














DEMO PRAKTIKUM 4

Nama : Wildan Fauzy M. H.

NRP : 5025221044

AC Praktikum : 1 (ERSY4)

AC Revisi : 3 (_R, DAYAT9, PPA)

_R	CPP	CORRECT	
_R	CPP	WRONG-ANSWER	
DAYAT9	CPP	CORRECT	
DAYAT9	CPP	WRONG-ANSWER	
DAYAT9	CPP	WRONG-ANSWER	
_R	CPP	WRONG-ANSWER	
_R	CPP	WRONG-ANSWER	
_R	CPP	WRONG-ANSWER	
_R	CPP	WRONG-ANSWER	
PPA	CPP	CORRECT	
PPA	CPP	WRONG-ANSWER	
PPA	CPP	WRONG-ANSWER	
ERSY4	CPP	CORRECT	

Penjelasan Soal Dayat9

Soal dari dayat dapat diselesaikan dengan menggunakan algoritma BFS yang disesuaikan dengan kebutuhan soal. Berikut merupakan function yang dapat dipakai :

```
const int MAX = 355;
const int dx[] = {0, 1, 0, -1};
const int dy[] = {1, 0, -1, 0};

char grid[MAX][MAX];
bool visited[MAX][MAX];
int rows, cols, queries;

int bfs(int x, int y) //Buat menghitung jumlah pieces yang terhubung dengan grid
{
    // Parameter x & y sebagai titik mula BFS
    queue<pair<int, int>> q;
    q.push({x, y});
    visited[x][y] = true;

    int pieces = 0;

    while (!q.empty())
    {
        x = q.front().first;
        y = q.front().second;
        q.pop();

        for (int i = 0; i < 4; i++)
        {
            int nx = x + dx[i];
            int ny = y + dy[i];

            if (nx < 0 || nx >= rows || ny < 0 || ny >= cols)
                continue;

            if (grid[nx][ny] == '#')
                pieces++;
            else if (!visited[nx][ny])
            {
                q.push({nx, ny});
                visited[nx][ny] = true;
            }
        }
    }

    return pieces;
}
```

Adapun fungsi utamanya dalah sebagai berikut :

```
int main()
{
    cin >> rows >> cols >> queries;

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            cin >> grid[i][j];
        }
    }

    while (queries--)
    {
        for (int i = 0; i < MAX; i++)
        {
            for (int j = 0; j < MAX; j++)
            {
                visited[i][j] = false;
            }
        }

        int x, y;
        cin >> x >> y;
        cout << bfs(x - 1, y - 1) << "\n";
    }

    return 0;
}
```

Fungsi utama tersebut pada dasarnya memiliki algoritma sebagai berikut :

- Program membaca input dari user untuk jumlah baris (rows), jumlah kolom (cols), dan jumlah query (queries).
- Program kemudian membaca input untuk setiap karakter grid
- Untuk setiap query, program akan melakukan hal-hal berikut :
 - Setel semua elemen visited menjadi false.
 - Baca input untuk koordinat x dan y.
 - Panggil fungsi bfs dengan koordinat (x - 1, y - 1) dan cetak hasilnya.
- Program akan berakhir jika semua query selesai.

Penjelasan Soal PPA (Perang Parit Air)

Soal PPA dapat diselesaikan dengan menggunakan algoritma BFS untuk menjelajahi dan menghitung luas area tergenang sesuai dengan keterangan soal. Adapun function yang dipakai adalah sebagai berikut :

```
struct Koordinat // Nyimpen koordinat X dan Y dalam Matriks
{
    int x;
    int y;
};

int DeteksiAreaTergenang // Mendeteksi Luas area tergenang pada matriks ketinggian suatu area
(vector<vector<int>>& heights, int startX, int startY)
{
    int numRows = heights.size();
    int numCols = heights[0].size();
    int AreaGenangan = 0;

    vector<vector<bool>> visited(numRows, vector<bool>(numCols, false));
    queue<Koordinat> q;

    q.push({startX, startY});
    visited[startX][startY] = true;

    while (!q.empty())
    {
        Koordinat curr = q.front();
        q.pop();

        int currX = curr.x;
        int currY = curr.y;

        AreaGenangan++;

        if (currX - 1 >= 0 && !visited[currX - 1][currY] && heights[currX - 1][currY] <= heights[currX][currY])
        {
            q.push({currX - 1, currY});
            visited[currX - 1][currY] = true;
        }

        if (currX + 1 < numRows && !visited[currX + 1][currY] && heights[currX + 1][currY] <= heights[currX][currY])
        {
            q.push({currX + 1, currY});
            visited[currX + 1][currY] = true;
        }

        if (currY - 1 >= 0 && !visited[currX][currY - 1] && heights[currX][currY - 1] <= heights[currX][currY])
        {
            q.push({currX, currY - 1});
            visited[currX][currY - 1] = true;
        }

        if (currY + 1 < numCols && !visited[currX][currY + 1] && heights[currX][currY + 1] <= heights[currX][currY])
        {
            q.push({currX, currY + 1});
            visited[currX][currY + 1] = true;
        }
    }

    return AreaGenangan;
}
```

Adapun fungsi utama dalam program ini adalah sebagai berikut :

- Program membaca input user yang berupa jumlah baris (N), jumlah kolom (M), koordinat awal startX dan startY.
- Program kemudian membuat matriks ketinggian heights dengan ukuran N x M
- Program kemudian membaca input ketinggian setiap elemen
- Program kemudian memanggil fungsi DeteksiAreaTergenang dengan koordinat awal (startX - 1, startY - 1) dan mencetak hasilnya.
- Setelah mencetak hasil, Program akan berakhir.

Penjelasan Soal _R

Soal dapat diselesaikan dengan algoritma BFS untuk mencari Shortest Path antara dua titik. Adapun function yang digunakan dalam program adalah sebagai berikut :

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

const int MAX_SIZE = 20;

struct Position //Utk posisi
{
    int x, y;
};

//Untuk cek apakah x,y yang diperiksa masih di dalam batas peta
bool isValidPosition(int x, int y, int maxX, int maxY)
{
    return (x >= 0 && x < maxX && y >= 0 && y < maxY);
}

vector<Position> findShortestPath(vector<vector<char>>& map, Position start, Position end)
{
    int maxX = map[0].size();
    int maxY = map.size();

    vector<vector<bool>> visited(maxY, vector<bool>(maxX, false));
    vector<vector<Position>> parent(maxY, vector<Position>(maxX, { -1, -1 }));

    queue<Position> q;
    q.push(start);
    visited[start.y][start.x] = true;
```

```

while (!q.empty())
{
    Position curr = q.front();
    q.pop();

    int currX = curr.x;
    int currY = curr.y;

    // Cek sudah di titik akhir
    if (currX == end.x && currY == end.y)
    {
        // Membangun jalur dari titik akhir ke titik awal menggunakan parent
        vector<Position> path;
        Position pos = end;
        while (pos.x != -1 && pos.y != -1)
        {
            path.push_back(pos);
            pos = parent[pos.y][pos.x];
        }
        return path;
    }

    //Eksplorasi tetangga-tetangga yang belum dikunjungi
    vector<Position> neighbors =
    {
        {currX - 1, currY},    // kiri
        {currX + 1, currY},    // kanan
        {currX, currY - 1},    // atas
        {currX, currY + 1}     // bawah
    };

    for (const auto& neighbor : neighbors)
    {
        int neighborX = neighbor.x;
        int neighborY = neighbor.y;

        if (isValidPosition(neighborX, neighborY, maxX, maxY) &&
            map[neighborY][neighborX] != '#' && !visited[neighborY][neighborX])
        {
            q.push(neighbor);
            visited[neighborY][neighborX] = true;
            parent[neighborY][neighborX] = { currX, currY };
        }
    }
}

// Jika tidak ditemukan jalur, kembalikan jalur kosong
return {};

// Fungsi mencetak map dengan jalur terpendek
void printMapWithPath(vector<vector<char>>& map, const vector<Position>& path)
{
    for (const auto& pos : path)
    {
        map[pos.y][pos.x] = 'x';    //Print X utk tiap path
    }

    Position start = path.front();
    Position end = path.back();
    map[start.y][start.x] = 'B';    //Biar path awal print 'B'
    map[end.y][end.x] = 'A';        //Biar path akhir print 'A'

    for (const auto& row : map)
    {
        for (const auto& cell : row)
        {
            cout << cell;
        }
        cout << endl;
    }
}

```

Adapun main functionnya adalah sebagai berikut :

```
int main()
{
    int X, Y;
    cin >> X >> Y;

    vector<vector<char>> map(Y, vector<char>(X));

    for (int i = 0; i < Y; i++)
    {
        for (int j = 0; j < X; j++)
        {
            cin >> map[i][j];
        }
    }

    Position start, end;

    for (int i = 0; i < Y; i++)
    {
        for (int j = 0; j < X; j++)
        {
            if (map[i][j] == 'A')
            {
                start = { j, i };
            }
            else if (map[i][j] == 'B')
            {
                end = { j, i };
            }
        }
    }

    vector<Position> path = findShortestPath(map, start, end);

    printMapWithPath(map, path);

    return 0;
}
```

Program diatas memiliki algoritma sebagai berikut :

- Program membaca input ukuran peta X dan Y dari user.
- Program membaca matriks karakter map yang merepresentasikan peta dari pengguna.
- Program mencari posisi awal (start) dan posisi akhir (end) dalam matriks map.
- Program menginisialisasi variabel maxX dan maxY dengan ukuran peta (map[0].size() dan map.size()).
- Program juga menginisialisasi matriks visited dengan ukuran maxY x maxX dan setiap elemennya diisi dengan nilai false. Matriks visited digunakan untuk menandai posisi yang telah dikunjungi selama pencarian jalur.
- Program kemudian menginisialisasi matriks parent dengan ukuran maxY x maxX dan setiap elemennya diisi dengan posisi (-1, -1). Matriks parent digunakan untuk menyimpan posisi tetangga sebelumnya yang menuju ke suatu posisi pada jalur terpendek.
- Kemudian, Program menginisialisasi antrian (queue) q dan masukkan posisi start ke dalam antrian. Set posisi start sebagai posisi yang telah dikunjungi (visited[start.y][start.x] = true).
- Selama antrian tidak kosong, Program akan melakukan hal-hal berikut:

- Ambil posisi curr dari depan antrian (q.front()) dan hapus posisi tersebut dari antrian (q.pop()).
- Program memeriksa apakah curr merupakan posisi akhir (currX == end.x && currY == end.y).
 - Jika iya, maka jalur terpendek telah ditemukan. Program membangun jalur dari posisi akhir ke posisi awal menggunakan matriks parent. Program menyimpan jalur tersebut dalam vektor path dan kembalikan sebagai hasil pencarian jalur terpendek.
 - Jika tidak, lanjutkan ke langkah berikutnya.
- Program mengeksplorasi tetangga-tetangga yang belum dikunjungi dari posisi curr. Tetangga-tetangga tersebut adalah posisi sebelah kiri, kanan, atas, dan bawah dari curr.
- Untuk setiap tetangga, Program memeriksa apakah tetangga tersebut valid (di dalam batas peta), bukan merupakan rintangan (map[neighborY][neighborX] != '#'), dan belum dikunjungi (!visited[neighborY][neighborX]).
- Jika semua kondisi terpenuhi, program memasukkan tetangga ke dalam antrian dan menandai tetangga sebagai telah dikunjungi (visited[neighborY][neighborX] = true), dan simpan posisi curr sebagai parent dari tetangga (parent[neighborY][neighborX] = { currX, currY }).

- Jika proses pencarian jalur terpendek selesai tanpa menemukan jalur, Program mengembalikan jalur kosong ({}).

- Fungsi printMapWithPath digunakan untuk mencetak peta dengan menandai jalur terpendek menggunakan karakter 'x'. Posisi awal ditandai dengan karakter 'B' dan posisi akhir ditandai dengan karakter 'A'.

- Di dalam fungsi main, Program akan menjalankan algoritma pencarian jalur terpendek dengan memanggil fungsi findShortestPath dengan argumen peta, posisi awal, dan posisi akhir.

- Program mencetak peta dengan jalur terpendek menggunakan fungsi printMapWithPath.