TUGAS EKSPLORASI MANDIRI

’Optimal Binary Search Trees’

202323430047

DOSEN PENGAMPU :

Randi Proska Sandra S.Pd., M.Sc.

OLEH :

Rawim Puja Aviola

22343010

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

DEPARTEMEN ELEKTRONIKA

UNIVERSITAS NEGERI PADANG

2024

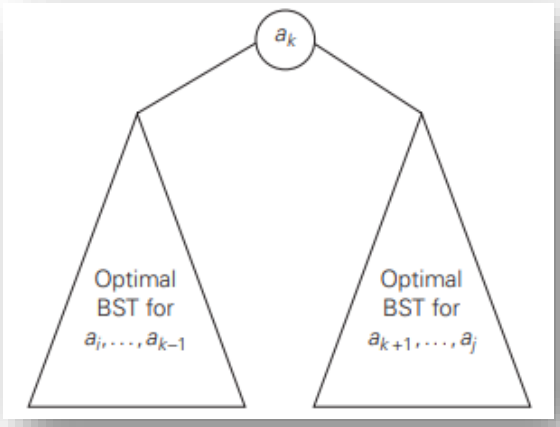
A. Penjelasan mengenai Optimal Binary Search Trees

Pohon pencarian biner adalah salah satu struktur data yang paling penting dalam ilmu komputer. Salah satu aplikasi utamanya adalah untuk mengimplementasikan kamus, sekumpulan elemen dengan operasi pencarian, penyisipan, dan penghapusan. Operasi ini dapat dijalankan dengan menggunakan prinsip pencarian biner karena datanya yang terurut.

Pohon pencarian biner (binary search tree) adalah pohon biner dengan aturan struktur data tertentu, yaitu:

1. Subpohon kiri mengandung simpul dengan kunci yang nilainya lebih kecil daripada nilai kunci akarnya.
2. Subpohon kanan mengandung simpul dengan kunci yang nilainya lebih besar daripada nilai kunci akarnya.
3. Baik subpohon kiri maupun subpohon kanan merupakan pohon biner.

Simpul dari pohon pencarian biner dapat berupa kunci pada data record atau data itu sendiri. Pohon biner dapat dibentuk menjadi binary search tree dengan mengikuti aturan yang disebutkan di atas.



Binary search trees dengan root Ak dan dua subpohon pencarian biner optimal

T_i^{k-1} and T_{k+1}^{j}.

Proses pencarian dimulai dari akar kemudian membandingkan elemen yang dicari dengan kunci pada akar. Jika elemen lebih kecil daripada nilai kunci akar, proses pencarian akan dilanjutkan pada subphon kiri. Sedagkan jika elemen lebih besar daripada nilai kunci akar, proses pencarian akan dilanjutkan pada subpohon kanan. Begitu seterusnya proses tersebut dilakukan hingga mencapai simpul atau daun yang nilainya sama dengan kunci yang sedang diproses.

Ada tiga jenis traversal pada *binary search tree* yaitu :

1. Pre-order Traversal
   Dilakukan dengan urutan proses data pada akar, lakukan traversal pada subpohon kiri secara keseluruhan, kemudian lakukan traversal pada subpohon kanan secara keseluruhan.

2. In-order Traversal
   Dilakukan dengan urutan proses melakukan traversal pada subpohon kiri secara kecara keseluruhan, proses data pada akar, kemudian lakukan traversal pada subpohon kanan secara keseluruhan.

3. Post-order Traversal
   Dilakukan dengan urutan proses melakukan traversal pada subpohon kiri secara keseluruhan, melakukan traversal pada subphon kanan secara keseluruhan, kemudian proses data pada akar.

EXAMPLE

Let us illustrate the algorithm by applying it to the four-key set we used at the beginning of this section:

Key	*:* A	B	C	D
Probability	*:0.1*	*0.2*	*0.4*	*0.3*

The initial tables look like this :

main table

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

root table

	0	1	2	3	4
1		1			
2			2		
3				3	
4					4
5					

Let us compute $C(1, 2)$*:*

$$C(1,2) = \min \begin{cases} k=1: & C(1,0) + C(2,2) + \sum_{s=1}^{2} p_s = 0 + 0.2 + 0.3 = 0.5 \\ k=2: & C(1,1) + C(3,2) + \sum_{s=1}^{2} p_s = 0.1 + 0 + 0.3 = 0.4 \end{cases}$$
$$= 0.4$$

Jadi, dari dua kemungkinan pohon biner yang berisi dua kunci pertama, *A* dan *B*, akar pohon optimal memiliki indeks 2 (yaitu mengandung *B*), dan jumlah rata-rata perbandingan dalam pencarian yang berhasil di pohon ini adalah 0,4.

B. Pseudocode Binary Search Trees

Sumber : Introduction to the Design & Analysis of Algorithms 3rd Edition karya **Anany Levitin**

```
//menemukan pohon pencarian biner yang optimal dengan pemograman dinamis

//masukkan : larik P[1..n] dari probabilitas pencarian untuk sebuah daftar terurut dari n kunci

//keluaran : jumlah rata-rata perbandingan dalam pencarian yang berhasil dalam

//   BTS optimal dan tabel R dari akar-akar subpohon dalam BTS optimal

for i ← 1 to n do

C[i,i − 1]← 0

C[i,i]← P[i]

R[i,i]← i

C[n + 1, n]← 0

for d ← 1 to n − 1 do //diagonal count

      for i ← 1 to n − d do

j ← i + d

minval ← ∞

for k ← i to j do

      if C[i, k − 1] + C[k + 1, j ] < minval

      minval ← C[i,k − 1] + C[k + 1, j ]; kmin ← k

      R[i,j] ← kmin

sum ← P[i]; for s ← i + 1 to j do sum ← sum + P[s]

C[i,j]← minval + sum

      return C[1,n], R
```

C. Program Binary Search Trees

```
#Program by 22343010_Rawim Puja Aviola

def optimal_binary_search_tree(P):
    n = len(P)


    # Inisialisasi tabel C, R, dan penambahan dummy entry di P
    C = [[0] * (n + 2) for _ in range(n + 2)]  # Perlu
diperpanjang dengan 1 elemen tambahan
    R = [[0] * n for _ in range(n)]
    for i in range(1, n + 1):
        C[i][i - 1] = 0
        C[i][i] = P[i - 1]
        R[i - 1][i - 1] = i
        C[i + 1][i] = 0


    # Perhitungan tabel C dan R
    for d in range(1, n):
        for i in range(1, n - d + 1):
            j = i + d
            minval = float('inf')
            for k in range(i, j + 1):
                if C[i][k - 1] + C[k + 1][j] < minval:
                    minval = C[i][k - 1] + C[k + 1][j]
                    kmin = k
            R[i - 1][j - 1] = kmin
            sum_P = sum(P[i - 1:j])
            C[i][j] = minval + sum_P


    # Mengembalikan jumlah perbandingan minimum dan tabel R
    return C[1][n], R
```

```
# Contoh penggunaan
P = [0.1, 0.2, 0.3, 0.1, 0.15, 0.15]
min_comparisons, R_table = optimal_binary_search_tree(P)
print("Jumlah perbandingan minimum:", min_comparisons)
print("Tabel R:")
for row in R_table:
    print(row)
```

D. Analisis Kebutuhan Waktu Algoritma

a) Analisis Source Code

Analisis menyeluruh berdasarkan operasi / instruksi

- Setiap iterasi dari loop for d akan memiliki kompleksitas waktu sebesar O(n).
- Di dalam setiap iterasi for d, terdapat loop for i yang melakukan iterasi sebanyak O(n - d) kali, dengan setiap iterasi berisi loop for k yang juga melakukan iterasi sebanyak O(d) kali.
- Dalam setiap iterasi for k, dilakukan beberapa operasi aritmatika sederhana (misalnya, penjumlahan, pengurangan, perbandingan), yang mungkin berkontribusi pada kompleksitas waktu tetapi memiliki dampak yang lebih kecil dibandingkan dengan loop luar.

  Oleh karena itu, secara keseluruhan, kompleksitas waktu algoritma ini adalah O(n^3).

Analisis berdasarkan jumlah operasi abstrak / operasi khas

- Dalam algoritma ini, jumlah operasi utama terjadi di dalam loop for d, di mana setiap iterasi memiliki kompleksitas O(n^2), dan di dalamnya ada beberapa operasi aritmatika sederhana.
- Jumlah operasi abstrak tergantung pada ukuran masukan n, dengan kompleksitas waktu yang diestimasi sebesar O(n^3).

Analisis menggunakan pendekatan best-case, worst-case, dan average-case

- Dalam konteks algoritma ini, best-case, worst-case, dan average-case akan memiliki kompleksitas yang sama, yaitu O(n^3), karena algoritma ini tidak bergantung pada input yang spesifik atau distribusi probabilitas tertentu dari P[*i*].
- Pada kasus terbaik, worst-case, dan average-case, algoritma ini akan memiliki kompleksitas waktu yang sama karena loop dan operasi dalam algoritma tidak bergantung pada sifat spesifik dari input.

  Oleh karena itu, kompleksitas waktu adalah O(n^3) untuk semua kasus tersebut.

b) Analisis Program

Analisis menyeluruh berdasarkan operasi / instruksi

- Dalam setiap iterasi dari loop for d, terdapat loop for i yang melakukan iterasi sebanyak n - d kali. Di dalam setiap iterasi for i, terdapat loop for k yang juga melakukan iterasi sebanyak d kali.
- Dalam setiap iterasi for k, dilakukan beberapa operasi aritmatika sederhana seperti penjumlahan, pengurangan, dan perbandingan.

Oleh karena itu, kompleksitas waktu dari setiap iterasi dalam loop for d adalah sekitar O(d^2), dengan total O(n^3) untuk seluruh algoritma.

Analisis berdasarkan jumlah operasi abstrak / operasi khas

- Dalam algoritma ini, setiap iterasi dari loop for d akan melakukan sekitar O(n^2) operasi aritmatika sederhana.
- Karena ada loop for d yang berjalan sebanyak n - 1 kali, total jumlah operasi abstrak adalah sekitar O(n^3).

Analisis menggunakan pendekatan best-case, worst-case, dan average-case

Dalam konteks algoritma ini, baik kasus terbaik, kasus terburuk, maupun kasus rata-rata semuanya memiliki kompleksitas waktu yang sama, yaitu O(n^3). Ini karena algoritma ini memiliki pola kerja yang serupa pada semua kasus input, di mana iterasi terdalam dari loop memiliki kompleksitas waktu yang sama untuk setiap iterasi.

E. Referensi

Levitin, A. (2011). *Introduction to the Design* . Anany Levitin. — 3rd ed.
Widodo, N. U. (2018/2019). Aplikasi Binary Search Tree Sebagai Metode. 2-3.

Link Github : https://github.com/rawmpujaa/rawmpujaa