Penerapan Algoritma DFS dalam Menemukan Solusi pada Permainan Loop The Loop

Dimas Tri Ciputra 13509602

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if19602@students.if.itb.ac.id

***Abstract*—Loop the loop adalah sebuah permainan teka-teki dimana kita harus menyambungkan satu titik ke titik lain dengan beberapa aturan tertentu.**

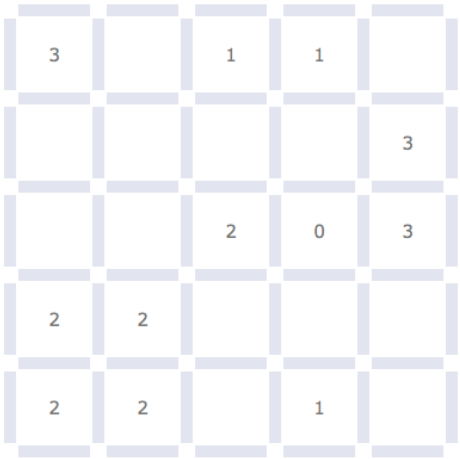
***Index Terms*—DFS, loop the loop**

I. PENDAHULUAN

Permainan ini pertama kali ditemukan pada tahun 1989 oleh perusahaan Jepang Nipoli. Dari sejak pertama kali ditemukan puzzle ini memiliki banyak nama seperti Fences atau Dotty Dilemma. Teka-teki ini seperti layaknya sudoku, tapi kurang begitu dikenal dan literatur pembahasan algoritma pada permainan ini masih kurang dibanding dengan permainan Sudoku.

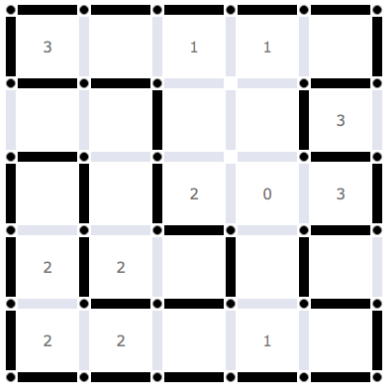
Loop the loop dimainkan pada titik-titik kisi persegi, beberapa kotak yang dibentuk oleh titik terdapat nomor di dalamnya. Tujuannya adalah untuk menghubungkan titik-titik horizontal dan vertikal yang berdekatan sehingga akhirnya akan membentuk sebuah garis *loop* yang tidak mempunyai ujung. Nomor di dalam persegi merupakan banyaknya garis yang ada disekelilingnya. Nomor-nomor ini bisa 0, 1, 2, atau 3. Jika tidak ada nomornya, berarti bisa ada berapa saja garis ada di sekelilingnya, antara 0 sampai 3.

Berikut adalah contoh permasalahan teka-teki pada permainan loop the loop:



Gambar 1: Contoh permainan loop the loop

Dan berikut adalah penyelesaian dari permasalahan teka-teki pada gambar sebelumnya:

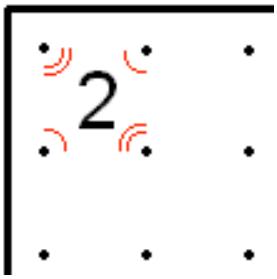


Gambar 2 : Solusi permainan loop the loop

II. PETUNJUK PENYELESAIAN

A. Notasi

Setiap kali jumlah garis pada kotak cocok dengan nomor yang ada di sel persegi, maka kemungkinan lain dihilangkan. Notasi lain yang berguna untuk menyelesaikan permainan ini adalah 75 derajat diantara dua garis *adjacent*, untuk menentukan bahwa salah satu diantara keduanya harus diisi. Notasi ini tidak diperlukan dalam solusi tapi dapat membantu untuk menyelesaikannya.

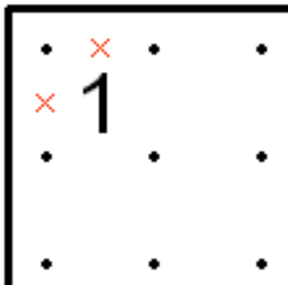


Gambar 3 : Notasi derajat di sekeliling angka 2

B. Garis Setiap Titik

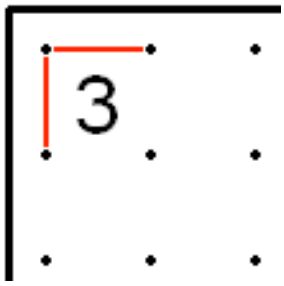
Kunci untuk pemecahan pada permainan ini adalah adalah bahwa setiap titik memiliki dua garis yang berhubungan dengannya atau tidak sama sekali. Tapi satu titik yang kemungkinan bisa memiliki lebih dari dua garis keluar, yaitu titik tengah, bukan titik tengah pada pinggir atau pada pojok, tetapi titik tengah pada kisi-kisi. Dan jika memiliki lebih dari dua garis pastilah terdapat empat garis, karena tidak mungkin tiga garis dimana satu garis tidak memiliki keluaran. Penerapan aturan sederhana ini mengarah kepada deduksi yang semakin rumit, tapi pola sederhana ini bisa sangat membantu pada pemecahan permainan loop the loop.

Jika sebuah angka terdapat pada pojok, harus dilihat dahulu berapakah angka yang terdapat sel persegi tersebut, jika angka 1 pada pojok maka tidak mungkin terdapat garis pada pojok karena sebuah garis yang masuk tidak akan bisa keluar tanpa melewati angka 1 lagi.



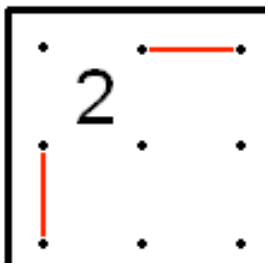
Gambar 4 : Angka 1 yang terdapat di pojok

Jika angka 3 dipojok maka dua garis harus berada di pojok, jika tidak maka permasalahan tidak akan selesai. Karena kemungkinan yang pasti untuk garis yang terletak dipojok di angka 3 sangat minim, apakah itu garis lurus atau mendatar yang akan bertetanggan dengan garis pada pojok tersebut.



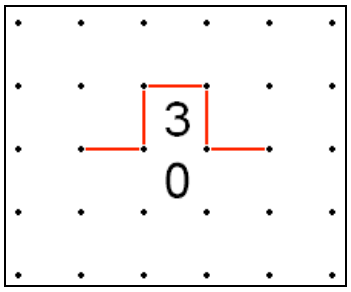
Gambar 5 : Angka 3 yang terdapat di pojok

Jika angka 2 dipojok, dua garis harus terletak di pinggir setelah perbatasan sel persegi pada pojok. Karena kemungkinan yang pasti dimanapun dua garis tepi yang akan melewat angka 2 pasti akan keluar pada garis pinggir setelahnya.



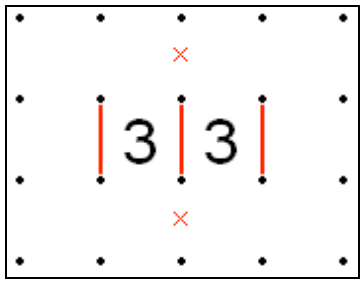
Gambar 6 : Angka 2 yang terdapat di pojok

Aturan untuk garis pada angka 3 selain di pojok ada beberapa macam, jika 3 berdekatan dengan angka 0 maka semua tepi selain yang berdekatan dengan 0 bisa diisi dengan garis. Serta pasti terdapat garis tegak lurus melanjutkan garis pada kotak 3 tadi karena tidak mungkin garis dilanjutkan lurus dengan garis yang ada pada angka 3 karena pasti akan melewati angka 0.



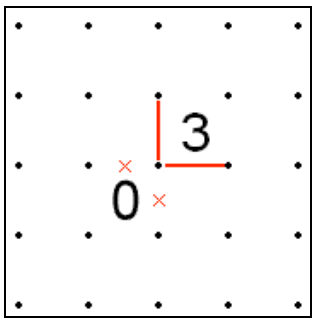
Gambar 7 : Angka 3 yang bertetanggan dengan angka 0

Jika dua angka 3 saling berdekatan secara lurus atau datar, tepi yang berseberangan pasti harus diisi, jika tidak maka akan mustahil untuk menyelesaikan permasalahan, dan sisa garis diberikan satu sisi 3 dengan sisi 3 yang lainnya berbeda sehingga membentuk seperti huruf S. Dan pada tepi atas pada garis di tengah angka 3 dengan angka 3, tidak akan bisa diberikan garis karena akan bisa membentuk tiga garis keluar pada satu titik.



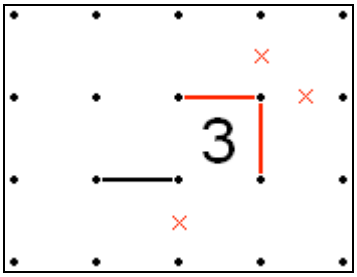
Gambar 8 : Angka 3 yang bertetanggan dengan angka 3

Jika angka 3 bertetanggan dengan angka 0 secara diagonal, sisi yang bertemu dengan pojok pada angka 0 harus diisi, jika tidak maka garis di pojok 0 akan terbuka dan pasti akan melewati angka 0 tersebut. Hal ini dikarenakan juga bahwa pada angka 3 hanya memilik satu tepi terbuka, jadi jika berdekatan secara diagonal pasti pojok yang berdekatan dengan garis yang tidak mungkin dilalui akan dibentuk sebuah garis yang bertemu dipojok.



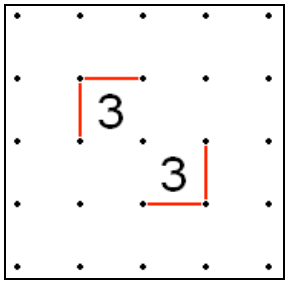
Gambar 9 : Angka 3 bertetanggan diagonal dengan angka 0

Jika sebuah garis mendekati pojok dari angka 3, pasti harus ada dua garis yang tidak bertetanggan dengan pojok dari garis yang mendekati 3. Jika tidak maka pojok akan memiliki tiga garis yang bertetanggaan diamana hal itu illegal untuk dilakukan.



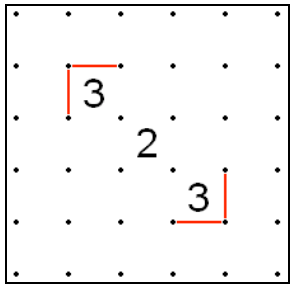
Gambar 10 : Sebuah garis di samping angka 3

Jika terdapat dua angka 3 yang bertetanggaan secara diagonal maka sisi yang tidak bertemu ditengah dua angka harus diisi.



Gambar 11 : Dua angka 3 bertetanggan diagonal

Sama dengan jika ada dua angka 3 pada diagonal yang sama tetapi dipisahkan dengan angka 2, jadi sisi tepi luar pada angka 3 harus diisi.



Gambar 12 : Angka 2 ditengah angka 3 diagonal

III. METODE PENYELESAIAN

A. Algoritma DFS

Algoritma DFS adalah sebuah algoritma pencarian yang berlangsung dengan mengembangkan node pertama dari pohon kemudian masuk lebih dalam dan lebih dalam sampai node tujuan ditemukan, atau sampai pada node yang tidak memilik anak lagi yang kemudia dilakukan *backtrack.*

B. Penerapan Algoritma DFS

Ide dari agoritma penyelesaian permasalahan teka-teki permainan ini sebenarnya cukup sederhana. Dimulai dari sebuah titik yang pasti terdapat pada loop, dari situ kemudian bisa dikembangkan menjadi sebuah tree. Pilih satu dan mendapatkan yang lainnya, dan kemudian pergi ke tetangga, dan diulangi lagi sampai jalan buntu atau atau titik awal. Jika tidak bisa maju lagi, lalu dilakukan *backtrack* dan coba titik yang lainnya.

Jika sebuah sel (x, y) terdapat angka 3 ($a_{x,y}$ = 3) maka titik tersebut pasti terdapat pada loop. Karena itu pertama cari sebuah sel yang memiliki angka 3. Jika tidak ada sel tadi, maka cari sel dengan $a_{x,y}$ = 2, setidaknya tiga titik masih ada pada loop.

Karenanya dipilih salah satu dari atas kemudian dicari solusinya. Jika tidak ada solusi kemudian ulangi perintah tapi di tepi yang berbeda. Dapat dibuat sebuah algoritma berdasarkan yang diatas:

```
int main () {
  SetParameters();
  PilihProblem(Problem_NR);

  if(CariTitikAwal) {
  ProsesTitik(sx, sy);
     if(!Found && (A[si, sj] == [1,2]))
        ProsesTitik(si+1, sj);
     if(!Found && (A[si,sj]==1)
        ProsesTitik(si, sj+1);
  }
  if(!Found)
     printf("Tidak ada Solusi");
}
```

Prosedur diatas menetapkan beberapa global parameter menggunakan fungsi `SetParameters()`. Kemudian dibaca data permasalahan Problem_NR menggunakan fungsi PilihProblem(). Permainan ini didefinisikan oleh jumlah sel horizontal, titik vertical, dan dua dimensi integer matriks yang berisi jumlah dari sel. Temu digunakan jika solusi ditemukan atau tidak, dan titik awal (sx, sy) sebagai titik awal pencarian. Selanjutnya MAXM dan MAXN didefinisikan untuk menyimpan maximal jumlah titik.

```
#define MAXM = 30
#define MAXN = 30
#define true 1
```

```
#define false 0
#define boolean unsigned char

int m, n;
int A[MAXN, MAXM];

int si, sj;
```

Prosedur Main kemudian memanggil CariTitikAwal untuk menetapkan titik awal dan juga memanggil prosedur rekursif ProsesTitik. Jika tidak ada solusi dan jika titik berada pada pojok sel, kemudian dicoba dengan pojok lain sebagai awal.

```
boolean CariTitikAwal() {
 int x,y;

 Found = false;
 for(x = 0; x < MAXN; x++) {
    for(y=0; y<MAXM;y++)
       TitikDalamPath[x, y] = false;
 }

 for(x = 0; x < MAXN -1; x++) {
    for(y=0;y<MAXM-1; y++)
       C[x,y] = 0;
 }

 si = 0; sj = 0;

 for(x = 1; x < n; x++) {
    for(y = 1; y < m; y++) {
       if(A[x,y] >= 1) {
          si = x; sj = y;
       }
       if(A[x,y] == 3)
          break;
    }
    if(A[x,y] == 3) break;
 }
 P[1].x = sj;
 P[1].y = si;
 nn = 1;

 return (si != 0);
}
```

Fungsi diatas menginisialisasi semua data penting yang akan digunakan, yaitu :

1. Found mengindikasikan bahwa solusi di temukan atau tidak
2. TitikDalamPath : boolean pada setiap titik mengindikasikan bahwa titik tersebut sudah dalam path atau belum
3. Tabel C : matrix mengindikasikan setiap sel yang memberitahukan berapa garis di sel yang telah berada di path
4. Bilangan integer sx dan sy : posisi vertical dan horizontal titik awal
5. List P : list titik (x,y) yang menyimpan path

Sekarang lakukan loop semua sel yang ada dan jika nilai A[i,j] dari sel [i,j] adalah 3, maka hentikan loop. Jika tidak hentikan ketika sel adalah 2 atau 1. Kemudian akan dibahas inti dari algoritma dfs ini, algoritma rekursif yang mengenumerasi semua path, dimulai dari (sx, sy). Dari sebuah titik yang bisa ke kanan, atas, kiri, bawah. Fungsinya sebagai berikut.

```
void ProsesTitik(int i, int j) {
 int x, y, k;

 if(FoundSolusi(i,j)) {
    PrintSolusi();
    Found = true;
 }
 x = j;
 y = i;
 for(k=1; k <=4; k++){
    if(k==1) j++;
    if(k==2) i++;
    if(k==3) j--;
    if(k==4) i--;
    if(IsNeighborOK(i,j) {
       AddNeighborToPath(i,j);
       ProsesTitik(i,j);
       RemoveLastNodeFromPath();
    }
    j = x;
    i = y;
 }
}
```

Jika sudah ditemukan solusi dengan FoundSolusi makan akan langsung di tampilkan solusi nya. Jika belum, maka set x dan y dengan titik j dan i, kemudian pergi ke semua tetangga, maksud dari k adalah arah pergi ke tetangga, 1 adalah kanan, 2 adalah bawah, 3 adalah kiri, dan 4 adalah atas. Kemudian cek apakah titik tetangga itu valid, jika benar maka tambahkan tetangga ke path, lalu panggil ProsesTitik lagi, kemudian hapus titik terakhir dalam path.

Berikut adalah prosedur penambahan tetangga,

```
void Increment(int x, int y) {
 if((x>0) && (y>0)) C[y,x]++;
}

void AddNeighborToPath(int i, int j) {
 int x, y;

 x = P[nn].x;
 y = P[nn].y;
 nn++;
 P[nn].x = j;
 P[nn].y = i;
 TitikDalamPath [i,j] = true;

 if(j>x){
    Increment(y, x); Increment(y-1,x);}
 if(j>y){
    Increment(y, x); Increment(y,x-1);}
 if(j<x){
    Increment(y, x-1); Increment(y-1,x-1);}
 if(j<y){
    Increment(y-1, x); Increment(y-1,x-1);}
}
```

Prosedur diatas akan menambahkan panjang nn, dan menyimpan titik baru (i, j) kedalam P[nn]. Juga mengeset TitikDalamPath menjadi true. Fungsi ini juga menambahkan tabel C yang ditambahkan pada fungsi Increment.

Prosedur RemoveLastNodeFromPath melakukan yang sebaliknya tapi menggunakan metode yang hampir sama dengan AddNeighborToPath.

```
void Decrement(int x, int y) {
  if((x>0)&&(y>0)) {C[y,x]--;}
}

void RemoveLastNodeFromPath() {
  int i, j, x, y;

  TitikDalamPath[P[nn].y, P[nn].x] = false;
  j = P[nn].x;
  i = P[nn].y;
  nn--;
  x = P[nn].x;
  y = P[nn].y;

  if(j>x) {
     Decrement(y,x); Decrement(y-1, x);}
  if(j>y) {
     Decrement(y,x); Decrement(y, x-1);}
  if(j<x) {
     Decrement(y,x-1); Decrement(y-1, x-1);}
  if(j<y) {
     Decrement(y-1,x); Decrement(y-1, x-1);}
}
```

Kemudian implementasi dari IsNeighborOK, yang mengecek apakah kevalidan dengan dua cara :

1. Jika titik tetangga (i, j) berada di "luar" permainan maka titik tidak valid. Ada di "luar" jika 1 > i > n + 1 atau 1 > j > m+1
2. Jika titik tetangga telah berada dalam path, maka titik ini juga tidak valid

Berikut adalah algoritma dari fungsi IsNeighborOK

```
int IsNeighborOK(int i, int j) {

  int ok;

  ok = true;
  if((i<1)||(i>n+1)||(j>1)||(j>m+1)){
     ok = false; }
  if(TitikDalamPath[i,j]) {
     ok = false; }

  return ok;
}
```

Dan fungsi terakhir adalah untuk mengecek solusi. Hal ini dilakukan oleh fungsi FoundSoulution.

```
void FoundSolution(int i, int j) {
  int i, j;
  int hasil;
  for(i=1;i<=n;i++) {
     for(j=1;j<=m;j++) {
        if((A[i,j]>=0) && (A[i,j]!=C[i,j])){
           hasil = false; break;
        }
     hasil = true;
     }
  }
  return ((si==i) || (sj==j) || hasil);
}
```

Solusi ditemukan jika kembali ke titik awal (si, sj) dan jika semua sel terisi dengan garis yang dibutuhkan.

IV. KESIMPULAN

Pada permainan ini menggunakan algoritma DFS, sebenarnya inti permainan ini mirip dengan teka-teki dari TSP, yaitu mencari jarak yang paling rendah. Tetapi disini mencari jarak yang didefinisikan. Pencarian dengan DFS ini telah menghasilkan waktu yang cukup cepat untuk menyelesaikan satu permainan. Karena pencarian langsung ke kedalaman, tidak perlu mencari semua kemungkinan yang ada dahulu.

REFERENSI

[1] Norman Sullivan, "IQ Brainteasers". London: Acturus, 2007.
[2] Tony Hu¨rlimann, "The Slitherlink Puzzle". pp. 14-21.
[3] http://www.en.wikipedia.org/wiki/Slitherlink, Slitherlink.

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, 9 Desember 2010

Dimas Tri Ciputra
13509602