Penerapan Algoritma Flood Fill untuk Mengurangi Ruang Pencarian pada Pencarian Solusi Puzzle Pentomino

Adhitya Ramadhanus 13511032
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511032@std.stei.itb.ac.id

***Abstract*—Dewasa ini banyak sekali permainan puzzle yang digemari masyarakat seperti Sudoku, jigsaw, dan puzzle pentomino. Permainan puzzle digemari karena selain dapat mengasah otak juga dapat digunakan untuk mengisi waktu luang. Pentomino adalah bentuk geometri yang terdiri atas 5 bujur sangkar yang saling menempel. Pentomino adalah turunan dari polyomino. Permainan puzzle pentomino terdiri atas 12 pentomino dengan bentuk yang berbeda yang dapat diubah orientasinya sehingga dapat menempati kotak atau bentuk apapun yang sudah disiapkan. Paper ini akan membahas penerapan algoritma flood fill untuk mengurangi ruang pencarian pada pencarian solusi puzzle pentomino .**

***Index Terms*—Pentomino Puzzle, Flood Fill, DFS, Search Pruning.**

I. PENDAHULUAN

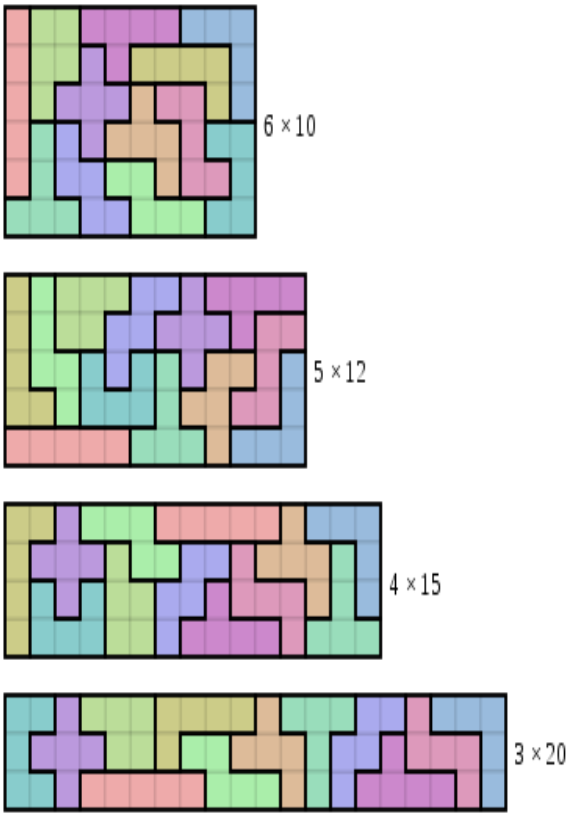
Permainan puzzle pentomino terdiri atas 12 macam pentomino dengan bentuk bermacam-macam, 12 bentuk tersebut berbeda-beda dan memiliki jumlah rotasi yang berbeda. Pentomino-pentomino tadi dapat diputar dan dibalik sedemikian rupa supaya 12 pentomino tadi dapat dimasukkan kedalam suatu bentuk geometri seperti persegi, persegi panjang yang mengandung 60 kotak.

Papan tempat pentomino puzzle diletakkan haruslah memiliki 60 kotak yang mana kotak tersebut adalah kotak-kotak yang ukurannya serupa dengan kotak pada pentomino supaya puzzle dapat diselesaikan.

Layaknya sebagian besar puzzle dan permasalahan yang serupa, puzzle pentomino dapat diselesaikan dengan strategi brute force,DFS,BFS.

Hingga saat ini jumlah solusi yang telah ditemukan untuk puzzle pentomino pada papan berukuran 6x10 adalah 2339 solusi, pada papan 5 x 12 adalah 1010 solusi, pada papan 4 x 15 adalah 368 solusi, pada papan 3x 20 adalah 2 solusi.

Terdapat algoritma yang hingga saat ini menjadi algoritma paling efisien dalam menyelesaikan puzzle pentomino yaitu algoritma dancing link yang ditemukan djikstra.



Gambar 1.1 Papan puzzle pentomino

Pada paper ini algoritma flood fill akan dibuat sebagai fungsi yang memetakan daerah-daerah hole/gap/celah yang ada pada papan permainan.

Selanjutnya akan ditentukan apakah celah ini dapat ditempati oleh pentomino atau tidak.

Jika tidak maka kondisi papan sekarang dianggap tidak akan menghasilkan solusi puzzle sehingga tidak perlu dilanjutkan.

II. DASAR TEORI

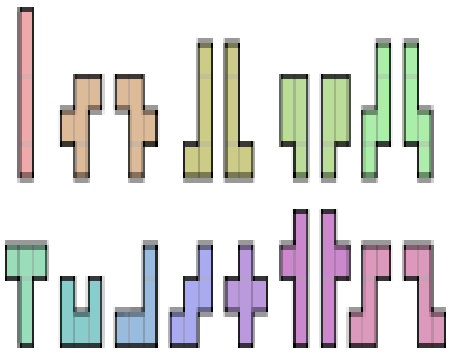
A. Pentomino

Polyomino adalah bidang geometri yang dibentuk oleh satu atau lebih bujur sangkar yang menempel.

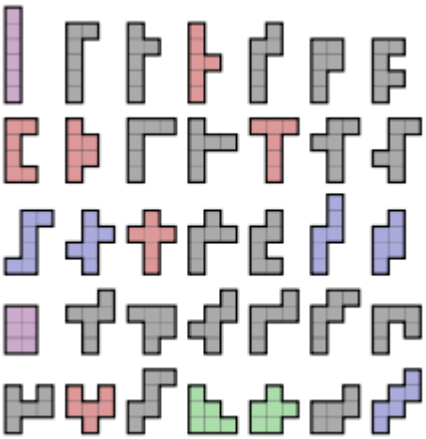
Bujur sangkar tersebut dinamakan cells. Terdapat beberapa jenis polyomino seperti monomino (1 cell), domino (2 cell), tromino (3 cell).

Pentomino adalah salah satu bentuk polyomino yaitu polyomino dengan cell berjumlah 5. Ada 12 kemungkinan pentomino yang dapat dibentuk dari 5 cell tersebut.

Selain 12 bentuk tersebut juga ada refleksi dari 12 bentuk tersebut (beberapa tidak memiliki refleksi), sehingga total ada 18 bentuk pentomino yang berbeda.



Gambar 2.1 Jenis pentomino



Gambar 2.2 Jenis Hexomino

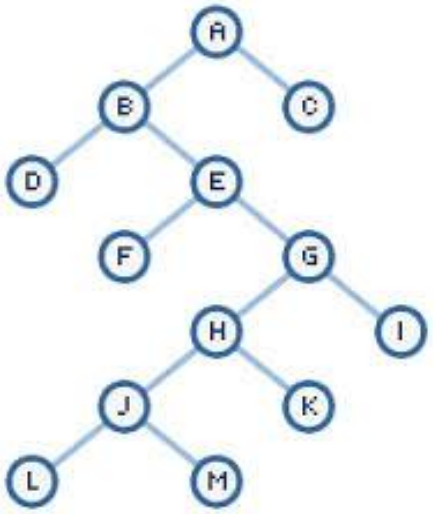
B. Depth-First Search

Pada dasarnya DFS adalah metode pencarian uninformed yang dimulai dari akar lalu menuju ke anak pertama lalu ekspansi terus hingga ke simpul solusi atau ke suatu simpul yang tidak memiliki anak.

Jika DFS berada pada suatu simpul yang tidak memiliki anak maka DFS akan melakukan backtracking ke simpul sebelumnya lalu mencoba anak-anak lain dari simpul tersebut.

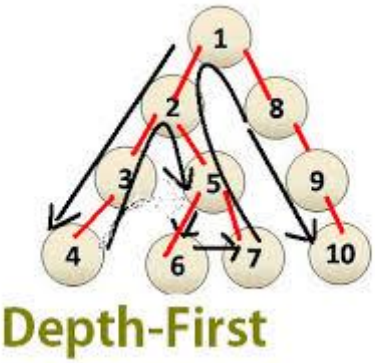
Terdapat kemungkinan sebuah algoritma DFS tidak dapat menemukan solusi atau waktu untuk mencapai solusi kurang feasible jika kedalaman solusi cukup jauh atau simpul backtrack cukup jauh.

Depth-First Search dapat diimplementasikan dalam struktur data stack atau menggunakan fungsi rekursif.



Gambar 2.3 Contoh Graph

Pada contoh, algoritma DFS akan menelusuri graph mulai dari A -> B -> D -> E -> F -> G -> H-> J -> L -> M -> K -> I ->C.



Gambar 2.4 Contoh lain penelusuran DFS

Kompleksitas ruang dan waktu dari DFS adalah O(|E|) dimana |E| adalah jumlah sisi pada suatu graf.

Aplikasi DFS antara lain adalah pencarian jalur keluar dari suatu labirin, Graph traversing (jika kedalaman pohon tidak terlalu dalam), Topological sorting, menemukan connected component pada suatu graf.

Pseudo-code Depth-First Search :

```
Procedure  DFS(Graph G, Vertex V){
        Marked V as visited
        For each (Vertex V1 in G.AdjacencyList(V)){
            If (V1 not visited){
                DFS(G,V1);}
        }
}
```

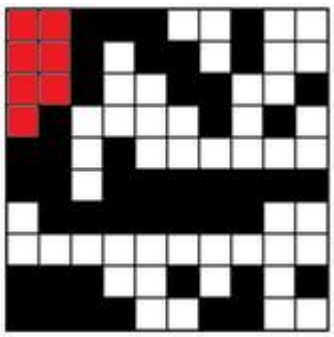
C. Flood Fill

Flood fill adalah algoritma pencarian connected area pada suatu multidimensional array. Pada dasarnya flood fill adalah turunan dari algoritma DFS pada kasus khusus yaitu DFS pada multidimensional array.

Multidimensional array pada dasarnya adalah graf akan tetapi dibandingkan mentransformasi multidimensional array menjadi matriks ketetanggan maka dibuatlah algoritma flood fill.

Algoritma flood fill akan "mewarnai" node-node pada connected area yang ditemukan sehingga dapat ditentukan luas dari connected area tersebut.

Algoritma Flood Fill dapat diimplementasikan dengan struktur data stack (rekursif) dan queue (non rekursif).



Gambar 2.5 Contoh pewarnaan connected area

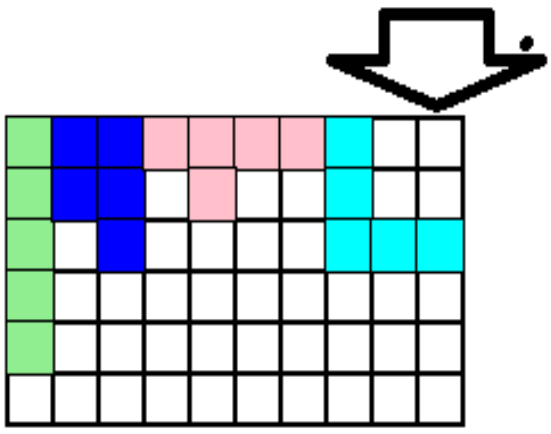
Gambar diatas menunjukkan daerah pada matriks yang telah "diwarnai" oleh algoritma flood fill. Pada implementasi nyatanya, flood fill mewarnai suatu daerah dengan mengganti nilai dari daerah tersebut dengan nilai tertentu yang didefinisikan oleh programmer.

Pseudo-code Flood Fill :

```
Procedure Flood-fill (node, target-color,
replacement-color){
          If node.color
          node.color <- replacement-color
          Flood-fill(north,target-
color,replacement-color)
          Flood-fill(West,target-
color,replacement-color)
          Flood-fill(South,target-
color,replacement-color)
          Flood-fill(East,target-
color,replacement-color)
Return.
```

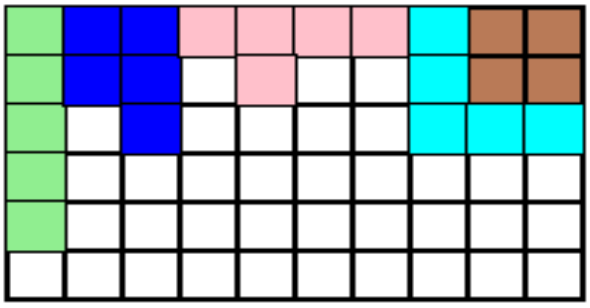
III. IMPLEMENTASI FLOOD FILL

Implementasinya adalah pertama-tama akan dicari petak awal dimana hole area pertama ditemukan.



Gambar 3.1 Gap/Hole

Setelah hole pertama ditemukan maka algoritma flood fill mulai bekerja dan memetakan keseluruhan hole tersebut.



Gambar 3.2 Hasil pemetaan

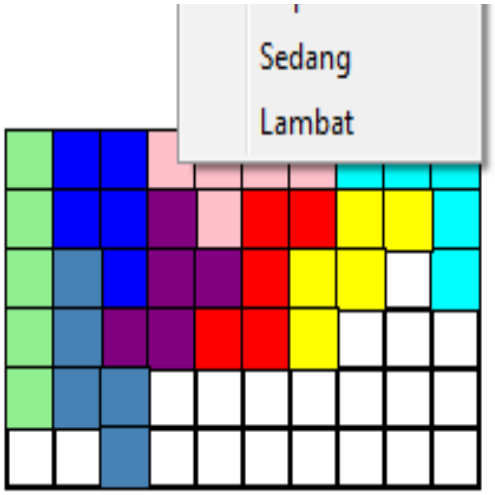
Daerah yang diberi warna cokelat adalah hole yang dipetakan oleh algoritma Flood Fill, dari pemetaan tersebut dapat diketahui luas hole tersebut berapa petak.

Dari informasi luas tersebut dapat diketahui apakah solusi puzzle dapat ditemukan dari state ini yaitu dengan cara menghitung apakah luas hole adalah kelipatan dari 5.

Jika luas hole bukan kelipatan 5 maka tidak mungkin ada pentomino yang dapat ditempatkan di hole tersebut sehingga state tersebut dimatikan dan tidak akan diekspansi lagi nantinya

Dengan demikian, state pada gambar diatas adalah state yang tidak akan menghasilkan solusi puzzle pentomino sehingga tidak perlu ditelusuri lebih lanjut.

Contoh lain adalah pada kasus berikut :



Gambar 3.3 Contoh kasus state buntu

Pad a gambar diatas terdapat hole yang luasnya 2 petak pada ujung kiri bawah papan permainan. Hole ini tidaklah dapat ditempati oleh pentomino manapun. Akan tetapi Flood Fill tidak menelusuri hole tersebut melainkan hole yang berada disebelahnya yang lebih besar. Oleh karena jumlah petak adalah 60 buah dan pentomino memiliki 5 petak maka untuk setiap n

$1 <= n <= 12$

$60 - n*5 = X$

$X \bmod 5 = 0$

Maka jika terdapat hole yang luasnya L, dimana

$L \bmod 5 = Y$

$Y \neq 0$

maka akan terdapat pula hole yang luasnya Y1 dimana

$Y1 \bmod 5 = 5 - Y$

Sehingga algoritma flood fill cukup memetakan satu daerah hole saja untuk memverifikasi apakah state tersebut mungkin menghasilkan solusi atau tidak.

Implementasi Flood Fill pada C#
(Implementasi menggunakan queue, implementasi menggunakan DFS/stack belum dibuat)

Source code dibawah adalah potongan sebagian code.

```
private bool CheckWithFloodFill(int Row, int Col)
{
        int[] PosGap = new int[2];
        int GapSize = 0;
        PosGap[0] = Row;
        PosGap[1] = Col;
        Queue GapQueue = new Queue();
        GapQueue.Enqueue(PosGap);
        while(GapQueue.Count!=0)
        {
                int[] Front= new int[2];
                Front=(int[])GapQueue.Dequeue();
                int i = Front[0];
                int j = Front[1];
                if (BoardCopy[i, j] == 0)
                {
                        GapSize++;
                        BoardCopy[i, j] = 2;
                        //search north
                        if (i - 1 >= 0)
                        {
                                int[] PosNorth = new
int[2];
                                PosNorth[0] = i - 1;
                                PosNorth[1] = j;

        GapQueue.Enqueue(PosNorth);
                        }
                        //search west
                        if (j - 1 >= 0)
                        {
                                int[] PosWest = new
int[2];
                                PosWest[0] = i;
                                PosWest[1] = j - 1;

        GapQueue.Enqueue(PosWest);
                        }
```

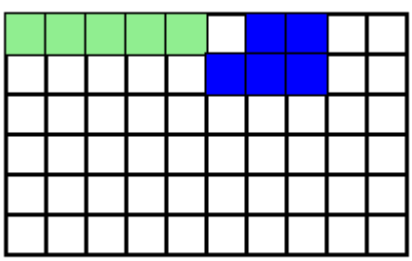
```
        //search south
        if (i + 1 < iRows)
        {
        int[] PosSouth = new int[2];

        PosSouth[0] = i + 1;

        PosSouth[1] = j;

        GapQueue.Enqueue(PosSouth);
        }
        //search east
        if (j + 1 <iCols)
        {
        int[] PosEast = new int[2];
        PosEast[0] = i;
        PosEast[1] = j + 1;

        GapQueue.Enqueue(PosEast);
        }
        }
}
if (GapSize % 5 == 0)
{
        return true;
}
else
{
        return false;
}
}
```

IV. ANALISIS

Algoritma Flood Fill pada dasarnya adalah algoritma turunan dari DFS untuk mencari connected component pada suatu graph.

Algoritma Flood Fill bekerja dengan efisien dan baik pada pencarian connected component pada suatu multidimensional array.

Search pruning pada pencarian solusi puzzle pentomino menggunakan algoritma flood fill dapat meningkatkan performansi dari pencarian solusi tersebut.



Gambar 4.1 Analisis pemangkasan state

Pada contoh gambar di atas, jika tidak menggunakan pengecekan hole menggunakan flood fill maka algoritma pencarian seperti DFS misalnya akan terus mencari hingga kedalaman 12 dikarenakan DFS bersifat uninformed search.

Dengan memangkas ruang pencarian menggunakan algoritma flood fill maka pencarian tidak perlu menelusuri semua kemungkinan state yang ada.

Pada kasus di atas state yang dapat dipangkas adalah
Jumlah cara menempatkan sisa pentomino,

$$\frac{n!}{(n-r)!} \prod_{k=i}^{12} R(k) \text{ , } 2<=k<=12$$

i adalah nomor pentomino pertama yang belum ditempatkan.
R(k) adalah jumlah rotasi dan refleksi yang dimiliki oleh pentomino nomor k.
$\frac{n!}{(n-r)!}$ adalah permutasi posisi sisa pentomino.

State yang dapat dipangkas umumnya cukup signifikan terutama jika sate yang tidak akan menghasilkan goal berhasil dideteksi di awal pencarian.

V. KESIMPULAN

Algoritma Flood Fill dapat digunakan untuk memangkas ruang pencarian pada pencarian solusi pentomino puzzle secara signifikan sehingga dapat meningkatkan performansi pencarian solusi puzzle.

Dibandingkan pencarian tanpa pemangkasan ruang pencarian terdapat perbedaan yang cukup signifikan dari segi performansi program.

REFERENCES

[1] Munir ,Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika, 2009.

[2] http://lodev.org/cgtutor/floodfill.html
Diakses pada tanggal 20 desember pukul 10.38

[3] http://rosettacode.org/wiki/Bitmap/Flood_fill
Diakses pada tanggal 20 desember pukul 11.12

[4] Halim, Felix, "Competitive Programming : Increasing the Lower Bound of Programming Contests", National University of Singapore , 2010.

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, 20 Desember 2013

ttd

Adhitya Ramadhanus,13511032