Penerapan Algoritma DFS pada Permainan Solitaire

Manuella Ivana Uli Sianipar - 13521051
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521051@mahasiswa.itb.ac.id

***Abstract*—Solitaire adalah salah satu permainan strategi kartu yang sudah dimainkan sejak dulu. Permainan ini dimainkan dengan memindahkan kartu remi yang berisi beberapa set kartu. Kartu-kartu ini awalnya tersusun acak dan pemain harus menyusunnya agar kartu-kartu tersebut berurutan. Untuk mencari semua kemungkinan susunan, dapat digunakan algoritma Depth First Search (DFS).**

Keywords—Depth First Search, solitaire, game

I. Pendahuluan

Permainan solitaire adalah salah satu permainan kartu yang sudah lama dimainkan. Bahkan di komputer-komputer lama permainan ini sudah terinstal. Hampir semua orang mengetahui atau pernah memainkan permainan ini.

Permainan ini dimainkan menggunakan kartu remi yang terdiri atas beberapa set. Kartu-kartu ini diacak dan disusun pada beberapa kolom. Beberapa baris dibuat tertutup dan beberapa dibuat terbuka. Tujuan dari game ini adalah menyusun kartu dengan ketentuan tertentu sehingga semua kartu dapat tersusun berurutan yaitu K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2, As.

Salah satu algoritma yang bisa menyelesaikan permainan ini adalah algoritma *Depth First Search* (DFS). Dibutuhkan pencarian langkah-langkah yang mungkin dilakukan, lalu mengecek apakah langkah-langkah tersebut bisa menyusun semua kartu. Algoritma DFS akan mengecek satu kemungkinan sampai tidak ada langkah lagi, lalu mencari ke kemungkinan yang lain lagi.

II. Dasar Teori

A. *Permainan Solitaire*

Permainan solitaire adalah permainan kartu yang menggunakan kartu remi. Kartu yang digunakan terdiri atas beberapa set (biasanya 4). Setiap set terdiri atas 13 kartu yaitu K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2, As secara berurutan.

Terdapat beberapa istilah dalam permainan solitaire:

1. Stock

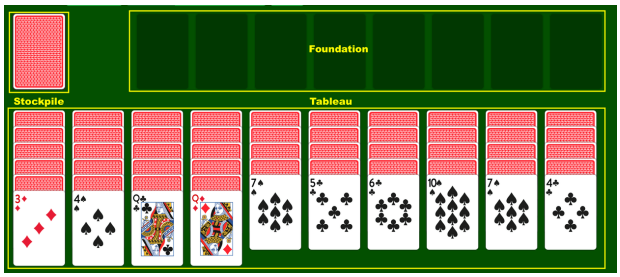
   Tumpukan kartu yang akan digunakan untuk mengambil kartu yang baru.

2. Tableau

   Tempat susunan kartu. Disinilah pemain akan menyusun kartu dengan cara memindahkannya dengan ketentuan tertentu.

3. Foundation

   Tempat kartu yang sudah tersusun disimpan.



Gambar 1 Permainan solitaire (Sumber: [3])

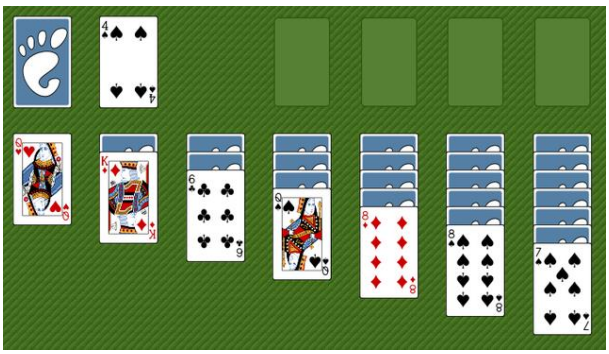
Pada awal permainan kartu akan disusun secara acak pada tiap kolom. Beberapa baris dibuat tertutup dan beberapa baris dibuat terbuka. Kartu yang tertutup akan terbuka apabila kartu dibawahnya sudah diambil.

Pemain dapat memindahkan kartu dengan ketentuan kartu yang dipindah nilainya tepat kurang 1 dari kartu paling bawah kolom tempat kartu akan dipindahkan. Misalnya pada Gambar 1, kartu dengan angka 5 pada kolom keenam dapat dipindahkan ke kolom ketujuh karena 5 nilainya kurang 1 dari 6 (kartu paling bawah di kolom ketujuh).

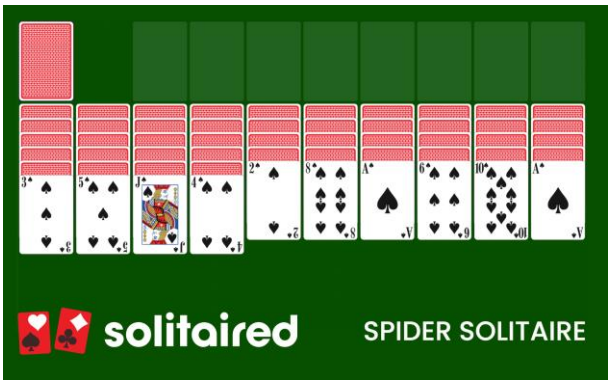
Selain memindahkan kartu pemain juga dapat membuka kartu pada stock. Kartu dari stock akan diambil satu persatu mengisi semua kolom di tableau.

Jika ada baris yang terdapat set berurutan (K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1) maka set tersebut dapat dipindahkan ke foundation. Permainan akan terus berlanjut sampai kartu di stock dan tableau sudah habis tersusun di foundation.

Susunan awal kartu bermacam-macam. Beberapa permainan membuat jumlah barisan kartu sama semua di setiap kolom dan beberapa membuat jumlah barisan yang sama. Kartu remi berisi 4 set jenis kartu, namun pada permainan digital terkadang set kartu yang digunakan bisa lebih banyak.



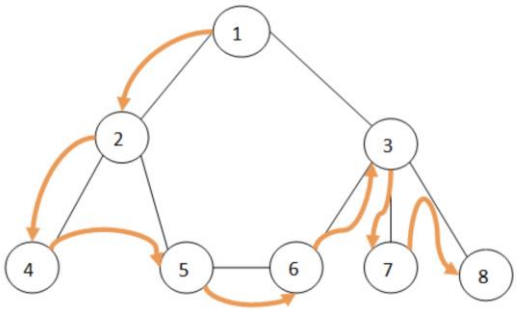
Gambar 2 Susunan kartu awal segitiga



Gambar 3 Susunan kartu awal sejajar

B. Algoritma Depth First Search(DFS)

Algoritma DFS adalah pencarian mendalam pada suatu pohon.



Gambar 4 Algoritma DFS (Sumber: [4])

Dari simpul pertama program akan mengunjungi salah satu simpul anak. Simpul berikutnya yang dikunjungi adalah simpul anak dari simpul anak tersebut jika ada. Jika tidak ada, yang dikunjungi adalah simpul tetangga dari simpul anak tersebut.

Pada DFS dapat terjadi *backtracking* yaitu kembali ke simpul sebelumnya yang masih memiliki anak yang belum dikunjungi. Ini dapat dikaitkan ke sistem permainan solitaire yang memperbolehkan aksi *undo* sehingga algoritma DFS adalah algoritma yang cocok untuk menyelesaikan solitaire.

III. PENERAPAN ALGORITMA

Pada makalah ini penerapan set kartu yang dilakukan adalah set kartu yang memiliki warna sama. Susunan awal kartu juga dibuat sama semua di awal. Jumlah baris set dan kartu yang tertutup dapat didefinisikan di awal.

Algoritma dibuat dengan menyimpan state-state boardgame. State awal didefinisikan terlebih dahulu lalu state berikutnya akan dihasilkan dari state awal.

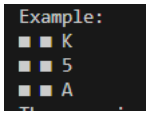
```
class BoardGame {
    private Card[][] board;
    private int m;
    private int n;
    private int sets;
    private Stocks stocks;
    private Map<String, Integer> dict;
    private int card_count;
```

Gambar 5 Struktur kelas BoardGame

Kelas BoardGame menyimpan informasi tentang tableau yang akan diperlukan dalam permainan.

1. Card[][] board

   Atribut ini menyimpan matriks kartu berukuran m x n yang akan didefinisikan pada konstruktor. Untuk mempermudah struktur data, barisan kartu dibuat menyamping.

   Gambar 6 Contoh tampilan kartu

2. int m

   Atribut ini menyimpan jumlah baris yang mungkin diisi barisan kartu. Nilai m dapat dicustom sesuai keinginan pengguna.

3. int n

   Atribut ini menyimpan jumlah kolom yang valid yaitu jumlah kartu pada barisan kartu. Jumlah yang didefinisikan adalah 13 kali jumlah set kartu karena ini merupakan jumlah maksimal yang mungkin terjadi.

4. int sets

   Atribut ini menyimpan nilai set kartu yang ada. Jadi, permainan akan berakhir ketika sudah terjadi kartu berurutan sebanyak sets.

5. Stocks stocks

   Atribut ini menyimpan stok kartu yang belum terbuka. Atribut ini disimpan dalam bentuk stocks yang berisi

list l yaitu list semua elemen yang sudah dirandom dan Map stocks yang berisi jumlah tiap jenis kartu di stoks.

```
class Stocks {
    private ArrayList<String> l;
    private Map<String,Integer> stocks;
```

6. Map dict

   Atribut ini berisi konversi nilai string kartu seperti “K” dan “10” menjadi nilainya. Misalnya “K” akan dikonversi menjadi 13.

7. int card_count

   Atribut ini berisi jumlah kartu yang sudah tersusun.

```
class DFS {
    private BoardGame board;
    private Map<Action, Boolean> acts;
```

Gambar 7 Struktur kelas DFS

Ketika mendefinisikan DFS, aksi-aksi yang mungkin akan diinisialisasi secara langsung dan disimpan di atribut acts. Aksi-aksi yang mungkina adalah memindahkan kartu atau membuka stock baru.

```
public DFS(BoardGame board, ArrayList<DFS> s){
    this.board = board;
    this.acts = new HashMap<Action, Boolean>();
    generateActs(s);
}
```

Gambar 8 Konstruktor DFS

Aksi-aksi ini kemudian akan dijalankan dan menghasilkan state DFS baru dengan menggunakan fungsi generate yang merupakan fungsi rekursif. Kunjungan state akan disimpan di acts yang berbentuk map. Jika state sudah dikunjungi maka akan ditandai true pada map acts.

```
public void generate(ArrayList<DFS> s, ArrayList<BoardGame> path){
    System.out.println(this.acts.size());
    path.add(this.board);
    Map<Action, Boolean> acts = this.acts;
    boolean found_acts = false;
    Iterator<Entry<Action, Boolean> > new_Iterator = acts.entrySet().iterator();
    // testing
    Set<Action> a = getActs().keySet();
    for(Action el: a){
        System.out.println(el.getType());
        if (el.getType() == "move"){
            System.out.println(el.getInit()[0] + ", " + el.getInit()[1] + ", " + el.getFin());
        }
    }
    while(new_Iterator.hasNext() && !found_acts){
        try{
            Thread.sleep(millis:1000);
        }
        catch(InterruptedException e){
            System.out.println(x:"Interrupted");
        }
        Map.Entry<Action, Boolean> new_Map = (Map.Entry<Action, Boolean>) new_Iterator.next();
        if ((!new_Map.getValue()) && new_Map.getKey().getType() == "move"){ // not yet executed action
            found_acts = true;
            Action key = new_Map.getKey();
            this.acts.put(key, value:true); // set true
            // do action
            BoardGame generated = this.board.doAct(key);
            DFS newdfs = new DFS(generated, s);
            if (!s.contains(newdfs)){
                generated.printBoard();
                newdfs.generate(s, path);
                s.add(newdfs);
            }
        }
    }
}
```

Gambar 9 Fungsi utama DFS

Terdapat dua list global yaitu history dan path. history adalah list untuk menampung semua state yang akan didefinisikan. path adalah list yang digunakan untuk menampung state board yang dilalui selama pengecekan.

Pada fungsi generate, pertama-tama state board saat ini ditambahkan ke list path. Kemudian dilakukan iterasi rekursif terhadap acts yang mungkin dilakukan. Tiap elemen acts akan dijalankan ke board sekarang dan dibentuk state DFS baru berdasarkan hasil aksi tersebut. DFS yang baru akan dijalankan pada looping elemen list acts tersebut sehingga pengecekan akan dilakukan secara mendalam.

Looping aksi dapat terjadi apabila tidak dilakukan pengecekan state yang akan degenerate dari state sebelumnya. Misalnya pada baris pertama terdapat urutan x x 7 6 5 dan pada baris kedua terdapat urutan x x 6. Aksi yang dapat dilakukan adalah pemindahan angka 5 ke baris kedua. Namun, setelah pemindahan terjadi, program juga akan mendeteksi pemindahan angka 5 ke baris pertama kembali. Untuk itu, sebelum penambahan state ke list history, dilakukan pengecekan state terlebih dahulu. Jika state hasil aksi sudah terdapat pada list history, state tersebut tidak akan ditambahkan dan tidak di-generate lebih lanjut.

```
if (!history.contains(newdfs)){
    generated.printBoard();
    newdfs.generate(history, path);
    history.add(newdfs);
    generated.printBoard();
}
```

Gambar 10 Pencegahan looping aksi

Untuk men-generate aksi yang mungkin pada suatu state, digunakan fungsi generateActs yang digunakan pada konstruktor DFS. Fungsi generateActs memanggil fungsi

getMovable yaitu fungsi yang menghasilkan indeks-indeks yang dapat diangkat dari barisan.

```java
public ArrayList<int[]> getMovable(){
    Map<String, Integer> dict = this.board.getDict();
    Card[][] board = this.board.getBoard();
    ArrayList<int[]> cons = new ArrayList<int[]>();
    boolean valid;
    int test;
    for(int i = 0; i < this.board.getM(); i++){
        for(int j = 0; j < this.board.getLastIdxRow(i)+1; j++){
            if (board[i][j].getOpen()){
                test = j;
                valid = true;
                while (test < this.board.getLastIdxRow(i) && valid){
                    if (dict.get(board[i][test].getName()) != (dict.get(board[i][test+1].getName()) + 1)){
                        valid = false;
                    }
                    else{
                        test++;
                    }
                }
                if (valid){
                    cons.add(new int[] {i,j});
                }
            }
        }
    }
    Collections.sort(cons, new intComparator());
    return cons;
}
```

Gambar 11 Fungsi untuk mencari indeks-indeks kartu yang dapat dipindahkan

Karena pemindahan kartu memiliki aturan tertentu, pengecekan pengangkatan kartu harus dilakukan. Kartu yang dapat diangkat adalah kartu yang memiliki urutan nilai menurun pada kartu dibawahnya. Misalnya “6 5 4 3” dapat diangkat, namun “6 5 4 7” tidak dapat diangkat.

Pengecekan dilakukan dengan cara melakukan perulangan pada tiap baris lalu melakukan perulangan juga pada tiap indeks yang terbuka (kartu memiliki atribut terbuka dan tidak) apakah kartu di belakangnya terurut menurun dengan selisih tepat 1 nilai atau tidak.

```java
public void generateActs(ArrayList<DFS> s){
    Card[][] board = this.board.getBoard();
    Map <String, Integer> dict = this.board.getDict();
    ArrayList<int[]> movable = getMovable();
    for(int[] el: movable){
        for(int i = 0; i < this.board.getM(); i++){
            if (dict.get(board[i][this.board.getLastIdxRow(i)].getName()) == (dict.get(board[el[0]][el[1]].getName()) + 1)){
                Action act = new Action(type:"move", el, i);
                if (!containsAct(s, act)){
                    this.acts.put(act, value:false);
                }
            }
        }
    }
    if (!this.board.getStocks().isEmpty()){
        this.acts.put(new Action(type:"open"), value:false);
    }
}
```

Gambar 12 Fungsi untuk men-generate aksi yang mungkin

Untuk mengenerate aksi yang mungkin, dipanggil fungsi getMovable yang berisi list indeks yang mungkin dipindahkan. Untuk setiap indeks, dicek apakah ada baris yang dapat menerima indeks barisan kartu tersebut. Jika ada, maka action move dengan indeks dan baris bersangkutan akan ditambahkan. Setelah semua kemungkinan dicari, dilakukan pengecekan stok apakah masih ada atau tidak. Jika stok masih ada, maka akan ditambahkan juga action open untuk membuka kartu-kartu baru.

```java
public BoardGame doAct(Action act){
    BoardGame boardnew = new BoardGame(this.m, this.sets);
    boardnew.board = this.board;
    boardnew.n = this.n;
    boardnew.stocks = this.stocks;
    boardnew.dict = this.dict;
    boardnew.card_count = this.card_count;
    if (act.getType() == "open"){
        boardnew.open();
    }
    else{
        boardnew.move(act.getInit()[0], act.getInit()[1], act.getFin());
    }
    return boardnew;
}
```

Gambar 13 Fungsi untuk melakukan aksi pada board

Setiap state akan digenerate dengan melakukan aksi pada sebuah board. Terdapat fungsi doAct untuk melakukan aksi pada board dan menghasilkan board baru dari aksi tersebut. Pertama-tama dibuat sebuah board baru yang sama dengan board lama. Kemudian tentukan aksi apakah berbentuk move atau open. Lalu lakukan aksi sesuai dengan tipe aksi yang dicek sebelumnya.

```java
public void move(int i, int j, int in){
    if (validMove(i, j) && validPut(i,j,in)){
        for (int jj = j; jj < getLastIdxRow(i)+1; jj++){
            this.board[in][getLastIdxRow(in)+1].setName(this.board[i][jj].getName());
            this.board[in][getLastIdxRow(in)].setOpen(open:true);
            this.board[i][jj].setName(name:"x");
            this.board[i][jj].setOpen(open:false);
        }
        if (j > 0){
            this.board[i][j-1].setOpen(open:true);
        }
        System.out.println("moved from " + i + ", " + j + " to " + in);
    }
}
```

Gambar 14 Fungsi untuk memindahkan kartu

Untuk memindahkan kartu digunakan fungsi move dengan argumen i, j dan in yaitu (i, j) adalah indeks yang akan dipindahkan dan in adalah baris tempat barisan kartu akan dipindahkan.

```java
public void open(){
    if (!this.stocks.isEmpty()){
        int i = 0;
        while(!this.stocks.isEmpty() && i < m){
            String name = this.stocks.take();
            this.board[i][getLastIdxRow(i)+1].setName(name);
            this.board[i][getLastIdxRow(i)].setOpen(open:true);
            i++;
        }
        System.out.println(x:"opened");
    }
}
```

Gambar 15 Fungsi untuk membuka kartu-kartu baru

Untuk membuka kartu digunakan fungsi open. Fungsi ini akan menambahkan kartu secara otomatis dimulai dari indeks ke nol apabila stok kartu masih tersedia.

IV. TESTING

Berikut contoh pengecekan dengan fungsi main. Nilai baris terdefinisi adalah 3, set 1, jumlah kolom kartu 3 dan jumlah kolom tertutup 2. Kartu pada baris pertama kolom ketiga dibuat "J" dan pada baris kedua kolom ketiga dibuat "10" agar ada aksi move yang dapat dilakukan. Kemudian dibentuk dfs dengan board tersebut dan history awal kosong. Akan ditampilkan nilai-nilai indeks yang dapat diangkat dan aksi yang dapat dilakukan.

```
public static void main(String args[]){
    int rows = 3;
    int sets = 1;
    int nCardCol = 3;
    int nCloseCol = 2;
    BoardGame b = new BoardGame(rows, sets);
    b.initialize(nCardCol, nCloseCol);
    b.setIdx(i:0,j:2, new Card(name:"J"));
    b.setIdx(i:1,j:2, new Card(name:"10"));
    b.setIdxOopen(i:0, j:2);
    b.setIdxOopen(i:1, j:2);
    b.printBoard();
    DFS dfs = new DFS(b, new ArrayList<DFS>());

    System.out.println(x:"movable: ");
    ArrayList<int[]> movable = dfs.getMovable();
    for(int[] i: movable){
        System.out.println(i[0] + ", " + i[1]);
    }

    System.out.println(x:"acts: ");
    Set<Action> a = dfs.getActs().keySet();
    for(Action el: a){
        System.out.println(el.getType());
        if (el.getType() == "move"){
            System.out.println(el.getInit()[0] + ", " + el.getInit()[1] + " to row " + el.getFin());
        }
    }
}
```

Gambar 16 Testing 1

Didapatkan hasil sebagai berikut.

```
■ ■ J
■ ■ 10
■ ■ 9
movable:
0, 2
1, 2
2, 2
acts:
move
2, 2 to row 1
move
1, 2 to row 0
open
```

Gambar 17 Hasil testing 1

Indeks yang dapat dipindahkan adalah (0,2), (1,2) dan (2,2). Karena tidak ada kartu di sebelah kanannya, ketiga indeks tersebut pasti dapat dipindahkan.

Aksi-aksi yang dapat dilakukan adalah:

1. move (2,2) to row 1

   Kartu "10" dapat dipindahkan ke bawah kartu "J".

2. move (1,2) to row 0

   Kartu "9" dapat dipindahkan ke bawah kartu "10".

3. open

   Karena sisa stok kartu masih ada, masih bisa dilakukan open.

Akan dilakukan testing membuka kartu pada sebuah board.

```
public static void main(String args[]){
    BoardGame b = new BoardGame(m:3, sets:1);
    b.initialize(nCardCol:3, nCloseCol:2);
    b.printBoard();
    b.open();
    b.printBoard();
    b.open();
    b.printBoard();
}
```

Gambar 18 Testing 2

Didapatkan hasil sebagai berikut.

```
■ ■ J
■ ■ A
■ ■ 7
opened
■ ■ J 10
■ ■ A 5
■ ■ 7 9
opened
■ ■ J 10 Q
■ ■ A 5
■ ■ 7 9
```

Gambar 19 Hasil testing 2

Akan dilakukan testing memindahkan kartu.

```
public static void main(String args[]){
    BoardGame b = new BoardGame(m:3, sets:1);
    b.initialize(nCardCol:3, nCloseCol:2);
    System.out.println(x:"--Initial--");
    b.setIdx(i:0,j:2, new Card(name:"J"));
    b.setIdx(i:1,j:2, new Card(name:"10"));
    b.setIdxOopen(i:0, j:2);
    b.setIdxOopen(i:1, j:2);
    b.printBoard();
    b.move(i:1,j:2,in:0);
    b.printBoard();
}
```

Gambar 20 Testing 3

Didapatkan hasil sebagai berikut.

```
--Initial--
■ ■ J
■ ■ 10
■ ■ 5
moved from 1, 2 to 0
■ ■ J 10
■ 4
■ ■ 5
```

Gambar 21 Hasil testing 3

V. Link Github

Pengembangan lebih lanjut dilakukan pada link berikut https://github.com/manuellaiv/DFSolitaire.

VI. Ucapan Terima Kasih

Puji dan syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini. Penulis ingin menyampaikan terima kasih kepada dosen pengampu mata kuliah Strategi Algoritma yaitu Pak Rinaldi Munir dan dosen-dosen lainnya yang telah memberikan ilmunya dalam mata kuliah ini. Penulis berharap makalah ini dapat bermanfaat bagi pembaca.

VI. Referensi

[1] https://bicyclecards.com/how-to-play/solitaire

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm

[3] https://solitaired.com/spider-solitaire

[4] https://www.encora.com/insights/dfs-vs-bfs

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, 22 Mei 2023

Manuella Ivana Uli Sianipar 13521051