«به نام خدا»

گزارش کار تمرین کامپیوتری 1

محمد جواد بشارتی

شماره دانشجویی : 810199386

نحوه مدل کردن مسأله :

```
5   class Vertex :
6       def __init__(self, number, includes_devotee) :
7           self.number = number
8           self.includes_devotee = includes_devotee
9           self.includes_recipe = False
```

برای هر رأس تعریف بالا آورده شده و در هر رأس شماره آن، این که شامل مرید است یا خیر و اینکه شامل دستور پخت است یا خیر مشخص است.

برای مدلسازی گراف از لسیتی که شامل اشیاء رئوس است و یک لیست مجاورت برای رئوس همسایه استفاده شده است :

```
25  adj_vertexes = [[] for i in range(n + 1)]
26
27  for i in range(1, m + 1) :
28      u, v = map(int, input_lines[i].split(' '))
29      adj_vertexes[u].append(v)
30      adj_vertexes[v].append(u)
31
32  vertexes = [Vertex(i, False) for i in range(n + 1)]
```

برای مدلسازی رئوس صعب العبور، شماره رئوس صعب العبور در یک لیست نگه داری میشود :

```
34  h = int(input_lines[m + 1])
35  line = input_lines[m + 2].split(' ')
36  impassable_vertexes = []
37
38  for i in range(h) :
39      impassable_vertexes.append(int(line[i]))
```

برای مدلسازی دستور پخت ها و دستور پخت های مورد نیاز هر مرید به ترتیب از یک لیست که شامل شماره رئوس دارای دستور پخت است و یک دیکشنری از مرید به دستور پخت های مورد نیازش استفاده شده است. ضمنا هرگاه یک شماره دستور پخت یا مرید بخوانیم، در رأس با همان شماره مشخص میکنیم که این رأس دارای دستور پخت یا مرید است :

```
43  for i in range(m + 4, m + 4 + s) :
44      line = input_lines[i].split(' ')
45      p = int(line[0])
46      q = int(line[1])
47
48      vertexes[p].includes_devotee = True
49      recipe = []
50      for i in range(q) :
51          recipe.append(int(line[i + 2]))
52      required_recipes[p] = recipe
53
54      for i in range(q) :
55          if recipe[i] not in recipes :
56              recipes.append(recipe[i])
57          if vertexes[recipe[i]].includes_recipe != True :
58              vertexes[recipe[i]].includes_recipe = True
```

تعریف state ها : هر استیت شامل شماره رأسی که روی آن هستیم، یک لیست از دستور پخت هایی که در مسیرمان از مبدأ تا رأس فعلی دیده ایم و یک لیست از مرید هایی که در این مسیر راضی شده اند، است. پس استیت اولیه شامل مکان اولیه سید و دو لیست خالی میشود و زمانی به استیت نهایی میرسیم که تمامی مرید ها راضی باشند.

توضیح الگوریتم ها :

- BFS : ابتدا در صف مان مکان اولیه سید، زمان سپری شده 1، سه لیست خالی که به ترتیب نماینده مرید های راضی شده، مسیر و دستور پخت های دیده شده هستند و دیکشنری مرید ها به دستور پخت های مورد نیازشان را پوش میکنیم. در ادامه تا زمانی که صف خالی نشود کار های زیر را انجام میدهیم :
عنصر ابتدای صف را پاپ میکنیم و اطلاعاتش را میخوانیم. سپس بررسی میکنیم که رأسی که در استیت فعلی هستیم، صعب العبور هست یا خیر. اگر صعب العبور بود، تعداد تکرار های آن در مسیرمان از مبدأ به این رأس را پیدا میکنیم و بررسی میکنیم چه قدر در این رأس مانده ایم اگر زمانی که در این رأس مانده بودیم بیشتر یا مساوی تعداد تکرار ها بود، به ادامه الگوریتم میرویم. در غیر این صورت زمان سپری شده را یک واحد زیاد کرده و شماره رأس فعلی، زمان سپری شده ، مرید های راضی شده، مسیر، دستور پخت های دیده شده و دیکشنری مرید ها به دستور پخت های مورد نیازشان را در صف پوش میکنیم و به استیت بعدی میرویم (عنصر بعدی صف را پاپ میکنیم). در ادامه بررسی میکنیم که در رأسی که در استیت فعلی هستیم، دستور پختی وجود دارد یا خیر. اگر دستور پخت وجود داشت و این دستور پخت را قبلا ندیده بودیم، دستور پخت را به دستور پخت های دیده شده اضافه میکنیم سپس به دیکشنری مرید به لیست دستور پخت های مورد نیازش رفته و دستور پخت دیده شده اگر در لیست دستور پخت های مورد نیاز مریدی وجود داشت ، آن را از لیست دستور پخت های مورد نیازش حذف میکنیم. سپس بررسی میکنیم که رأس مان شامل مرید میشود یا خیر اگر مریدی وجود داشت که قبلا راضی نشده بود و لیست دستور پخت های مورد نیازش خالی شده بود، این مرید و لیست خالی شده اش را از دیکشنری مرید به لیست دستور پخت های مورد نیاز پاک میکنیم و شماره رأس را به لیست مرید های راضی شده اضافه میکنیم. سپس سراغ رئوس مجاور رأس فعلی میرویم اگر رأس همسایه در استیت های دیده شده نبود، ابتدا تست گل را انجام میدهیم (بررسی میکنیم ک مرید دارد یا خیر اگر مرید داشت و فقط یک مرید ناراضی داشتیم که همین مرید بود و تمام دستور پخت ها را هم دیده بودیم یعنی به گل رسیده ایم) اگر به گل رسیده بودیم، رأس همسایه را به مسیر اضافه کرده و مسیر را برمیگردانیم و در غیر این صورت در صف مان شماره رأس همسایه، زمان سپری شده ، مرید های راضی شده، مسیر، دستور پخت های دیده شده و دیکشنری مرید ها به دستور پخت های مورد نیازشان را پوش میکنیم. که در صفحه بعدی تصاویر کد های مربوطه آورده شده است :

```python
def BFS() :
    global required_recipes
    q.append((seyed_loc, 1, [], [], [], required_recipes))
    while q :
        curr_v, spent_time, tmp_satisfied_devotees, tmp_path, tmp_seen_recipes, tmp_required_recipes = q.pop(0)
        if (curr_v, tmp_seen_recipes, tmp_satisfied_devotees) not in visited_states :
            visited_states.append((curr_v, tmp_seen_recipes, tmp_satisfied_devotees))
        else :
            continue
        path = deepcopy(tmp_path)
        seen_recipes = deepcopy(tmp_seen_recipes)
        satisfied_devotees = deepcopy(tmp_satisfied_devotees)
        required_recipes = {key: list(val) for key, val in tmp_required_recipes.items()}
        path.append(curr_v)

        if curr_v in impassable_vertexes :
            n = path.count(curr_v)
            if n > 1 and spent_time < n :
                q.append((curr_v, spent_time + 1, satisfied_devotees, path, seen_recipes, required_recipes))
                continue

        check_recipe(curr_v, seen_recipes, required_recipes)
        check_devotee(curr_v, required_recipes, satisfied_devotees)

        for vertex in adj_vertexes[curr_v] :
            if (vertex, seen_recipes, satisfied_devotees) not in visited_states :
                if (vertexes[vertex].includes_devotee and len(list(required_recipes.keys())) == 1
                    and len(seen_recipes) == recipes_count) :
                    if vertex in list(required_recipes.keys()) :
                        path.append(vertex)
                        return path
                q.append((vertex, 1, satisfied_devotees, path, seen_recipes, required_recipes))
```

```python
def check_devotee(v, required_recipes, satisfied_devotees) :
    if vertexes[v].includes_devotee :
        if v in list(required_recipes.keys()) :
            if len(required_recipes[v]) == 0 :
                required_recipes.pop(v)
                if v not in satisfied_devotees :
                    satisfied_devotees.append(v)
def check_recipe(v, seen_recipes, required_recipes) :
    if vertexes[v].includes_recipe :
        if v not in seen_recipes :
            seen_recipes.append(v)
            del_recipe(v, required_recipes)
def del_recipe(recipe, required_recipes) :
    for each in list(required_recipes.values()) :
        if recipe in each :
            each.remove(recipe)
```

- IDS : تست گل همان تست گل مربوط به BFS است و فرق ندارد. چیز هایی که در BFS داخل صف push میشدند، اینجا به خود تابع pass داده میشوند. بررسی این که رأس دستور پخت دارد یا خیر و انجام دادن کار های بعد از آن و همچنین بررسی این که رأس مرید دارد یا خیر و انجام دادن کار های بعد از آن داخل خود تابع IDS انجام میشود و تابع جدا نزده شده. تنها نکته ای که در اینجا با BFS متفاوت است این است که به جای BFS، DFS انجام میدهیم ولی هر بار این کار را تا عمقی مشخص انجام میدهیم که این عمق از 0 شروع شده و تا پیدا شدن جواب یک واحد یک واحد افزایش می یابد. نکته مهم دیگر در این الگوریتم این است که چون در یک عمق که به جواب برسیم ممکن است چندین رأس صعب العبور در مسیرمان وجود داشته باشد یا از یک رأس صعب العبور چندین بار رد شده باشیم، تمام جواب های ممکن در یک عمق را می یابیم و در انتها کم هزینه ترین را به عنوان جواب بهینه اعلام میکنیم. در ادامه کد های مربوطه آورده شده است :

```python
def IDS(curr_v, visited_states, satisfied_devotees, seen_recipes, required_recipes, path, depth, depth_limit) :
    global solution_path, ans_depth, ans_found, visited_states_count

    if ans_found and depth > ans_depth :
        return

    if (vertexes[curr_v].includes_devotee and len(list(required_recipes.keys())) == 1
    and len(seen_recipes) == recipes_count and curr_v in list(required_recipes.keys())) :
        path.append(curr_v)
        solution_paths.append(path)
        ans_depth = depth
        ans_found = True

    if depth > depth_limit :
        visited_states_count += len(visited_states)
        return

    tmp_visited_states = deepcopy(visited_states)
    tmp_satisfied_devotees = deepcopy(satisfied_devotees)
    tmp_seen_recipes = deepcopy(seen_recipes)
    tmp_required_recipes = {key : list(val) for key, val in required_recipes.items()}
    tmp_path = deepcopy(path)

    if (curr_v, tmp_seen_recipes, tmp_satisfied_devotees) not in visited_states :
        tmp_visited_states.append((curr_v, tmp_seen_recipes, tmp_satisfied_devotees))

    if vertexes[curr_v].includes_recipe :
        if curr_v not in tmp_seen_recipes :
            tmp_seen_recipes.append(curr_v)
            del_recipe(curr_v, tmp_required_recipes)

    if vertexes[curr_v].includes_devotee :
        if curr_v in list(tmp_required_recipes.keys()) :
            if len(tmp_required_recipes[curr_v]) == 0 :
                tmp_required_recipes.pop(curr_v)
                if curr_v not in tmp_satisfied_devotees :
                    tmp_satisfied_devotees.append(curr_v)
```

```python
while len(solution_paths) == 0 :
    depth_limit += 1
    IDS(seyed_loc, [], [], [], required_recipes, [], 0, depth_limit)
```

- A* :

چیزی که به عنوان heuristic در نظر گرفته شده مجموع تعداد مرید های ناراضی و مجموع دستور پخت های دیده نشده (اگر رأسی چندین مورد از مرید ها یا دستور پخت ها را داشت باید این تعداد را از مجموع مان کم کنیم چون در واقع باید اجتماع بگیریم و چنین مواردی موارد تکراری هستند)است. برای آن که heuristic مان consistent باشد باید رابطه زیر برقرار باشد (رأس C بین دو رأس A و G است) :

$$\text{real cost}(A \text{ to } C) \geq h(A) - h(C)$$

از طرفی برای آن که به هدفمان برسیم بایستی هم تمام دستور پخت ها را دیده باشیم و هم تمام مرید ها را و در این بین ممکن است مجبور باشیم از رئوس دیگری هم عبور کنیم. پس اگر فرض کنیم $h(A) = a$ و $h(C) = c$ در این صورت $a - c$ تا خانه را برای رفتن از A تا C باید حداقل ببینیم. پس هزینه واقعی بیشتر یا مساوی این مقدار است و heuristic انتخاب شده consistent است.

در مورد پیاده سازی الگوریتم کاملا مشابه BFS است با این تفاوت که این بار به جای صف از min-heap استفاده میکنیم و هر بار کم هزینه ترین استیت pop خواهد شد. ضمنا برای این که weighted A* هم پوشش داده شود متغیری به نام alpha به تابع pass میدهیم که در حالت عادی alpha = 1 و در حالت وزن دار alpha > 1 است.
در ادامه کد های مربوطه آورده شده است :

```
95  def astar(alpha, seen_recipes, tmp_required_recipes, satisfied_devotees) :
96      if len(q) == 0 :
97          check_recipe(seyed_loc, seen_recipes, tmp_required_recipes)
98          check_devotee(seyed_loc, tmp_required_recipes, satisfied_devotees)
99          visited_states.append((seyed_loc, seen_recipes, satisfied_devotees))
100         heappush(q, (alpha * (s - len(satisfied_devotees) + recipes_count - len(seen_recipes)),
101         (seyed_loc, satisfied_devotees, [seyed_loc], seen_recipes, tmp_required_recipes)))
102     while q :
103         displeased_devotees_count, info = heappop(q)
104         curr_v, tmp_satisfied_devotees, tmp_path, tmp_seen_recipes, tmp_required_recipes = info
105         for vertex in adj_vertexes[curr_v] :
106             path = deepcopy(tmp_path)
107             seen_recipes = deepcopy(tmp_seen_recipes)
108             satisfied_devotees = deepcopy(tmp_satisfied_devotees)
109             required_recipes = {key: list(val) for key, val in tmp_required_recipes.items()}
110             path.append(vertex)
111             if (vertexes[vertex].includes_devotee and len(list(required_recipes.keys())) == 1
112             and len(seen_recipes) == recipes_count) :
113                 if vertex in list(required_recipes.keys()) :
114                     return path
115             check_recipe(vertex, seen_recipes, required_recipes)
116             check_devotee(vertex, required_recipes, satisfied_devotees)
117             if (vertex, seen_recipes, satisfied_devotees) not in visited_states :
118                 visited_states.append((vertex, seen_recipes, satisfied_devotees))
119                 heappush(q, (alpha * (s - len(satisfied_devotees)  + recipes_count - len(seen_recipes))
120                 + calc_path_time(path), (vertex, satisfied_devotees, path, seen_recipes, required_recipes)))
```

در ادامه هر الگوریتم را به ازای تست های مختلف 3 بار اجرا میکنیم و نتیجه و زمان میانگین را نمایش میدهیم :

- اجرای هر 3 الگوریتم به ازای فایل input.txt :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
BFS execution time = 0.0009772777557373047 seconds
visited states count = 29
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
BFS execution time = 0.0009348392486572266 seconds
visited states count = 29
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
BFS execution time = 0.0009243488311767578 seconds
visited states count = 29
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
IDS execution time = 0.03693032264709473 seconds
visited states count = 2491
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
IDS execution time = 0.03656315803527832 seconds
visited states count = 2491
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
IDS execution time = 0.03687930107116699 seconds
visited states count = 2491
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 1 A* execution time = 0.0014705657958984375 seconds
visited states count = 36
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 1 A* execution time = 0.002270221710205078 seconds
visited states count = 36
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 1 A* execution time = 0.0022470951080322266 seconds
visited states count = 36
```

```
Average Time for BFS = 0.00094548861185709 63 seconds
Average Time for IDS = 0.036790927251180015 seconds
Average Time for A* = 0.0019959608713785806 seconds
```

- اجرای هر 3 الگوریتم به ازای فایل input2.txt از تست های ساده :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 9 -> 10 -> 9 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
BFS execution time = 0.0037221908569335938 seconds
visited states count = 94
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 9 -> 10 -> 9 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
BFS execution time = 0.0037055015563964844 seconds
visited states count = 94
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 9 -> 10 -> 9 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
BFS execution time = 0.003722667694091797 seconds
visited states count = 94
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 9 -> 10 -> 9 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
IDS execution time = 0.08320498466491699 seconds
visited states count = 4235
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 9 -> 10 -> 9 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
IDS execution time = 0.0820779800415039 seconds
visited states count = 4235
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 9 -> 10 -> 9 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
IDS execution time = 0.08200335502624512 seconds
visited states count = 4235
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 10 -> 2 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
For alpha = 1 A* execution time = 0.0031540393829345703 seconds
visited states count = 63
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 10 -> 2 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
For alpha = 1 A* execution time = 0.0010950565338134766 seconds
visited states count = 63
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 10 -> 2 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
For alpha = 1 A* execution time = 0.0030837059020996094 seconds
visited states count = 63
```

```
Average Time for BFS = 0.0023311376571655273 seconds
Average Time for IDS = 0.059609850247701 01 seconds
Average Time for A* = 0.0022201140721639 seconds
```

- اجرای هر 3 الگوریتم به ازای فایل input3.txt از تست های ساده :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
BFS execution time = 0.31853485107421875 seconds
visited states count = 3153
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
BFS execution time = 0.4064762592315674 seconds
visited states count = 3153
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
BFS execution time = 0.40838146209716797 seconds
visited states count = 3153
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
IDS execution time = 0.830805778503418 seconds
visited states count = 175098
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
IDS execution time = 0.8250677585601807 seconds
visited states count = 175098
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 IDS.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
IDS execution time = 0.8213474750518799 seconds
visited states count = 175098
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
For alpha = 1 A* execution time = 0.1262800693511963 seconds
visited states count = 1279
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
For alpha = 1 A* execution time = 0.12772583961486816 seconds
visited states count = 1279
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
For alpha = 1 A* execution time = 0.1294236183166504 seconds
visited states count = 1279
```

```
Average Time for BFS = 0.37779752413431805 seconds
Average Time for IDS = 0.8257403373718262 seconds
Average Time for A* = 0.12780984242757162 seconds
```

- اجرای 2 الگوریتم BFS و A* به ازای فایل input2.txt از تست های عادی :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 28 -> 19 -> 13 -> 3 -> 11 -> 24 -> 9 -> 23 -> 28 -> 23 -> 5 -> 7 -> 29
path time = 12
BFS execution time = 59.76086974143982 seconds
visited states count = 34377
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 28 -> 19 -> 13 -> 3 -> 11 -> 24 -> 9 -> 23 -> 28 -> 23 -> 5 -> 7 -> 29
path time = 12
BFS execution time = 58.769814014434814 seconds
visited states count = 34377
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 28 -> 19 -> 13 -> 3 -> 11 -> 24 -> 9 -> 23 -> 28 -> 23 -> 5 -> 7 -> 29
path time = 12
BFS execution time = 56.02470135688782 seconds
visited states count = 34377
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 13 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 22 -> 28
path time = 12
For alpha = 1 A* execution time = 0.5806772708892822 seconds
visited states count = 7809
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 13 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 22 -> 28
path time = 12
For alpha = 1 A* execution time = 0.5968000888824463 seconds
visited states count = 7809
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 13 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 22 -> 28
path time = 12
For alpha = 1 A* execution time = 0.5908350944519043 seconds
visited states count = 7809
```

```
Average Time for BFS = 58.185128370920815 seconds
Average Time for A* = 0.5894374847412109 seconds
```

- اجرای 2 الگوریتم BFS و A* به ازای فایل input3.txt از تست های عادی :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
BFS execution time = 3.367645263671875 seconds
visited states count = 11504
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
BFS execution time = 3.3217265605926514 seconds
visited states count = 11504
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 BFS.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
BFS execution time = 3.350698709487915 seconds
visited states count = 11504
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 1 A* execution time = 1.2486255168914795 seconds
visited states count = 9270
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 1 A* execution time = 1.2743346691131592 seconds
visited states count = 9270
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 1 A* execution time = 1.2524571418762207 seconds
visited states count = 9270
```

```
Average Time for BFS = 3.3466901779174805 seconds
Average Time for A* = 1.2584724426269531 seconds
```

اجرای الگوریتم A* به ازای دو مقدار $\alpha = 1.8$ و $\alpha = 4$ برای تمام فایل های ورودی :

- فایل input.txt :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 1.8 A* execution time = 0.0016169548034667969 seconds
visited states count = 29
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 1.8 A* execution time = 0.001657247543334961 seconds
visited states count = 29
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 1.8 A* execution time = 0.001659393310546875 seconds
visited states count = 29
```

```
Average Time for A* = 0.0016445318857828777 seconds
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 4 A* execution time = 0.0011134147644042969 seconds
visited states count = 19
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 4 A* execution time = 0.0010864734649658203 seconds
visited states count = 19
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 1 -> 3 -> 4 -> 5 -> 7 -> 10 -> 11 -> 9 -> 8
path time = 8
For alpha = 4 A* execution time = 0.0011069774627685547 seconds
visited states count = 19
```

```
Average Time for A* = 0.0011022885640462239 seconds
```

- فایل input2.txt از تست های ساده :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 10 -> 2 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
For alpha = 1.8 A* execution time = 0.0026314258575439453 seconds
visited states count = 51
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 10 -> 2 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
For alpha = 1.8 A* execution time = 0.0026731491088867188 seconds
visited states count = 51
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 10 -> 2 -> 4 -> 12 -> 3 -> 7 -> 5 -> 8
path time = 8
For alpha = 1.8 A* execution time = 0.002595663070678711 seconds
visited states count = 51
```

```
Average Time for A* = 0.0026334126790364585 seconds
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 4 -> 2 -> 10 -> 8 -> 5 -> 7 -> 3 -> 7 -> 5 -> 8
path time = 12
For alpha = 4 A* execution time = 0.0015757083892822266 seconds
visited states count = 33
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 4 -> 2 -> 10 -> 8 -> 5 -> 7 -> 3 -> 7 -> 5 -> 8
path time = 12
For alpha = 4 A* execution time = 0.001544952392578125 seconds
visited states count = 33
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 9 -> 4 -> 2 -> 10 -> 8 -> 5 -> 7 -> 3 -> 7 -> 5 -> 8
path time = 12
For alpha = 4 A* execution time = 0.0015306472778320312 seconds
visited states count = 33
```

```
Average Time for A* = 0.0015504360198974609 seconds
```

- فایل input3.txt از تست های ساده :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
For alpha = 1.8 A* execution time = 0.0024302005767822266 seconds
visited states count = 49
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
For alpha = 1.8 A* execution time = 0.002456188201904297 seconds
visited states count = 49
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11 -> 10
path time = 13
For alpha = 1.8 A* execution time = 0.0024051666259765625 seconds
visited states count = 49
```

```
Average Time for A* = 0.0024305184682210 29 seconds
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 10 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11
path time = 14
For alpha = 4 A* execution time = 0.002110004425048828 seconds
visited states count = 39
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 10 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11
path time = 14
For alpha = 4 A* execution time = 0.0020835399627685547 seconds
visited states count = 39
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 13 -> 11 -> 10 -> 3 -> 2 -> 6 -> 12 -> 5 -> 10 -> 5 -> 9 -> 4 -> 1 -> 13 -> 11
path time = 14
For alpha = 4 A* execution time = 0.002099275588989258 seconds
visited states count = 39
```

```
Average Time for A* = 0.0020976066589355 47 seconds
```

- فایل input2.txt از تست های عادی :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 20 -> 13 -> 19 -> 28
path time = 13
For alpha = 1.8 A* execution time = 0.007253170013427734 seconds
visited states count = 112
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 20 -> 13 -> 19 -> 28
path time = 13
For alpha = 1.8 A* execution time = 0.007242918014526367 seconds
visited states count = 112
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 20 -> 13 -> 19 -> 28
path time = 13
For alpha = 1.8 A* execution time = 0.007302522659301758 seconds
visited states count = 112
```

```
Average Time for A* = 0.00726620356241862 seconds
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 20 -> 13 -> 19 -> 28
path time = 13
For alpha = 4 A* execution time = 0.006523847579956055 seconds
visited states count = 110
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 20 -> 13 -> 19 -> 28
path time = 13
For alpha = 4 A* execution time = 0.0065991878509521484 seconds
visited states count = 110
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 28 -> 19 -> 3 -> 11 -> 24 -> 9 -> 2 -> 5 -> 7 -> 29 -> 20 -> 13 -> 19 -> 28
path time = 13
For alpha = 4 A* execution time = 0.006453752517700195 seconds
visited states count = 110
```

```
Average Time for A* = 0.0065255959828694 66 seconds
```

- فایل input3.txt از تست های عادی :

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 1.8 A* execution time = 0.5125980377197266 seconds
visited states count = 3968
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 1.8 A* execution time = 0.5053021907806396 seconds
visited states count = 3968
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 1.8 A* execution time = 0.5048811435699463 seconds
visited states count = 3968
```

```
Average Time for A* = 0.5075937906901041 seconds
```

```
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 4 A* execution time = 0.008261919021606445 seconds
visited states count = 136
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 4 A* execution time = 0.00829458236694336 seconds
visited states count = 136
javad@Javad-Bshrt:~/My Folders/University/5th Term/AI/CAs/CA1$ python3 astar.py
path : 40 -> 42 -> 38 -> 24 -> 31 -> 45 -> 30 -> 48 -> 41 -> 18 -> 1 -> 19 -> 43 -> 49 -> 47 -> 49 -> 9 -> 34 -> 25 -> 50 -> 12 -> 16
path time = 21
For alpha = 4 A* execution time = 0.008384943008422852 seconds
visited states count = 136
```

```
Average Time for A* = 0.008313814798990885 seconds
```

الگوریتم های BFS، IDS و A* عادی همواره جواب بهینه تولید میکنند. الگوریتم A* وزن دار، ممکن است جواب بهینه بدهد یا خیر ولی در قیاس با دیگر الگوریتم ها سریع ترین است.

از لحاظ مقایسه سرعت الگوریتم ها هم داریم :

weighted A* > A* > BFS > IDS