DESAIN DAN ANALISIS ALGORITMA

ALGORITMA BRUTE FORCE

Dosen Pengampu :

Dr. Dra. Luh Gede Astuti,M.Kom.

Disusun Oleh :

Aditya Chandra Nugraha 2308561092

PROGRAM STUDI INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS UDAYANA

2024

1. **Matrix Multiplication**

1.1 Pseudocode

```
procedure MatrixMultiplication(A, B)
  input A, B n*n matrix
  output C, n*n matrix

begin
  for ( i = 0; i < n; i++)
    for ( j = 0; j < n; j++)
      C[i,j] = 0;
    end for
  end for

  for ( i = 0; i < n; i++)
    for ( j = 0; j < n; j++)
      for( k = 0; k < n; k++)
       C[i,j] = C[i,j] + A[i,k] * B[k,j]
      end for
    end for
  end for
end MatrixMultiplication
```

1.2 Implementasi Algoritma (Python)

Contoh soal untuk digunakan:

Matriks A = [[1,2,3],
[4,5,6],
[7,8,9]]

Matriks B = [[11,12,13],
[14,15,16],
[1,2,3]]

```python
def perkalianMatriks(inputA,inputB):
  n = len(inputA[0])
  m = len(inputB)
  mC = [[0 for _ in range(n)] for _ in range(m)]

  # iterasi untuk kalkulasi matriks
  for i in range(m):
    for j in range(n):
```

```
        for k in range(m):
            mC[i][j] += inputA[i][k] * inputB[k][j]

  return mC



matriks1 = [[1,2,3],
        [4,5,6],
        [7,8,9]]
matriks2 = [[11,12,13],
        [14,15,16],
        [1,2,3]]
a = perkalianMatriks(matriks1,matriks2)

for x in a:
  print(x)
```

output

```
[42, 48, 54]
[120, 135, 150]
[198, 222, 246]
```

1.3 Tracing

Indeks i	Indeks j	Indeks k	perhitungan inputP[j] == Teks[i + j]
0	0	0	mC[0][0] += inputA[0][0] * inputB[0][0] mC[0][0] = 1*11 = 11
0	0	1	mC[0][0] += inputA[0][1] * inputB[1][0] mC[0][0] = 11 + (2*14) = 11+28 = 39
0	0	2	mC[0][0] += inputA[0][2] * inputB[2][0] mC[0][0] = 39 + (3*1) = 39+3 = 42
0	1	0	mC[0][1] += inputA[0][0] * inputB[0][1] mC[0][1] = 1*12 = 12
0	1	1	mC[0][1] += inputA[0][1] * inputB[1][1]

			mC[0][1]= 12 + (2*15)=12+30=42
0	**1**	**2**	**mC[0][1] += inputA[0][2] * inputB[2][1]** **mC[0][1] = 42 +(3*2) = 48**
0	**2**	**0**	**mC[0][2] += inputA[0][0] * inputB[0][2]** **mC[0][2] = 1*13 = 13**
0	**2**	**1**	**mC[0][2] += inputA[0][1] * inputB[0][2]** **mC[0][2] =13 + (2*16)=13+32 = 45**
0	**2**	**2**	**mC[0][2] += inputA[0][2] * inputB[0][2]** **mC[0][2] =45 + (3*3)=45+9 = 54**
…			

1.4 Analisis kompleksitas algoritma

Algoritma perkalian Matriks menggunakan nested loop untuk mengkalkulasi tiap elemen dari matriks mC,setiap loop ber iterasi sebanyak n kali sesuai dengan ukuran matriks. Dikarenakan terdapat 3 loop dalam algoritma maka jumlah total iterasi adalah n x n x n = n^3 .Maka kompleksitas Algoritma adalah $O(n^3)$

2. **Knapsack problem**

2.1

```
int  knapSack(int  W, int  wt[], int  val[], int  n)
{
   // Base Case
   if (n == 0 || W == 0)
       return 0;

   // If weight of the nth item is more than Knapsack capacity W, then
   // this item cannot be included in the optimal solution
   if (wt[n-1] > W)
       return knapSack(W, wt, val, n-1);

   // Return the maximum of two cases:
   // (1) nth item included
   // (2) not included
   else return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                    knapSack(W, wt, val, n-1) );

}
```

2.2 Implementasi Algoritma (Python)

```
def knapsackProblem(kapasitas,wt,val,n):
  if n == 0 or kapasitas == 0:
     return 0

  if (wt[n-1] > kapasitas):
     return knapsackProblem(kapasitas, wt, val, n-1)
  else :
     return max(val[n-1] + knapsackProblem(kapasitas-wt[n-1], wt, val, n-1),knapsackProblem(kapasitas,
wt, val, n-1))



profit = [60, 100, 120]
weight = [10, 20, 30]
W = 50
n = len(profit)
print("total profit adalah : ",knapsackProblem(W, weight, profit, n))
```

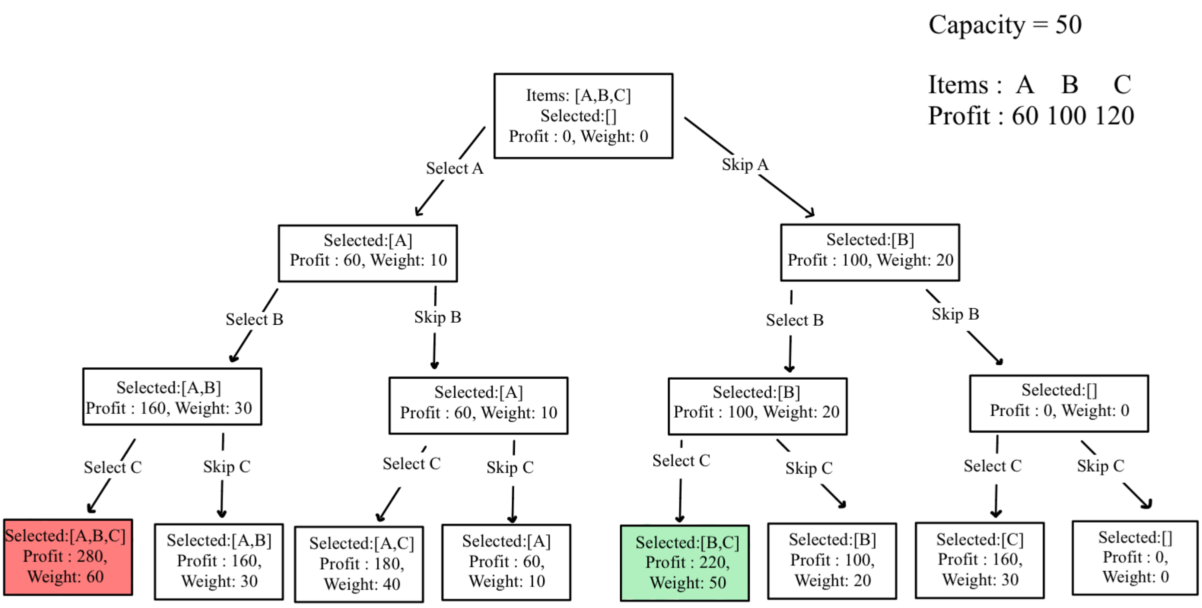
output

```
total profit adalah :  220
```

penjelasan:

hasil didapat dari 100 + 120 dengan weight yang memiliki weight masing-masing 20 dan 30. jika kedua weight dijumlah maka totalnya tidak melebihi kapasitas tas (W)

2.3 Tracing



2.4 Analisis kompleksitas algoritma

Karena pada setiap pemanggilan rekursif, algoritma memecah masalah menjadi dua sub-masalah — satu dengan menyertakan item ke-n dan satu lagi tanpa menyertakannya. Ini berarti bahwa setiap pemanggilan rekursif bercabang menjadi dua pemanggilan rekursif baru.Jadi ada 2^N kombinasi dari pilihan yang mungkin untuk setiap item (dimasukkan atau tidak), maka kompleksitas waktu dari algoritma ini adalah $O(2^N)$. Ini berarti bahwa waktu eksekusi algoritma akan meningkat secara eksponensial ketika jumlah item bertambah, sehingga algoritma ini menjadi tidak efisien untuk nilai n yang besar