Program Studi Informatika

Mata Kuliah Desain dan Analisis Algoritma

TUGAS 02 : 20 Desember 2021

Nama : Hasna Muna Munifah

NIM : M0520034

1. METODE DIVIDE AND CONQUER

Tugas anda adalah mengimplementasikan strategi Divide and Conquer dalam mendesain algoritma untuk memecahkan suatu masalah, dengan ketentuan :

a. Pilih salah satu permasalahan yang akan diselesaikan
b. Boleh menggunakan bahasa Pemrograman apapun, tetapi tidak boleh pakai tools yang sudah ada. Jadi ini murni hasil coding anda.
c. Laporan mencakup : deskripsi masalah, ide penyelesaian masalah, source code dan hasil running programnya.
d. Kesimpulan tentang algoritma yang dipilih.

2. ROUTE PLANNING PROBLEM

Tugas anda adalah menyelesaikan persoalan Route Planning, dengan ketentuan :

a. Graph yang merepresentasikan peta perjalanan silakan ditentukan sendiri, dengan jumlah vertex / node tidak kurang dari 15 titik.
b. Algortima yang diimplmentasikan Dijktra Algorithm.
c. Boleh menggunakan bahasa pemrograman apapun.
d. Laporan mencakup : deskripsi masalah, ide penyelesaian masalah, source code dan hasil running programnya.

JAWABAN

1. METODE DIVIDE AND CONQUER

Metode Divide And Conquer merupakan suatu paradigma yang digunakan dengan cara membagi suatu masalah menjadi masalah-masalah yang lebih kecil (subproblem) dan menyelesaikan sub-masalah tersebut terlebih dahulu. Dalam hal ini saya menerapkan metode divide and conquer kedalam masalah untuk mencari nilai terbesar dari suatu kumpulan integer yang ada di dalam suatu array.

Deskripsi masalah:

Seorang game developer ingin membuat system leaderboard untuk suatu game berserver local berisi 20 orang. Developer tersebut hanya ingin menampilkan 1 score tertinggi (Highest Score) dan 1 score terendah (Lowest Score).

Ide Penyelesaian Masalah:

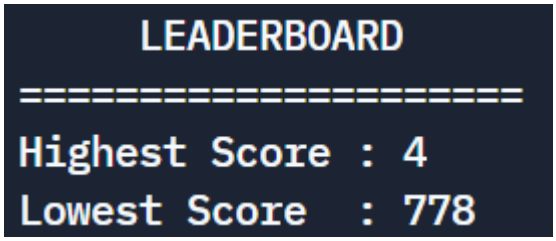
Membagi 20 input integer menjadi masing-masing 2 buah input kemudian membandingkan kedua input tersebut untuk mendapatkan nilai terkecil dari 2 input tersebut. Kemudian mengulangi cara yang sama sampai semua angka telah dibandingkan menggunakan fungsi rekursi sederhana.

Source code:

```
#include <stdio.h>
int DnC_Max(int a[], int index, int l);
int DnC_Min(int a[], int index, int l);

// function untuk mencari nilai maksimum dari suatu array
int DnC_Max(int a[], int index, int l)
{
    int max;
    // Jika sisa nilai index dan l
    if (index >= l - 2) {
            if (a[index] > a[index + 1])
                    return a[index];
            else
                    return a[index + 1];
    }

    // isi nilai max dengan nilai maximal dari suatu array secara rekursif menggunakan index
    max = DnC_Max(a, index + 1, l);

    if (a[index] > max)
            return a[index];
    else
```

```
		return max;
}

// function untuk mencari nilai minimum dari suatu array
int DnC_Min(int a[], int index, int l)
{
	int min;
	// Jika sisa nilai index dan l
	if (index >= l - 2) {
		if (a[index] < a[index + 1])
			return a[index];
		else
			return a[index + 1];
	}

	// isi nilai min dengan nilai minimum dari suatu array secara rekursif menggunakan index
	min = DnC_Min(a, index + 1, l);

	if (a[index] < min)
		return a[index];
	else
		return min;
}

// Main Function
int main(){
	int min, max, N;
	int a[20] = { 40, 30, 20, 11, 54, 129, 440, 95, 7, 32, 65, 98, 778, 324, 4, 44, 109, 105, 154,
132 };

	// masuk ke function DnC_Max untuk dijalankan secara rekursif
	max = DnC_Max(a, 0, 20);

	// masuk ke function DnC_Min untuk dijalankan secara rekursif
	min = DnC_Min(a, 0, 20);
	printf("     LEADERBOARD\n");
	printf("====================\n");
	printf("Highest Score : %d\n", min);
	printf("Lowest Score  : %d\n", max);
	return 0;
}
```

Hasil Running Program:



2. ROUTE PLANNING PROBLEM

Deskripsi Masalah:

Dalam graph yang menggambarkan jarak antar lokasi (kota) dengan bobot-bobot jarak tertentu, kita dapat mecari jarak terpendek yang dapat kita tempuh. Untuk mencari jarak terpendek, kita dapat menggunakan salah satu algoritma yaitu djikstra algorithm.

Ide Penyelesaian Masalah:

Algoritma Dijkstra bekerja dengan membuat jalur ke satu simpul optimal pada setiap langkah. Jadi pada langkah ke n, setidaknya ada n node yang sudah kita tahu jalur terpendek. Langkah-langkah algoritma Dijkstra dapat dilakukan dengan langkah-langkah berikut:

a. Tentukan titik mana yang akan menjadi node awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap.
b. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi) 2.
c. Set semua node yang belum dilalui dan set node awal sebagai “Node keberangkatan”
d. Dari node keberangkatan, pertimbangkan node tetangga yang belum dilalui dan hitung jaraknya dari titik keberangkatan. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru
e. Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah dilalui sebagai “Node dilewati”. Node yang dilewati tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
f. Set “Node belum dilewati” dengan jarak terkecil (dari node keberangkatan) sebagai “Node Keberangkatan” selanjutnya dan ulangi langkah e.

Source Code:

```
#include <limits.h> //Untuk menggunakan INT_MAX, INT_MIN
#include <stdio.h>
#include <stdbool.h> //Untuk menggunakan boolean
```

```
// Jumlah Vertex/Node dalam graph
#define V 20

// Function yang digunakan untuk mencari vertex/node dengan jarak minimum dari kumpulan node/vertex yang belum termasuk kedalam path terpendek
int minDistance(int dist[], bool sptSet[])
{
    // Inisialisasi nilai minimum
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// Function yang digunakan untuk mengeprint jarak terpendek dari masing-masing vertex
int printSolution(int dist[], int n)
{
    printf("Vertex\t| Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t|\t %d\n", i, dist[i]);
}

// Function untuk mencari jarak terpendek dengan algoritma djikstra
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // Array output yang akan menyimpan jarak terpendek dari vertex ke i

    bool sptSet[V]; // sptSet[i] berisi true jika vertex i telah mencapai shortest path nya dari source (0)

    // Menginisialisasi semua jarak menjadi infinity dan shortest path menjadi false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Jarak vertex source ke dirinya sendiri selalu 0
    dist[src] = 0;

    // Mencari jarak terpendek untuk semua vertex
    for (int count = 0; count < V - 1; count++) {
        // u diisi dengan jarak minimum suatu vertex dengan vertex lain yang belum diproses
```

```
            int u = minDistance(dist, sptSet);

            // Menandai vertex menjadi sudah di cek (true)
            sptSet[u] = true;

            // Update jarak pada matrix dari vertex yang sedang dipilih.
            for (int v = 0; v < V; v++)

        //Update jarak[v] jika belum di cek, ada edge dari u ke v, dan total nilai dari source ke v
melalui u lebih kecil dari nilai dist[v] yang sekarang
                    if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                            && dist[u] + graph[u][v] < dist[v])
                            dist[v] = dist[u] + graph[u][v];
    }

    // Print array yang ada
    printSolution(dist, V);
}

// Main Function
int main()
{
   // Matrix 20 x 20 representasi graph
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0, 0, 0, 5, 3, 2, 0 ,0, 7, 0, 0, 0 },
                                            { 4, 0, 8, 0, 0, 0, 0, 11, 0, 0, 0, 4, 10, 0, 0, 6, 0, 3, 0,
1 },
                                            { 0, 8, 0, 7, 0, 4, 0, 0, 2, 1, 1, 0, 4, 7, 0, 0, 0, 0, 0, 0 },
                                            { 0, 0, 7, 0, 9, 14, 0, 0, 0, 9, 9, 3, 0, 0, 0, 5, 0, 0, 1, 0 },
                                            { 0, 0, 0, 9, 0, 10, 0, 0, 0, 1, 1, 1, 9, 6, 0, 0, 0, 4, 0, 0 },
                                            { 0, 0, 4, 14, 10, 0, 2, 0, 0, 2, 5, 3, 0, 1, 10, 0, 4, 0, 0,
1 },
                                            { 0, 0, 0, 0, 0, 2, 0, 1, 6, 0, 0, 4, 0, 3, 0, 1, 0, 1, 0, 0 },
                                            { 8, 11, 0, 0, 0, 0, 1, 0, 7, 3, 0, 0, 6, 0, 0, 8, 0, 1, 0, 0 },
                                            { 0, 0, 2, 0, 0, 0, 6, 7, 0, 0, 0, 0, 0, 1, 2, 0, 0, 1, 0, 0 },
                                            { 0, 0, 1, 9, 1, 2, 0, 3, 0, 9, 0, 10, 14, 8, 0, 0, 3, 0, 2, 2},
                                            { 0, 0, 1, 9, 1, 5, 0, 0, 0, 0, 1, 5, 5, 2, 0, 0, 0, 0, 7, 8},
                                            { 5, 4, 0, 3, 1, 3, 4, 0, 0, 10, 5, 0, 0, 3, 0, 1, 0, 0, 4, 8},
                                            { 3, 10, 4, 0, 9, 0, 0, 6, 0, 14, 5, 0, 1, 4, 0, 0, 3, 7, 0},
                                            { 2, 0, 7, 0, 6, 1, 3, 0, 1, 8, 2, 3, 1, 0, 1, 0, 0, 8, 3},
                                            { 0, 0, 0, 0, 0, 10, 0, 0, 2, 0, 0, 0, 4, 0, 5, 3, 2, 0, 0},
                                            { 0, 6, 0, 5, 0, 0, 1, 8, 0, 0, 0, 1, 0, 1, 5, 0, 9, 5, 0, 1},
                                            { 7, 0, 0, 0, 0, 4, 0, 0, 0, 3, 0, 0, 0, 0, 3, 9, 0, 0, 10, 0},
                                            { 0, 3, 0, 0, 4, 0, 1, 1, 1, 0, 0, 0, 3, 0, 2, 5, 0, 6, 0, 5},
                                            { 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 7, 4, 7, 8, 0, 0, 10, 0, 0, 1},
                                            { 0, 1, 0, 0, 0, 1, 0, 0, 0, 2, 8, 8, 0, 3, 0, 1, 0, 5, 1, 0} };
```

```
    dijkstra(graph, 0);

    return 0;
}
```

Hasil Running Program:

```
Vertex  | Distance from Source
0   |   0
1   |   4
2   |   5
3   |   6
4   |   5
5   |   3
6   |   5
7   |   5
8   |   3
9   |   5
10  |   4
11  |   5
12  |   3
13  |   2
14  |   3
15  |   5
16  |   5
17  |   4
18  |   5
19  |   4
```