Implementasi Teori Graf pada Algoritma Pemecahan *Game* Onet untuk Optimasi Pencocokan *Tile*

Ranashahira Reztaputri - 13523007[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]ranashahira.rez@gmail.com, 13523007@std.stei.itb.ac.id

***Abstrak*—Permainan teka-teki memiliki popularitas terutama di kalangan masyarakat yang menyukai tantangan yang mengasah otak. Salah satu permainan teka-teki yang cukup populer adalah Onet. Onet merupakan game pencocokan ubin di mana ubin yang dicocokkan tidak boleh dihalangi ubin lain dan hanya bisa dihubungkan dengan garis yang memiliki maksimal 2 belokan. Memainkan permainan teka-teki membuat pemain berpikir keras bagaimana cara untuk memenangkan permainan secepat mungkin. Makalah ini bertujuan untuk meneliti cara pemecahan permainan Onet. Melihat adanya keterhubungan antar ubin pada permainan Onet, graf dapat digunakan sebagai salah satu pendekatan yang tepat untuk mengoptimasi langkah dalam permainan Onet. Dengan memanfaatkan algoritma traversal graf, yaitu BFS, ditemukan cara optimasi pemilihan langkah pada permainan Onet.**

***Kata Kunci*—BFS, Graf, Onet, Optimasi.**

I. PENDAHULUAN

Di era modern saat ini, semakin banyak beredar berbagai jenis permainan teka-teki yang menawarkan konsep dan tantangan unik. Fenomena meningkatnya variasi permainan teka-teki ini tidak terlepas dari tingginya popularitas genre tersebut di kalangan berbagai kelompok usia. Popularitas tersebut dipengaruhi oleh daya tarik mendasar dari permainan teka-teki yang mampu merangsang kemampuan berpikir logis, mengasah ketajaman otak, serta membangkitkan rasa penasaran yang mendalam. Sensasi menantang yang dihadirkan permainan teka-teki membuat para pemain terdorong untuk terus berusaha memecahkan tantangan yang disajikan hingga akhirnya berhasil menyelesaikan permainan dengan sempurna.

Di antara sekian banyak permainan teka-teki yang beredar di masyarakat, terdapat salah satu yang menarik untuk dibahas lebih dalam, yaitu Onet. Onet merupakan sebuah permainan teka-teki yang cukup populer di Indonesia, terutama di kalangan generasi yang lebih tua. Permainan ini terinspirasi dari permainan klasik asal Tiongkok, yaitu Mahjong, dengan mekanisme yang lebih disederhanakan namun tetap menantang. Tujuan utama dari permainan Onet adalah mencocokkan dua ubin dengan gambar yang sama, di mana kedua ubin tersebut hanya dapat dihubungkan dengan garis yang memiliki maksimal dua belokan. Permainan dianggap selesai dan dimenangkan jika seluruh pasangan gambar pada papan permainan berhasil dicocokkan dan dihilangkan sebelum waktu yang disediakan habis.

Meskipun konsep dasar permainan Onet tampak sederhana dan mudah dimengerti, pada kenyataannya permainan ini memerlukan tingkat ketelitian yang tinggi serta kemampuan berpikir cepat untuk menyelesaikan tantangan dalam batas waktu yang diberikan. Terdapat pula kondisi di mana papan permainan menyajikan banyak pilihan pasangan ubin yang dapat dicocokkan secara bersamaan. Situasi ini memunculkan pertanyaan tentang hubungan antara urutan dalam memilih pasangan gambar yang dicocokkan dengan efisiensi waktu. Untuk menjawab pertanyaan tersebut, salah satu pendekatan yang dapat digunakan adalah dengan menerapkan konsep teori graf. Konsep ini telah banyak diterapkan dalam penyelesaian berbagai jenis permainan teka-teki lainnya, seperti Tic Tac Toe, Connect Four, dan permainan strategi lainnya yang melibatkan pengambilan keputusan optimal. Oleh karena itu, muncul pertanyaan apakah pendekatan teori graf juga dapat diterapkan secara efektif untuk menyelesaikan permainan Onet dengan lebih optimal. Untuk menguji hal tersebut, akan dilakukan eksplorasi penerapan teori graf pada permainan Onet guna mengoptimalkan strategi penyelesaian teka-teki.

II. LANDASAN TEORI

A. Permainan Onet

Onet adalah permainan teka-teki yang menguji keterampilan pemain dalam mencocokkan pasangan ubin yang memiliki gambar identik. Permainan ini terinspirasi dari konsep permainan klasik Mahjong, tetapi dengan aturan yang lebih sederhana dan mudah dipahami. Tujuan utama dari permainan Onet adalah mencocokkan semua pasangan gambar pada papan permainan hingga seluruh ubin berhasil dihilangkan. Permainan dianggap selesai ketika seluruh ubin berhasil dicocokkan sebelum waktu yang disediakan habis. Prinsip dasar dalam Onet adalah mencocokkan dua ubin dengan gambar yang sama, dengan ketentuan kedua ubin tersebut dapat dihubungkan oleh sebuah garis yang memiliki maksimal dua kali belokan dan tidak dihalangi oleh ubin lain, jika sebaliknya, maka pasangan ubin tersebut tidak dapat dicocokkan.

Secara garis besar, permainan Onet memiliki beberapa aturan dasar, yaitu:

1. Objektif permainan adalah untuk mencocokkan seluruh ubin dengan gambar yang sama hingga papan permainan kosong.

2. Dua ubin dengan gambar yang sama hanya dapat dicocokkan jika dapat dihubungkan dengan garis yang memiliki maksimal dua belokan dan tidak dihalangi oleh ubin lainnya.
3. Beberapa versi dari permainan Onet menerapkan batas waktu yang mengharuskan pemain menyelesaikan teka-teki dalam waktu tertentu
4. Beberapa versi dari permainan Onet juga menyediakan fitur bantuan seperti petunjuk dan pengacakan ulang papan untuk membantu pemain.

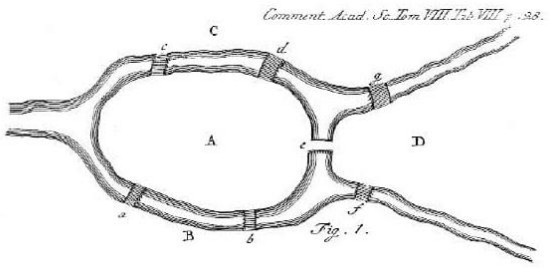
Meskipun memiliki aturan yang sederhana, permainan Onet memerlukan strategi dan perencanaan yang matang dalam memilih pasangan ubin yang dicocokkan. Kompleksitas muncul ketika terdapat banyak pilihan pasangan yang dapat dihubungkan secara bersamaan. Hal ini menimbulkan tantangan bagi pemain untuk menentukan urutan pencocokan yang optimal guna memastikan semua ubin dapat dihilangkan dari papan permainan.

Gambar 2.1 Cuplikan permainan Onet
Sumber: Next Generation Indonesia

B. Teori Graf

Teori graf lahir pada abad ke-18. Pada saat itu, Leonard Euler, seorang ilmuwan asal Swiss, dikenal sebagai ilmuwan ternama dalam bidang matematika. Persoalan mengenai graf mulai muncul dari kota bernama Konigsberg (sekarang kaliningrad), yang berada di Rusia. Terdapat sungai yang bernama Sungai Pregel yang melewati Kota Konigsberg. Sungai ini membagi kota menjadi empat wilayah yang dihubungkan oleh tujuh wilayah jembatan. Melihat jembatan tersebut, mulai muncul pertanyaan di benak masyarakat kota, apakah mungkin untuk melintasi jembatan tersebut dengan cara yang tidak hanya melewati semua jembatan, tetapi juga melewati setiap jembatan tepat satu kali? Untuk mengetahui jawaban dari pertanyaan itu, masyarakat kota biasanya mencoba-coba untuk melintasi semua jembatan tepat satu kali. Namun, masyarakat kota tidak pernah berhasil melakukannya. Euler sebagai matematikawan memiliki pendekatan yang berbeda terhadap masalah ini dan mencoba untuk melihat permasalahan ini dari sudut pandang matematika. Dia menyadari bahwa topik ini bisa menjadi bagian dari bidang matematika yang baru.

Solusi awal Euler terhadap masalah ini adalah mempertimbangkan daratan dengan nama A, B, C, dan D, serta menyebut nama ketujuh jembatan itu sebgai a, b, c, d, e, f, dan g.



Gambar 2.2. Gambar Euler untuk permasalahan jembatan Konnigsberg
Sumber: ResearchGate

Dari gambar Euler ini, muncullah istilah graf. Graf adalah bentuk geometri yang menghubungkan objek-objek. Objek-objek di dalam graf disebut sebagai simpul atau *node*. Objek-objek ini dihubungkan oleh busur yang disebut sisi atau *edge*. Graf dapat juga didefinisikan sebagai komponen yang terdiri dari himpunan tidak kosong dari simpul-simpul dan himpunan sisi yang menghubungkan sepasang simpul.

Graf memiliki beberapa terminologi penting sebagai berikut:

1. Ketetanggaan
   Dua buah simpul pada graf dikatakan bertetangga jika keduanya terhubung secara langsung.
2. Bersisian
   Untuk sembarang sisi $e = (v_j, v_k)$, e bersisian dengan simpul v_j dan simpul v_k.
3. Derajat
   Derajat dari suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.
4. Lintasan
   Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam graf G adalah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1, e_2, v_2, \dots, v_{n-1}, e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \dots, e_n = (v_{n-1}, v_n)$ sisi-sisi dari graf G.
5. Siklus atau Sirkuit
   Siklus atau sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama.
6. Keterhubungan
   Dua buah simpul v_1 dan v_2 disebut terhubung jika terdapat lintasan v_1 ke v_2.

Berdasarkan hubungan antar simpul, graf dapat dibedakan menjadi:

a) Graf tak berarah
   Simpul A dan B dapat dikunjungi dari dua arah
b) Graf berarah
   Simpul A dan B yang terhubung oleh sisi dari arah A ke B, maka tidak bisa mengunjungi A dari B.

Sementara berdasarkan bobot dari sisi, graf dapat dibedakan menjadi:

a) Graf tak berbobot
   Merupakan graf dengan edge yang bobotnya seragam

dan hanya bermakna terdapat hubungan antar node.

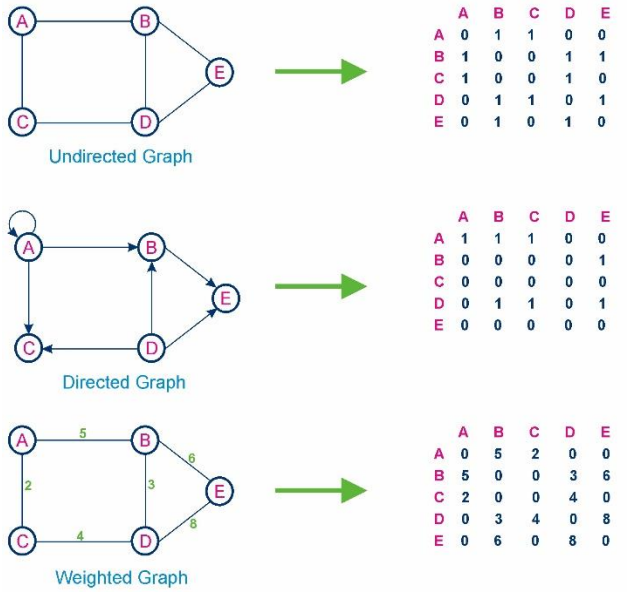
b) Graf berbobot

Merupakan graf dengan edge yang dapat memiliki bobot berbeda-beda. Bobot pada edge ini bisa jadi berupa biaya, jarak, atau waktu yang harus ditempuh jika menggunakan edge tersebut.

Selain memiliki berbagai terminologi, graf juga memiliki berbagai representasi dalam bentuk selain gambar, di antaranya adalah:

1. Matriks Ketetanggaan

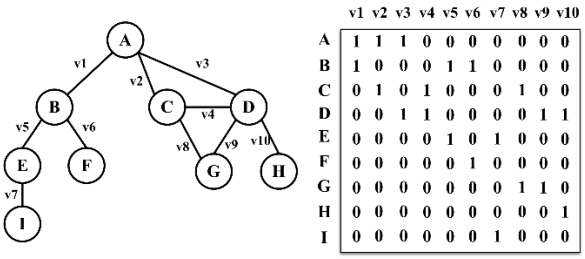
   Menyatakan ketetanggaan dua buah simpul dengan menggunakan matriks, di mana simpul berada pada baris dan kolom. Nilai matriks diisi sesuai ketetanggaan (0 jika tidak bertetangga, 1 atau bobot jika bertetangga).



Gambar 2.3. Matriks ketetanggaan
Sumber: Study Glance

2. Matriks Bersisian

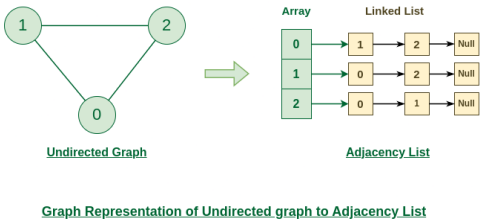
   Menyatakan simpul sebagai baris dan sisi sebagai kolom. Matriks berisi nilai yang menunjukkan apakah suatu simpul bersisian dengan sisi tertentu (bernilai 0 jika tidak berisisian, bernilai 1 atau bobot jika bersisian)



Gambar 2.4. Matriks bersisian
Sumber: O'Reilly

3. Senarai Ketetanggaan

   Disajikan dalam bentuk tabel atau list, di mana tiap simpul yang bertetangga dengan suatu simpul dituliskan dalam list.

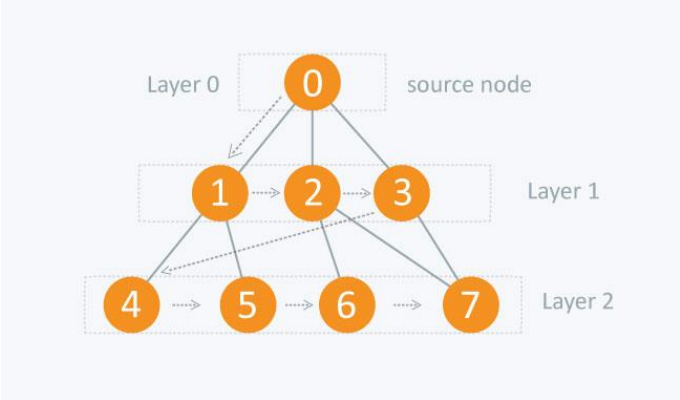


Gambar 2.5. Senarai ketetanggaan
Sumber: Geeksforgeeks

C. BFS (Breadth First Search)

Dalam penjelajahan graf, terdapat dua algoritma yang biasanya diterapkan, yaitu DFS (Depth First Search) dan BFS (Breadth First Search). Kedua algoritma memiliki pendekatan yang berbeda. DFS meninjau tiap simpul sampai kedalaman terdalam, lalu kembali dan meninjau simpul lain sampai kedalaman terdalam. BFS memiliki cara yang berbeda.

BFS atau Breadth First Search adalah algoritma traversal yang memulai traversal dari simpul yang dipilih dan melintasi graf secara berlapis sehingga menjelajahi simpul tetangga. Kemudian, akan terus bergerak menuju simpul tetangga pada tingkat berikutnya.



Gambar 2.6. Graf dengan tingkatannya
Sumber: hackerearth

Seperti namanya, kita diharuskan untuk menjelajahi graf *breadthwise* (dalam segi lebar), yaitu:

1. Pertama, bergeraklah secara horizontal dan kunjungi semua simpul pada tingkat tersebut sampai semua simpul pada tingkat tersebut telah dikunjungi.
2. Pindah ke tingkatan berikutnya.
3. Ulangi terus langkah 1 dan 2 sampai semua simpul pada graf sudah dikunjungi

Kompleksitas dari BFS berbeda tergantung representasi graf yang digunakan. Dengan V adalah jumlah simpul dan E adalah jumlah sisi, kompleksitas BFS adalah $O(V^2)$ untuk matriks ketetanggaan dan $O(V + E)$ untuk senarai ketetanggaan.

III. IMPLEMENTASI DAN PEMBAHASAN

A. Program Utama dan Algoritma Keseluruhan

```cpp
#include<bits/stdc++.h>
using namespace std;
int M, N;
vector<vector<int>> board;
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

bool isValid(int x, int y) {
    return x >= 0 && x < M && y >= 0 && y < N;
}

bool bfs(int startX, int startY, int endX, int endY) {
    queue<tuple<int, int, int, int>> q;
    vector<vector<vector<vector<bool>>>> visited(M, vector<vector<vector<bool>>>(N, vector<vector<bool>>(4, false)));

    if((startX == 0 || startX == M) && startX == endX) return true;
    if((startY == 0 || startY == N) && startY == endY) return true;

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

        if ((nx == endX && ny == endY) || (isValid(nx, ny) && board[nx][ny] == 0)) {
            q.push({startX, startY, 0, i});
            visited[startX][startY][i] = true;
        }
    }

    while (!q.empty()) {
        auto [x, y, turns, dir] = q.front();
        q.pop();

        if (x == endX && y == endY) return true;

        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            int newTurns = turns + (i != dir);
            if (isValid(nx, ny) && newTurns <= 2 && !visited[nx][ny][i]
            && (nx == endX && ny == endY || board[nx][ny] == 0)) {
                q.push({nx, ny, newTurns, i});
                visited[nx][ny][i] = true;
            }
        }
    }
    return false;
}
```

Gambar 3.1. Program utama bagian pertama

```cpp

int evaluateMove(int x1, int y1, int x2, int y2) {
    int saved1 = board[x1][y1];
    int saved2 = board[x2][y2];
    board[x1][y1] = board[x2][y2] = 0;

    int count = 0;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j] != 0) {
                for (int x = 0; x < M; x++) {
                    for (int y = 0; y < N; y++) {
                        if (board[x][y] == board[i][j] && (i != x || j != y)) {
                            if (bfs(i, j, x, y)) {
                                count++;
                            }
                        }
                    }
                }
            }
        }
    }
    board[x1][y1] = saved1;
    board[x2][y2] = saved2;
    return count;
}

pair<int, int> findBestMove() {
    pair<int, int> bestMove = {-1, -1};
    int maxOpenings = -1;

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j] != 0) {
                for (int x = 0; x < M; x++) {
                    for (int y = 0; y < N; y++) {
                        if (board[x][y] == board[i][j] && (i != x || j != y)) {
                            if (bfs(i, j, x, y)) {
                                int openings = evaluateMove(i, j, x, y);
                                if (openings > maxOpenings) {
                                    maxOpenings = openings;
                                    bestMove = {i * N + j, x * N + y};
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return bestMove;
}
```

Gambar 3.2. Program utama bagian kedua

```cpp

void removePair(pair<int, int> p) {
    int x1 = p.first / N, y1 = p.first % N;
    int x2 = p.second / N, y2 = p.second % N;
    board[x1][y1] = board[x2][y2] = 0;
}

int main() {
    cout << "Masukkan ukuran papan (M N): ";
    cin >> M >> N;
    board.resize(M, vector<int>(N));

    cout << "Masukkan papan permainan (" << M << "x" << N << ") dengan angka (0 untuk gambar yang hilang):\n";
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            cin >> board[i][j];
        }
    }

    while (true) {
        cout << "Papan saat ini:\n";
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                cout << board[i][j] << " ";
            }
            cout << endl;
        }

        auto bestMove = findBestMove();
        if (bestMove.first == -1) {
            cout << "Tidak ada langkah yang tersisa, permainan selesai!\n";
            break;
        }

        int x1 = bestMove.first / N, y1 = bestMove.first % N;
        int x2 = bestMove.second / N, y2 = bestMove.second % N;
        cout << "Langkah terbaik: Pasangan (" << x1 << ", " << y1 << ") dengan (" << x2 << ", " << y2 << ")\n";
        cout << "Masukkan pasangan yang ingin dihapus (x1 y1 x2 y2), -1 jika ingin mengikuti move terbaik: ";
        int x;
        cin >> x;
        if (x == -1) {
            removePair({x1 * (N) + y1, x2 * (N) + y2});
        } else {
            cin >> y1 >> x2 >> y2;
            removePair({x * (N) + y1, x2 * (N) + y2});
        }
    }
}
```

Gambar 3.3. Program utama bagian ketiga

Program ini menggunakan bahasa C++ sebagai bahasa untuk implementasi. Pemilihan C++ disebabkan oleh ketersediaan struktur data sehingga dapat memudahkan penerapan algoritma. Secara garis besar, pendekatan traversal yang digunakan pada program ini adalah BFS. Setiap gambar pada ubin dikodekan menjadi angka > 0. Sedangkan, ubin yang sudah dihapus, ditandai dengan angka 0. BFS digunakan untuk mencari tahu apakah dua simpul yang dijelajahi dapat dihubungkan dengan maksimal dua belokan. Diambilnya algoritma BFS daripada DFS adalah dengan alasan memprioritaskan pencarian simpul terdekat terlebih dahulu. DFS mencari sampai simpul terdalam, tetapi jaraknya jauh sehingga kurang cocok untuk optimasi. Selain dua simpul yang jaraknya dekat, pada Onet, simpul yang optimal untuk dihapus adalah simpul yang membuka jalan untuk menghubungkan banyak pasangan ubin. Maka, dalam pencarian *best move*, dihitung berapa jumlah ubin yang dapat dipasangkan setelah menghapus suatu pasang ubin. Algoritma ini akan terus berjalan dan berhenti sampai papan sudah kosong atau tidak ada ubin yang dapat dipasangkan lagi.

B. Traversal Graf

Untuk menelusuri graf dan mencari apakah dua pasang simpul (ubin) dapat dipasangkan, digunakan pendekatan BFS. BFS menelusuri tiap simpul per tingkat. Hal tersebut membuat BFS dipilih sebagai pendekatan dalam mencari langkah terbaik dalam bermain Onet.

Penerapan kode BFS memerlukan validasi input dan array yang berisi nilai arah, yaitu:

```cpp
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

bool isValid(int x, int y) {
    return x >= 0 && x < M && y >= 0 && y < N;
}
```

Gambar 3.4. Array dx dy dan fungsi isValid()

Array dx dy menunjukkan pasangan arah gerakan dengan urutan atas, bawah, kiri, kanan, dari index 0 sampai 3. Sedangkan, fungsi isValid() mengecek dan memastikan bahwa pengecekan simpul tidak akan keluar dari papan berukuran $M \times N$.

```
bool bfs(int startX, int startY, int endX, int endY) {
    queue<tuple<int, int, int, int>> q;
    vector<vector<vector<bool>>> visited(M, vector<vector<bool>>(N, vector<bool>(4, false)));

    if((startX == 0 || startX == M) && startX == endX) return true;
    if((startY == 0 || startY == N) && startY == endY) return true;

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

        if ((nx == endX && ny == endY) || (isValid(nx, ny) && board[nx][ny] == 0)) {
            q.push({startX, startY, 0, i});
            visited[startX][startY][i] = true;
        }
    }

    while (!q.empty()) {
        auto [x, y, turns, dir] = q.front();
        q.pop();

        if (x == endX && y == endY) return true;

        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            int newTurns = turns + (i != dir);
            if (isValid(nx, ny) && newTurns <= 2 && !visited[nx][ny][i]
            && (nx == endX && ny == endY || board[nx][ny] == 0)) {
                q.push({nx, ny, newTurns, i});
                visited[nx][ny][i] = true;
            }
        }
    }
    return false;
}
```

Gambar 3.5. Fungsi BFS

Pada gambar di atas, terdapat potongan kode yang berisi fungsi BFS dengan parameter startX (baris dari simpul awal), startY (kolom dari simpul awal), endX (baris dari simpul akhir), dan endY (kolom dari simpul akhir). Fungsi BFS akan mengembalikan nilai boolean berupa *true* atau *false*. *True* jika kedua ubin dapat dihubungkan dan *false* jika kedua ubin tidak dapat dihubungkan. Dalam implementasi BFS ini, digunakan struktur data queue dan vector (di sini berukuran 3 dimensi). Queue digunakan untuk menampung simpul yang sedang dikunjungi saat ini beserta jumlah belokan dan arahnya. Sedangkan, vector visited terdiri dari posisi baris, koom, dan arah. Vector ini dibuat untuk memastikan bahwa tidak akan dilakukan pengecekan pada simpul yang sama berulang kali.

Kasus pengecekan ubin dapat dibagi menjadi dua. Pertama, kasus normal dimana ubin terletak di bagian tengah papan sehingga bisa dihubungkan dengan cara normal. Kedua, kasus di mana ubin berada di ujung. Jika kedua ubin berpasangan berada di ujung, algoritma normal tidak akan mendeteksi kedua ubin tersebut sebagai pasangan karena tidak bisa digapai (tidak bisa bergerak di luar papan). Untuk mengatasi hal tersebut dilakukan pengecekan apakah kedua pasangan berada di ujung dan sebaris atau sekolom, jika iya, maka sudah pasti kedua ubin bisa dihubungkan sehingga fungsi BFS akan mengembalikan *true*. Selanjutnya, untuk kasus normal, akan dicek ubin di segala arah. Jika terdapat jalur dari simpul awal ke simpul akhir, maka simpul awal beserta arahnya akan dimasukkan ke queue, sebaliknya, maka tidak akan ada anggota di queue sehingga BFS tidak akan dilanjutkan. Setelah simpul awal dimasukkan ke queue, semua simpul yang berhubungan dengan simpul awal akan ditelusuri dan dipastikan bahwa jumlah belokan tidak lebih dari dua. Proses ini akan terus berjalan sampai semua simpul sudah dikunjungi (mengembalikan *false*) atau ditemukan simpul akhir (mengembalikan *true*).

C. Perhitungan Efek Movement

```
int evaluateMove(int x1, int y1, int x2, int y2) {
    int saved1 = board[x1][y1];
    int saved2 = board[x2][y2];
    board[x1][y1] = board[x2][y2] = 0;

    int count = 0;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j] != 0) {
                for (int x = 0; x < M; x++) {
                    for (int y = 0; y < N; y++) {
                        if (board[x][y] == board[i][j] && (i != x || j != y)) {
                            if (bfs(i, j, x, y)) {
                                count++;
                            }
                        }
                    }
                }
            }
        }
    }
    board[x1][y1] = saved1;
    board[x2][y2] = saved2;
    return count;
}
```

Gambar 3.6. Fungsi evaluateMove()

Fungsi evaluateMove() di atas merupakan bagian dari algoritma pencarian *best move*. Fungsi ini akan menghitung jumlah pasangan ubin yang dapat dihubungkan setelah pasangan ubin (x1, y1) dan (x2, y2) dihubungkan. Fungsi akan mengembalikan kondisi papan seperti semua (sebelum pasangan ubin dihapus) dan akan mengembalikan jumlah pasangan ubin yang dapat dihubungkan dengan kondisi (x1, y1) dan (x2, y2) dihapus.

D. Pencarian Movement Teroptimal

```
pair<int, int> findBestMove() {
    pair<int, int> bestMove = {-1, -1};
    int maxOpenings = -1;

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j] != 0) {
                for (int x = 0; x < M; x++) {
                    for (int y = 0; y < N; y++) {
                        if (board[x][y] == board[i][j] && (i != x || j != y)) {
                            if (bfs(i, j, x, y)) {
                                int openings = evaluateMove(i, j, x, y);
                                if (openings > maxOpenings) {
                                    maxOpenings = openings;
                                    bestMove = {i * N + j, x * N + y};
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return bestMove;
}
```

Gambar 3.7. Fungsi findBestMove()

Fungsi findBestMove() merupakan fungsi yang memanfaatkan fungsi evaluateMove(). Fungsi ini mengecek semua ubin yang masih ada dan mengembalikan pasangan ubin dengan jumlah pasangan ubin maksimal yang dapat dihubungkan setelah penghapusan. Pasangan ubin ini disimpan dalam pair yang mengkonversi ukuran dua dimensi (baris dan

kolom) menjadi satu dimensi untuk mempermudah penggunaan struktur data pair.

E. Penghapusan Pasangan Ubin

```
101  void removePair(pair<int, int> p) {
102      int x1 = p.first / N, y1 = p.first % N;
103      int x2 = p.second / N, y2 = p.second % N;
104      board[x1][y1] = board[x2][y2] = 0;
105  }
```

Gambar 3.8. Fungsi removePair()

Fungsi removePair() dipanggil dalam main untuk menghapus pasangan ubin yang dipilih. Fungsi ini akan mengembalikan nilai awal dari x dan y yang sebelumnya diubah menjadi satu dimensi, lalu mengganti nilainya dengan 0 yang berarti menghapusnya dari papan.

IV. EKSPERIMEN

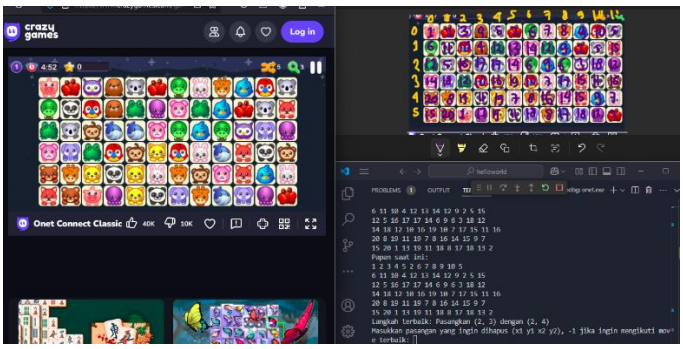
A. Kasus Normal

Gambar 4.1. Papan Onet

Papan Onet di atas terdiri dari gambar-gambar yang berbeda. Jika gambar tersebut diubah ke dalam bentuk angka, maka akan menghasilkan matriks 6×12 dengan bentuk:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 2 & 6 & 7 & 8 & 9 & 10 & 5 \\ 6 & 11 & 10 & 4 & 12 & 13 & 14 & 12 & 9 & 2 & 5 & 15 \\ 12 & 5 & 16 & 17 & 17 & 14 & 6 & 9 & 6 & 3 & 18 & 13 \\ 14 & 18 & 12 & 10 & 16 & 19 & 10 & 7 & 17 & 15 & 11 & 16 \\ 20 & 8 & 19 & 11 & 19 & 7 & 8 & 16 & 14 & 15 & 9 & 7 \\ 15 & 20 & 1 & 13 & 19 & 11 & 18 & 8 & 17 & 18 & 13 & 2 \end{bmatrix}$$

Papan berisi ubin yang sudah diubah ke representasi matriks dan angka dapat diproses oleh program untuk mencari langkah terbaiknya.



Gambar 4.2. Pemrosesan papan Onet (ada kesalahan pada ubin(3, 12) yang seharusnya 13 pada pengujian tersebut)

B. Kasus Papan Kosong

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 2 & 6 & 7 & 8 & 9 & 10 & 5 \\ 6 & 11 & 10 & 4 & 12 & 13 & 14 & 12 & 9 & 2 & 5 & 15 \\ 12 & 5 & 16 & 17 & 17 & 14 & 6 & 9 & 6 & 3 & 18 & 13 \\ 14 & 18 & 12 & 10 & 16 & 19 & 10 & 7 & 17 & 15 & 11 & 16 \\ 20 & 8 & 19 & 11 & 19 & 7 & 8 & 16 & 14 & 15 & 9 & 7 \\ 15 & 20 & 1 & 13 & 19 & 11 & 18 & 8 & 17 & 18 & 13 & 2 \end{bmatrix}$$

Input matriks di atas akan menghasilkan papan yang kosong pada akhir program.

Kondisi awal:

```
Masukkan ukuran papan (M N): 6 12
Masukkan papan permainan (6x12) dengan angka (0 untuk gambar yang hilang):
1 2 3 4 5 2 6 7 8 9 10 5
6 11 10 4 12 13 14 12 9 2 5 15
12 5 16 17 17 14 6 9 6 3 18 13
14 18 12 10 16 19 10 7 17 15 11 16
20 8 19 11 19 7 8 16 14 15 9 7
15 20 1 13 19 11 18 8 17 18 13 2
Papan saat ini:
1 2 3 4 5 2 6 7 8 9 10 5
6 11 10 4 12 13 14 12 9 2 5 15
12 5 16 17 17 14 6 9 6 3 18 13
14 18 12 10 16 19 10 7 17 15 11 16
20 8 19 11 19 7 8 16 14 15 9 7
15 20 1 13 19 11 18 8 17 18 13 2
Langkah terbaik: Pasangkan (2, 3) dengan (2, 4)
Masukkan pasangan yang ingin dihapus (x1 y1 x2 y2), -1 jika ingin mengikuti move terbaik: -1
```

Gambar 4.3. Output awal kasus normal

Kondisi akhir:

```
Langkah terbaik: Pasangkan (4, 5) dengan (4, 11)
Masukkan pasangan yang ingin dihapus (x1 y1 x2 y2), -1 jika ingin mengikuti move terbaik: -1
Papan saat ini:
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 19 0 0 0 0 0 0
0 0 19 0 19 0 0 0 0 0 0 0
0 0 0 0 19 0 0 0 0 0 0 0
Langkah terbaik: Pasangkan (3, 5) dengan (4, 2)
Masukkan pasangan yang ingin dihapus (x1 y1 x2 y2), -1 jika ingin mengikuti move terbaik: -1
Papan saat ini:
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 19 0 0 0 0 0 0 0
0 0 0 0 19 0 0 0 0 0 0 0
Langkah terbaik: Pasangkan (4, 4) dengan (5, 4)
Masukkan pasangan yang ingin dihapus (x1 y1 x2 y2), -1 jika ingin mengikuti move terbaik: -1
Papan saat ini:
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
Tidak ada langkah yang tersisa, permainan selesai!
```

Gambar 4.4. Output akhir kasus normal

C. Kasus Tidak Ada Langkah

```
Masukkan ukuran papan (M N): 4 4
Masukkan papan permainan (4x4) dengan angka (0 untuk gambar yang hilang):
1 2 3 4
2 3 4 5
3 4 5 6
5 6 7 8
Papan saat ini:
1 2 3 4
2 3 4 5
3 4 5 6
5 6 7 8
Tidak ada langkah yang tersisa, permainan selesai!
```

Gambar 4.5. Output ketika tidak ada langkah

Tidak ada ubin yang dapat dipasangkan sehingga tidak ada langkah valid yang dapat dilakukan. Program akan mencetak bahwa tidak ada langkah yang tersisa.

D. Kasus Ujung Papan

```
Masukkan ukuran papan (M N): 2 6
Masukkan papan permainan (2x6) dengan angka (0 untuk gambar yang hilang):
1 2 3 4 5 1
2 3 4 5 6 7
Papan saat ini:
1 2 3 4 5 1
2 3 4 5 6 7
Langkah terbaik: Pasangkan (0, 0) dengan (0, 5)
Masukkan pasangan yang ingin dihapus (x1 y1 x2 y2), -1 jika ingin mengikuti move terbaik: -1
Papan saat ini:
0 2 3 4 5 0
2 3 4 5 6 7
```

Gambar 4.6. Output ketika pasangan ubin berada di ujung

Angka 1 berada di ujung atas papan. Keduanya dapat dihubungkan dengan garis sebanyak dua belokan. Program akan mencetak posisi angka 1 sebagai *move* terbaik karena itu adalah satu-satunya *move* yang valid pada giliran tersebut. Ini merupakan salah satu kasus spesial karena sebenarnya tidak dapat dilakukan traversal pada kedua angka 1 tersebut karena mereka berada di ujung sehingga bagian atas dari keduanya tidak dapat diperiksa.

E. Kasus Ujung Papan yang Berbeda

```
Masukkan ukuran papan (M N): 3 6
Masukkan papan permainan (3x6) dengan angka (0 untuk gambar yang hilang):
6 1 2 3 4 5
2 3 4 5 6 7
3 1 6 7 8 9
Papan saat ini:
6 1 2 3 4 5
2 3 4 5 6 7
3 1 6 7 8 9
Tidak ada langkah yang tersisa, permainan selesai!
```

Gambar 4.7. Output ketika pasangan ubin berada di ujung berbeda

Walaupun angka 1 pada papan sama-sama berada di ujung, tetapi keduanya tidak bisa dihubungkan dengan garis maksimal dua belokan sehingga tidak ada langkah yang valid.

V. KESIMPULAN

Onet merupakan salah satu dari permainan teka-teki yang menggunakan konsep pencocokan. Permainan ini cukup terkenal karena mengasah otak, padahal cara bermainnya sederhana. Untuk memenangkan permainan, pemain hanya perlu mencocokkan ubin dengan gambar yang sama yang tidak terhalang oleh ubin lain dan dapat dihubungkan dengan garis yang memiliki maksimal dua belokan. Jika ditelaah, konsep permainan Onet dapat digambarkan dengan graf. Ubin berperan sebagai simpul, hubungan pasangan ubin yang bisa dihubungkan dapat digambarkan dengan sisi. Pengimplementasian ke dalam bentuk graf ini memungkinkan optimasi pemilihan ubin yang akan dicocokkan tiap gilirannya. Untuk optimasi pencocokan ubin, digunakan BFS. BFS merupakan algoritma traversal yang menjelajahi graf per tingkat. Cara kerja BFS mendukung pencarian ubin berpasangan terdekat. Dengan kombinasi algoritma perbandingan tiap *move*, dihasilkan algoritma untuk mencari langkah terbaik pada permainan Onet.

VI. UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan makalah yang berjudul "Implementasi Teori Graf pada Algoritma Pemecahan *Game* Onet untuk Optimasi Pencocokan *Tile*" dengan baik. Penulis juga mengucapkan terima kasih kepada pihak-pihak yang telah mendukung dalam penulisan makalah ini, yaitu:

1. Bapak Dr. Rinaldi Munir, Bapak Dr. Rila Mandala, serta Bapak Arrival Dwi Sentosa, M.T selaku dosen-dosen pengajar IF1220 Matematika Diskrit atas ilmu, pengajaran, dan bimbingan yang telah diberikan kepada penulis selama perkuliahan,
2. Orang tua penulis yang telah memberikan dukungan dan semangat hingga saat ini,
3. Teman-teman penulis yang telah berjuang bersama-sama dan memberikan dukungan dalam menyelesaikan makalah ini,

Akhir kata, penulis berharap makalah ini dapat memberikan wawasan serta mendorong pembaca untuk mengeksplorasi algoritma graf lebih dalam.

REFERENCES

[1] R. Munir, "Graf (Bagian 1)," https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf. (diakses 4 Januari 2025).

[2] R. Munir, "Graf (Bagian 2)," https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf. (diakses 4 Januari 2025).

[3] W. Gozali and A. F. Aji, *Pemrograman Kompetitif Dasar*. Jakarta: Gramedia, 2024.

[4] R. J. Wilson, *Introduction to Graph Theory*, 4th ed. London: Pearson Education, 1996.

[5] M. Amin, "Graph Theory: History, Applications, and Vision," https://www.researchgate.net/publication/383218821_Graph_Theory_History_Applications_and_Vision. (diakses 5 Januari 2025).

[6] HackerEarth, "Breadth First Search (BFS)," https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/. (diakses 5 Januari 2025).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2024

Ranashahira Reztaputri
13523007

LAMPIRAN

Video penjelasan makalah: https://youtu.be/zxCTZbezhWY