دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

# گزارش ۲ درس هوش مصنوعی

# پیاده‌سازی هیوریستیک برای حل یک مساله تصمیم گیری ومقایسه روش‌ها روی نمونه‌های تصادفی

به قلم:
امیر بابامحمودی

استاد
دکتر مهدی قطعی

فروردین ۱۴۰۰

## مقدمه:

در این گزارش قصد داریم تا نسخه ی ساده شده‌ای از بازی سولیتر (solitaire) را پیاده‌سازی کرده وبا کمک یک تابع هیورستیک و الگوریتم A* به ازای ورودی های مختلف سعی در حل و رسید به هدف کنیم .

## شرح بازی :

در این بازی نیاز است تعدادی کارت که رنگ و عدد هر کدام با بقیه فرق دارد مرتب شوند .

قوانین بازی :

زمین بازی k بخش دارد که در هر بخش بینهایت کارت میتواند قرار بگیرد و کارت ها در هر بخش طوری گذاشته می‌شوند که کل کارت ها دیده شوند .

این کارت ها در هر بازی یکی از m رنگ را داشته (m<=k) و از هر رنگ نیز n کارت با شماره های ۱ تا n داریم . در این بازی هدف این است که در انتها کارت های هر بخش همرنگ بوده و اعداد آن‌ها از بالا به پایین به صورت نزولی مرتب شده باشد .

در هر مرحله بازی فقط می‌شود از روی هر دسته یک کارت برداشته و آن را به ستون‌های بعدی منتقل کرد آن هم به این شرط که کارت تنها میتواند بروی کارتی که عددش از خودش بیشتر میباشد بشیند .(یا یک ستون خالی باشد)

مثالی از نحوه ی ورودی کد :

```
5 3 6
6r 5g 5r 4y
6y 2g 4r 3y 3g 2y
1y 4g 1r
6g 1g 2r 5y 3r
#
```

در مثال روبرو خط اول شامل اعداد k , m , n بوده که تعداد بخش‌ها (section) تعداد رنگ‌ها و شماره ی روی کارت ها را به ترتیب از چپ به راست نشان میدهد .

در k خط بعدی نشان میدهد در بخش شماره ی k چه کارت هایی به چه ترتیبی وجود دارند . برای مثال در بخش اول به ترتیب کارت ۶ با رنگ قرمز , کارت سبز با شماره ی ۵ , کارت قرمز باشماره ی ۵ و کارت زرد باشماره ی ۴ میباشد که یعنی در این مساله سه رنگ سبز , قرمز و زرد با شماره های ۱ تا 6 وجود دارند . علامت # در خط آخر نشان دهنده ی این میباشد که بخش ۵ ام خالی میباشد و کارتی در آن وجود ندارد .

خروجی مورد انتظار :

6g 5g 4g 3g 2g 1g
6y 5y 4y 3y 2y 1y
6r 5r 4r 3r 2r 1r
#
#

البته لازم به ذکر میباشد که این حالت با حالت

6g 5g 4g 3g 2g 1g
5y 5y 4y 3y 2y 1y
#
6r 5r 4r 3r 2r 1r
#

تفاوتی نداشته و میتواند هدف مساله باشد .

حال به شرح کد پیاده‌سازی شده که لینک گیت هاب آن در آخر گزارش آمده است میپردازیم:

این کد در زبان java نوشته شده و شامل ۵ کلاس Card , Section , Graph , State و کلاس Main میباشد . در زیر به شرح هر یک از این کلاس‌ها میپردازیم :

```java
public class Card {

    private int number;
    private char color;

    public Card(String str) {
        this.color = str.charAt(1);
        String s=Character.toString(color);
        String [] st = str.split(s);
        this.number = Integer.parseInt(st[0]);  ;

    }

    public int getNumber() { return number; }

    public char getColor() { return color; }

    @Override
    public String toString() { return "" + number + color + " "; }
```

Card : این کلاس مربوط به کارت های بازی شده که هر کارت شامل رنگ و شماره میباشد و از آنجایی که ما اطلاعات هر کارت رو بصورت ترکیب یه کاراکتر به عنوان رنگ و یک رقم بعنوان عدد رو کارت میگیریم با کمک توابع parseInt و charAt و toString این دو اطلاعات رو بصورت جدا میگیریم .

```java
public class Section {
    private ArrayList<Card> cards = new ArrayList<>();

    public Section(String row) {
        if(!(row.equals("#") || row.equals(""))){
            String[] arrOfStr = row.split( regex: " ", limit: 0);

            for (String a : arrOfStr)
                addCard(a);
        }
    }

    public void addCard(String a) { this.cards.add(new Card(a)); }

    public void print(){
        for (Card card : cards){
            System.out.print(card.toString());
        }
    }

    public String toString(){
```

```java
public Boolean isSort(){
    for (int i = 0; i<cards.size() ; i++){
        char c = cards.get(0).getColor();
        if (cards.get(i).getNumber() != Main.n - i){
            return false;
        }else {
            if (c != cards.get(i).getColor()){
                return false;
            }

        }
    }
    return true;
}

public ArrayList<Card> getCards() { return cards; }

public Card getTopCard() { return cards.get(cards.size()-1); }

public Card popTopCard(){
    Card card = cards.get(cards.size()-1);
    cards.remove(cards.get(cards.size()-1));
    return card;
```

Section : در این کلاس درواقع هر بخش که کارت در آن قرار میگیرد پیاده‌سازی شده که شامل آرایه ای از کارت ها بوده که به آن بخش اضافه می‌شوند که این کار توسط تابع addcard صورت میگیرد .در این بخش توابع دیگری برای چک کردن سورت بودن شماره ی کارت ها همینطور برای برداشتن بالاتری کارت نیز وجود دارد .

```java
import java.util.*;

public class State {

    private State parent;
    private ArrayList<State> neighbour = new ArrayList<>();
    private ArrayList<Section> sections = new ArrayList<>();
    private int setNeigh = 0;
    private int gCost;
    private int heuristic;
    private int fCost;

    public State(ArrayList<Section> sections, int cost) {

        this.sections = sections;
        this.gCost = cost;
        setHeuristic();
        this.fCost = gCost + heuristic;
    }

    public void print(){
        for (Section sec : sections){
            if(sec.getCards().size() == 0){
```

```java
    public Boolean isFinish(){
        for (Section sec : sections){
            if (!sec.isSort()){
                return false;
            }
        }
        return true;
    }

    public ArrayList<State> getNeighbour() {
        return neighbour;
    }

    public void setNeighbour() {
        if (setNeigh == 0) {
            setNeigh =1;
            for (int i = 0; i < sections.size(); i++) {
                if (sections.get(i).getCards().size() > 0) {
                    int x = sections.get(i).getTopCard().getNumber();
                    outer:
                    for (int j = 0; j < sections.size(); j++) {
                        int y = 0;
                        if (sections.get(j).getCards().size() > 0) {
```

```java
                        int y = 0;
                        if (sections.get(j).getCards().size() > 0) {
                            y = sections.get(j).getTopCard().getNumber();
                        }
                        if (y == 0 || x < y) {
                            ArrayList<Section> newSections = new ArrayList<>();
                            for (Section sec : sections) {
                                Section newSec = new Section(sec.toString());
                                newSections.add(newSec);
                            }

                            Card card = newSections.get(i).popTopCard();
                            newSections.get(j).addCardToTop(card);
                            State s = new State(newSections, cost: this.getGCost() + 1);
                            for (State state : neighbour){
                                if (state.equals(s)){
                                    continue outer;
                                }
                            }
                            neighbour.add(s);
                            s.addNeighbour( s: this);
                        }
```

State: در این بخش استیت کلی بخش‌ها و کارت های داخل آن‌ها پیاده‌سازی شده که شامل آرایه ای برای قرار دادن همسایه ها یا به بیانی بهتر تمامی استیت هایی که میتوان از یک استیت مذکور به انجام یک action به آن‌ها رفت وجود دارد .همینطور لیستی برای نشان دادن بخش‌های یک استیت . تابع isFinish در این بخش برای بررسی اینکه به حالت هدف رسیدیم یا نه پیاده‌سازی شده است .حال به تابع setHeuristic میپردازیم .

```java
public void setHeuristic() {

    heuristic = 0;

    for (int i = 0; i < sections.size(); i++) {
        HashSet<Character> colors = new HashSet<>();

        if (sections.get(i).getCards().size() > 1) {
            for (int j = 0; j < sections.get(i).getCards().size(); j++) {
                colors.add(sections.get(i).getCards().get(j).getColor());
            }

            heuristic += colors.size()-1;

            for (int j = 1; j < sections.get(i).getCards().size(); j++) {
                if(sections.get(i).getCards().get(j).getNumber() >= sections.get(i).getCards().get(j-1).getNumber()){
                    heuristic += 1;
                }
            }

        }else if(sections.get(i).getCards().size() == 1){
            if (Main.m > 1){
                heuristic += 1;
```

## تابع هیوریستیک :

تابع هیوریستکی که برای این مسأله انتخاب شده درواقع شامل سه بخش است که برای هر استیت محاسبه شده و در نهایت جمع زده شده و به عنوان h(s) برای آن استیت تلقی می‌شود . بخش اول تعداد رنگ‌های متمایز هر بخش را شماره و یک واحد از آن کم کرده (زیرا در هر بخش یک رنگ یکسان میتواند قرار بگیرد ) و سپس عدد بدست آمده ی هر section را جمع میکند . در بخش دوم این تابع در هر یک از سکشن ها شماره ی کارت های متوالی را بررسی کرده و هرجا که صعودی بود یک واحد اضافه میکند به مقدار هیورستیک . در بخش سومم بررسی میکند اگر کارتی به تنهایی در یک سکشن وجود داشته باشد اگر n درواقع بیشتر از یک بود نشان میدهد جای آن اشتباه بوده و یک واحد به هیورستیک اضافه میکند .

```
public void AStar(State s) {
    frontier.add(s);

    while (!frontier.isEmpty()){
        State current;
        current = smallest(frontier);
          current.print();
          System.out.println("current g :" + current.getGCost() );
          System.out.println("current h :" + current.getHeuristic());
          System.out.println("current f :" + current.getFCost());
        visitedNodes.add(current);
        bast ++;
        if (current.isFinish()) {
            System.out.println("depth = " + current.getGCost());
            System.out.println("explored = " + visitedNodes.size());
            System.out.println("frontier = " + toolid);

            ArrayList<State> path = new ArrayList<>();

            System.out.println("path from root to goal");
            while (current != null){
                path.add(current);
```

```
            while (current != null){
                path.add(current);
                current = current.getParent();
            }
            Collections.reverse(path);
            for (State parent: path) {
                System.out.println("----------------------");
                parent.print();
            }
            return;
        }
        frontier.remove(current);
        current.setNeighbour();
        for (State state : current.getNeighbour()){
            if (!visitedNodes.contains(state)){
                state.setParent(current);
                frontier.add(state);
                toolid ++;
            }
        }
    }
```

Graph:در این بخش درواقع الگوریتم A* اجرا شده و هر بار تابع isFinished رو برای بررسی رسیدن به هدف صدا میکند

```java
import java.util.ArrayList;
import java.util.Scanner;
import java.lang.*;

public class Main {
    static int k, m, n;
    public static void main(String[] args) {
        long startTime = System.nanoTime();

        Scanner input = new Scanner(System.in);

        k = input.nextInt();
        m = input.nextInt();
        n = input.nextInt();
        input.nextLine();

        String row;

        ArrayList<Section> secRoot = new ArrayList<>();

        for(int i=0; i<k; i++){
            row  = input.nextLine();
            secRoot.add(new Section(row));
```

```java
            row  = input.nextLine();
            secRoot.add(new Section(row));
        }
        State root = new State(secRoot,  cost: 0);
        root.setParent(null);

        Graph graph = new Graph(root);


        graph.AStar(root);
        long endTime   = System.nanoTime();
        long totalTime = (endTime - startTime) / 1000000;
        System.out.print("total search time is : ");
        System.out.println(totalTime);
    }
```

Main: در این بخش مقادیر ورودی گرفته شده با شروع الگوریتم A* از ریشه شروع به سرچ کرده و همینطور برای بدست آوردن تایم ران شدن برنامه از System.nanoTime استفاده شده است .

چند مثال از ورودی و خروجی های آن :

۱ ــ ورودی :

```
3 1 3
3g
1g
2g
```

خروجی :

```
depth = 2
explored = 8
frontier = 13
path from root to goal
----------------------
 3g
 1g
 2g

----------------------
 3g 2g
 1g
#

----------------------
 3g 2g 1g
#
#

total search time is : 1133
```

۲-ورودی :

```
3 2 3
1r 1g
3r 2r
3g 2g
```

خروجی :

```
depth = 2
explored = 3
frontier = 4
path from root to goal
----------
 1r 1g
 3r 2r
 3g 2g

----------
 1r
 3r 2r
 3g 2g 1g

----------
#
 3r 2r 1r
 3g 2g 1g

total search time is : 15882
```

۳-ورودی :

```
5 3 5
5g 4g 3g 2g
1g 1r
5r 4r 3r
5b 4b 3b 2b 2r
1b
```

خروجی :

depth = 4
explored = 8
frontier = 45
path from root to goal
---
 5g 4g 3g 2g
 1g 1r
 5r 4r 3r
 5b 4b 3b 2b 2r
 1b

---
 5g 4g 3g 2g
 1g 1r
 5r 4r 3r 2r
 5b 4b 3b 2b
 1b

---
 5g 4g 3g 2g
 1g
 5r 4r 3r 2r 1r
 5b 4b 3b 2b
 1b

---
 5g 4g 3g 2g 1g
#
 5r 4r 3r 2r 1r
 5b 4b 3b 2b
 1b

---
 5g 4g 3g 2g 1g
#
 5r 4r 3r 2r 1r
 5b 4b 3b 2b 1b
#

total search time is : 53426

منابع :

1 ـ برای انتخاب مساله و پیدا کردن یک تابع هیورستیک مناسب در این گزارش با دو تا از دوستانم امیررضا رادجو با شماره دانشجویی ۹۷۱۳۰۱۸ و همچنین نیکا شهابی با شماره دانشجوری ۹۷۱۳۰۲۳ مشورت کردم

2- https://stackoverflow.com/questions/4845737/face-up-solitaire-algorithm

3- https://web.stanford.edu/~bvr/pubs/solitaire.pdf

4- https://github.com/andavies/n-puzzle

5- https://en.wikipedia.org/wiki/Solitaire

لینک گیت هاب برای دسترسی به کد:
https://github.com/amirbabamahmoudi/simple-version-of-solitaire-with-A-.git