operasi penugasan: O(n^3)
   - Jumlah operasi aritmatika: O(n^3)
   - Jumlah operasi perbandingan: O(n^3)

3. Analisis Menggunakan Pendekatan Best-Case, Worst-Case, dan Average-Case

   - Pohon pencarian binerBest-Case (Kasus Terbaik): Pada kasus terbaik, algoritma akan memiliki kompleksitas waktu yang lebih rendah. Dalam konteks ini, kasus terbaiknya adalah ketika algoritma tidak perlu melakukan pembagian array, yang berarti biaya minimal sudah terdapat pada diagonal matriks 'F'. Kompleksitas waktu dalam kasus terbaik adalah O(n^2).
   - Pohon pencarian binerWorst-Case (Kasus Terburuk): Kasus terburuk terjadi ketika algoritma harus mengevaluasi semua kemungkinan kombinasi untuk mencari biaya minimal. Dalam konteks ini, kompleksitas waktu dalam kasus terburuk adalah O(n^3), karena terdapat tiga perulangan nested.
   - Pohon pencarian binerAverage-Case (Kasus Rata-Rata): Analisis kasus rata-rata akan melibatkan analisis probabilitas distribusi input dan perkiraan rata-rata jumlah operasi yang dilakukan. Hal ini bisa menjadi lebih kompleks dan memerlukan analisis statistik yang mendalam, bergantung pada distribusi probabilitas yang digunakan dalam input.

Dalam kasus ini, akan tergantung pada distribusi probabilitas dari array 'P'.

Rumus:

- Pohon pencarian binerBest-Case: O(n^2)
- Pohon pencarian binerWorst-Case: O(n^3)
- Pohon pencarian binerAverage-Case: Bergantung pada distribusi probabilitas dari array 'P', namun umumnya diperkirakan sekitar O(n^3), karena dalam kasus rata-rata semua kemungkinan harus dievaluasi.

E. REFERENSI

GeeksforGeeks. (n.d.). Optimal Binary Search Tree | DP-24. Diakses pada 19 Maret 2024, dari https://www.geeksforgeeks.org/optimal-binary-search-tree-dp-24/

(*Introduction-to-the-Design-Analysis-of-Algorithms_compress*, n.d.)

F. LAMPIRAN LINK GITHUB

https://github.com/MANJA22343056/Perancangan-dan-Analisis-Algoritma