دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش ۲ درس هوش مصنوعی

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

مولف:
امیررضا رادجو

استاد
دکتر مهدی قطعی

فروردین ۱۴۰۰

توضیح بازی:

بازی شامل k ستون که در هر ستون میتوان بی نهایت کارت با یکی از m رنگ موجود که شماره ی هر کارت از ۱ تا n میباشد شروع شده .
تعداد ستون‌ها همواره بیشتر یا برابر با رنگ‌های متمایز موجود میباشد .
هدف این بازی این است که در نهایت کارت ها با رنگ مشابه و به سورت نزولی از بالا به پایین به ترتیب در هر ستون قرار بگیرند .

قوانین بازی :
در هر مرحله از بازی میتوان یک کارت را از یک ستون برداشته و به ستون‌های دیگر منتقل کرد اما کارت برداشته شده نمیتواند بر روی کارتی کوچک‌تر از خود نشسته و همچنین کارت را میتوان در ستونی خالی در صورت وجود قرار داد .
برای مثال :

ورودی مانند روبرو به کد داده شده :

```
4 2 4
4g 3g 1g
3r 4r 2r
2g 4r
#
```

در مثال روبرو # نشان دهنده ی ستون خالی و اعداد سطر اول به ترتیب نشان دهنده ی تعداد ستون‌ها تعداد رنگ‌های متمایز و شماره کارت ها بوده و در k خط بعدی کارت های موجود در ستون‌ها آمده‌اند . برای مثال در ستون سوم کارت سبز با شماره ی ۲ و کارت قرمز با شماره ی قرمز وجود دارد .

```python
class Node:
    def __init__(self, table, parent):
        self.table = table
        self.parent = parent
        if parent == None:
            self.depth = 0
        else:
            self.depth = parent.depth + 1

def goal_check(t, n, m, k):
    check = True
    for i in range(k):
        if len(t[i]) == n or len(t[i]) == 0:
            for j in range(len(t[i]) - 1):
                if t[i][j][1] == t[i][j+1][1] and int(t[i][j][0]) > int(t[i][j+1][0]):
                    continue
                else:
                    check = False
                    break
        else:
            check = False
            break
    return check

# making the new node by giving the parent and the movement
def make_node(node, k, l):
    new = []
    for i in range(len(node.table)):
        new.append([])
        for j in range(len(node.table[i])):
```

همانگونه که مشاهده می‌شود ابتدا کلاس Node ساخته شده که همان درواقع است و state ها میباشند و سپس تابعی

برای تشخیص اینکه آیا یک state هدف نهایی ما است یا خیر پیاده‌سازی شده است (goal-check) سپس تابع make_node مشاهده می‌شود که طبق جابجایی که باید برای یک کارت از یک ستون به ستونی دیگه اتفاق افتد را انجام میدهد.

```
# getting inputs
n, m, k = input().split()
# number
n = int(n)
# colors
m = int(m)
# rows
k = int(k)
table = []
frontier = []
explored = []
answer = []
found = False
total = 0

# heuristic function
def heuristic(t):
    counter = 0
    for i in range(k):
        if len(t[i]) > 0:
            if int(t[i][0][0]) == n:
                counter += 1
                for j in range(len(t[i]) - 1):
                    if int(t[i][j][0]) > int(t[i][j+1][0]) and t[i][j][1] == t[i][j+1][1]:
                        counter += 1
                    else:
                        break
            else:
                continue
        else:
            counter += 1
    return (m*n + (k - m)) - counter

def find_f(node):
    return node.depth + heuristic(node.table)
```

در اینجا ابتدا ورودی های مساله گرفته شده سپس تابع heuristic را به شرح زیر پیاده‌سازی میکنیم :
ابتدا تعداد کارت هایی که میتواند در جای درست قرار گرفته باشد را میشمارد.(ردیف های خالی هم امتیاز مثبت حساب میشوند) به این صورت که اولین کارت شماره تعداد کارت های هر رنگ باشد و همینطور کارت های بعدی نزولی باشد . سپس امتیاز بدست آمده از امتیاز هدف کم می‌شود و به عنوان h داده می‌شود .
تابع find-f برای حساب کردن هزینه کلی شامل مجموع هزینه از استیت قبلی به این استیت و مقدار هیوریستیک استیت مقصد میباشد .

```
def best_node(list):
    min = list[0]
    for i in range(len(list)):
        if find_f(list[i]) < find_f(min):
            min = list[i]
    return min

# setting the table
for i in range(k):
    x = input()
    if x == '#':
        table.append([])
    else:
        table.append(x.split())

# checking if the table is our goal
```

تابع best-node استیت با کمترین مقدار f را برمیگرداند . سپس یک چرخش در frontier انجام داده و هربار چک میکند که آیا به هدف رسیدیم یا خیر .

```
        # checking if the row is empty to move in
        if len(frontier[index].table[j]) == 0:
            temp_table = make_node(frontier[index], i ,j)
            # checking if the node is in frontier or explored
            if temp_table.table not in [n.table for n in explored] and temp_table.table not in [n.table for n in frontier]:
                total += 1
                if goal_check(temp_table.table, n, m, k):
                    answer = copy_node(temp_table)
                    found = True
                    break
                else:
                    frontier.append(copy_node(temp_table))
            else:
                continue
        # checking if the rows last card is grater than the card to move in
        elif int(frontier[index].table[i][-1][0]) < int(frontier[index].table[j][-1][0]):
            temp_table = make_node(frontier[index], i ,j)
            # checking if the node is in frontier or explored
            if temp_table.table not in [n.table for n in explored] and temp_table.table not in [n.table for n in frontier]:
                total += 1
                if goal_check(temp_table.table, n, m, k):
                    answer = copy_node(temp_table)
                    found = True
                    break
                else:
                    frontier.append(copy_node(temp_table))
            else:
                continue
        else:
            continue
    if found:
        break
# adding node to explored list and removing from frontier
explored.append(copy_node(frontier[index]))
del frontier[index]
```

```
#printing answer
if found :
    path = []
    depth= answer.depth
    for i in range(depth + 1):
        path.append(answer.table)
        answer = answer.parent

    for i in range(depth+1):
        print(path.pop())
        print('========>')

    print('depth = ', depth, ', explored : ', len(explored), ', total nodes created : ', total)
else :
    print('answer not found !!!')
```

در اینجا در ادامه طبق شرایط مساله بررسی میکند که آیا امکان جابجایی یک کارت از یک ستون به ستون دیگر وجود دارد یا نه که در صورت وجود تابع make-node صدا زده می‌شود . در نهایت مسیر جواب هدف رو با کمک آرایه path چاپ کرده و مقادیر قابل بررس که در ادامه به آن‌ها میپردازیم را خروجی میدهد .

ورودی ۱) :

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

```
3 2 3
3g 2g 1r
3r 2r 1g
#
[['3g', '2g', '1r'], ['3r', '2r', '1g'], []]
========>
[['3g', '2g'], ['3r', '2r', '1g'], ['1r']]
========>
[['3g', '2g', '1g'], ['3r', '2r'], ['1r']]
========>
[['3g', '2g', '1g'], ['3r', '2r', '1r'], []]
========>
depth =  3 , explored :  4 , total nodes created :  6
total search time :  15794.69
```

ورودی ۲):

```
4 2 4
4g 3g
4r 2g
3r 2r
1r 1g
```

```
4 2 4
4g 3g
4r 2g
3r 2r
1r 1g
[['4g', '3g'], ['4r', '2g'], ['3r', '2r'], ['1r', '1g']]
========>
[['4g', '3g', '2g'], ['4r'], ['3r', '2r'], ['1r', '1g']]
========>
[['4g', '3g', '2g', '1g'], ['4r'], ['3r', '2r'], ['1r']]
========>
[['4g', '3g', '2g', '1g'], ['4r', '1r'], ['3r', '2r'], []]
========>
[['4g', '3g', '2g', '1g'], ['4r', '1r'], ['3r'], ['2r']]
========>
[['4g', '3g', '2g', '1g'], ['4r'], ['3r'], ['2r', '1r']]
========>
```

خروجی ۳):

```
5 3 5
5d 4d 3d 2d
5a 4a 2b
5b 4b 3b 1d
1a 2a
3a 1b
[['5d', '4d', '3d', '2d'], ['5a', '4a', '2b'], ['5b', '4b', '3b', '1d'], ['3a', '1b'], ['1a', '2a']]
========>
[['5d', '4d', '3d', '2d', '1d'], ['5a', '4a', '2b'], ['5b', '4b', '3b'], ['3a', '1b'], ['1a', '2a']]
========>
[['5d', '4d', '3d', '2d', '1d'], ['5a', '4a'], ['5b', '4b', '3b', '2b'], ['3a', '1b'], ['1a', '2a']]
========>
[['5d', '4d', '3d', '2d', '1d'], ['5a', '4a'], ['5b', '4b', '3b', '2b', '1b'], ['3a'], ['1a', '2a']]
========>
[['5d', '4d', '3d', '2d', '1d'], ['5a', '4a', '3a'], ['5b', '4b', '3b', '2b', '1b'], [], ['1a', '2a']]
========>
[['5d', '4d', '3d', '2d', '1d'], ['5a', '4a', '3a', '2a'], ['5b', '4b', '3b', '2b', '1b'], [], ['1a']]
========>
[['5d', '4d', '3d', '2d', '1d'], ['5a', '4a', '3a', '2a', '1a'], ['5b', '4b', '3b', '2b', '1b'], [], []]
========>
depth =  6 , explored :  7 , total nodes created :  43
total search time :  79765.63
```

فایل اصلی کد پایتون پیوست همین گزارش آمده است .

منابع:

در پیدا کردن مسأله و همینطور بررسی تابع هیورستیک با امیر بابامحمودی با شماره داشنجویی ۹۷۱۳۰۰۶ مشورت و همفکری صورت گرفت .

/https://bicyclecards.com/how-to-play/solitaire
Aaai.org

Ai.dmi.unibas.ch

Github.com/EthanWelsh

https://www.google.com/amp/s/fa.actince.com/klondike-solitaire-%25D9%2582%25D9%2588%25D8%25A7%25D9%2586%25DB%258C%25D9%2586-%25D8%25A8%25D8%25A7%25D8%25B2%25DB%258C-/%25DA%25A9%25D8%25A7%25D8%25B1%25D8%25AA