حسابگری زیستی؛
حل مسئله تخصیص به کمک بهینه‌سازی کلونی مورچه؛
کامیار میرزاوزیری؛ ۶۱۰۳۹۶۱۵۲

---

۱ خواندن مسئله

پیش از هر کاری لازم است مسئله را از فایل ورودی بخوانیم و آن را پارس کنیم تا بتوانیم هزینهٔ لازم برای انجام تسک iـام توسط شخص jـام را برای هر i و j داشته باشیم. برای این منظور به سادگی کد زیر را می‌نویسیم.

```
class AP:
def __init__(self, path: str):
    with open(path) as f:
        self.costs = list(map(
            lambda line: list(map(int, line.split())),
            f.readlines()
        ))
```

برای مثال با اجرای کد زیر

```
FILE_PATH = 'job1.assign'
ap = AP(FILE_PATH)
print(ap.costs[2][1])
```

مقدار 10 را خروجی می‌گیریم که نمایانگر هزینهٔ انجام کار شمارهٔ 2 برای شخص شمارهٔ 1 می‌باشد.

۲ مورچه‌ها و پاسخ مسئله

می‌دانیم پاسخ مسئله به صورت یک لیست از کارها می‌باشد که عضو شمارهٔ iـام آن، نمایانگر شخصی است که کار iـام را انجام می‌دهد به طوری که هر شخص حداکثر کار را انجام می‌دهد. پس در واقع جایگشتی از اعداد صفر تا $n-1$ می‌باشد. لذا انتظار داریم که یک مورچه حافظه‌ای شامل تخصیص‌هایی که تا کنون انجام داده داشته باشد و در هر مرحله، کار جدیدی که به شخصی تخصیص نیافته، به یکی از اشخاص جدید تخصیص دهد.

```
class Ant:
    def __init__(self, ap: AP):
        self.ap = ap
        self.assignments = [-1] * self.ap.count
        self.cost = None
```

۳ فورمون

می‌توانیم تخصیص‌ها را به صورت یال‌هایی بین اشخاص و کارها در نظر بگیریم، برای هر یک از این یال‌ها مقداری فورمون در نظر می‌گیریم و این مقادیر را در کلاس AP ذخیره می‌کنیم تا برای تمامی مورچه‌ها قابل دسترسی باشد.

```
self.count = len(self.costs)
for _ in range(self.count):
    self.pheromones = [[1 / self.costs[i][j] for j in range(self.count)]
   for i in range(self.count)]
```

در ابتدا مقدار فومون روی هر یال را متناسب با هزینهٔ آن یال در نظر می‌گیریم به طوری که هرچه هزینهٔ کمتری داشته باشد فورمون بیشتری دارد.

۴ تخصیص

یک مورچه در مرحلهٔ kـام از یافتن پاسخ باید تصمیم بگیرد که کار kـام را به کدام شخص تخصیص دهد. که این فرایند به کمک فرمول موجود انجام می‌شود. پس به سادگی کد زیر را می‌نویسیم.

```
def assign(self, k):
    weights = [
        (self.ap.pheromones[k][i] ** ALPHA) *
        ((1 / self.ap.costs[k][i]) ** BETA)
        if i not in self.assignments[:k] else 0
        for i in range(self.ap.count)
    ]
    self.assignments[k] = random.choices(
        range(self.ap.count),
        weights=weights,
        k=1
    )[0]
```

۵ کلونی مورچه

یک کلونی شامل تعدادی مورچه که قابل تنظیم است به کلاس AP اضافه می‌کنیم.

```
self.colony = [Ant(self) for _ in range(COLONY_POPULATION)]
```

۶ قدم‌ها

برنامه را طوری طراحی می‌کنیم که در هر قدم از اجرای برنامه، تمامی مورچه‌ها هر کدام یک تخصیص را به صورت همزمان انجام دهند. برای این منظور متد step را به کلاس AP اضافه می‌کنیم.

```
def step(self, k):
    for ant in self.colony:
        ant.assign(k)
```

۷ تبخیر

متدی به کلاس AP اضافه می‌کنیم که وظیفهٔ تبخیر فورمون موجود بر روی یال‌ها را بر عهده بگیرد. این کار به سادگی قابل انجام می‌باشد.

```
def evaporate(self):
    for i in range(self.count):
        for j in range(self.count):
            self.pheromones[i][j] *= (1 - EVAPORATION_PARAMETER)
```

۸ ترشح فورمون

پس از این که یک دور کامل از قدم‌ها انجام شد و هر مورچه به یک پاسخ رسید، می‌توانیم به کمک فرمول موجود، مقداری فورمون روی هر یک از یال‌های موجود در مسیر هر مورچه ترشح کنیم. این کار را به سادگی به کمک متد زیر انجام می‌دهیم.

```
def secret_pheromones(self):
    for ant in self.colony:
        ant.calculate_cost()
        delta = 1 / ant.cost
        for i in range(self.count):
            j = ant.assignments[i]
            self.pheromones[i][j] += delta

        if self.total_best_ant_cost is not None:
            e_delta = ELITISM / self.total_best_ant_cost
            for i in range(self.count):
                j = self.total_best_ant_assignments[i]
                self.pheromones[i][j] += e_delta
```

توجه کنید که برای محاسبهٔ هزینه، متد `calculate_cost` را به کلاس `Ant` اضافه کردیم.

```
def calculate_cost(self):
    self.cost = sum([self.ap.costs[i][self.assignments[i]] for i in range(
    self.ap.count)])
```

همچنین در این متد ترشح فورمون از روش مورچه نخبه بهره می‌بریم تا نتیجهٔ بهتری حاصل شود.

۹ دورها

برای یافتن جواب قابل قبول لازم است که فرایند ذکرشده را بارها تکرار کنیم، برای این منظور متد iterate را در کلاس AP پیاده‌سازی می‌کنیم که هر بار اجرای آن موجب یافتن n پاسخ برای مسئله توسط هر یک از n مورچه می‌شود.

```
def iterate(self):
    for k in range(self.count):
        self.step(k)
    self.evaporate()
    self.secret_pheromones()
```

دقت کنیم که تصمیم‌گیری مورچه‌ها در قسمت اول، اثرات جانبی ندارد و می‌توانیم آن را به صورت موازی پیاده‌سازی کنیم تا از منابع سیستم حداکثر استفاده را برده و عملکرد بهتری داشته باشیم.

```
def iterate(self):
    if PARALLEL:
        def chunk_assgin(indices, ants, rd):
            for i in range(len(indices)):
                for k in range(self.count):
                    ants[i].assign(k)
                rd[indices[i]] = ants[i].assignments

        chunks = numpy.array_split(range(len(self.colony)), P_COUNT)
        manager = mp.Manager()
        rd = manager.dict()
        processes = [
            mp.Process(target=chunk_assgin, args=(chunk, self.colony[chunk
    [0]:chunk[-1] + 1], rd))
            for chunk in chunks
        ]

        for p in processes:
            p.start()
        for p in processes:
            p.join()

        for i in range(len(self.colony)):
            self.colony[i].assignments = rd[i]
    else:
        for k in range(self.count):
            self.step(k)

    self.evaporate()
```

```
        self.secret_pheromones()
```

برای این منظور پارامتر P_COUNT را به اندازه تعداد هسته‌های CPU سیستمی که برنامه روی آن اجرا می‌شود در نظر می‌گیریم، و کلونی را به این تعداد گروه تقسیم می‌کنیم و وظیفهٔ محاسبهٔ تخصیص‌های هر گروه از مورچه‌ها را به یک هسته محول می‌کنیم.

۱۰ حل مسئله

در نهایت لازم است که چند بار کل این فرایند را تکرار کنیم تا پاسخ خوبی به دست بیاوریم. برای این منظور متد solve را به کلاس AP اضافه می‌کنیم.

```
def solve(self):
    i = 0
    stagnancy = 0
    while True:
        self.iterate()
        i += 1

        new_best_ant = min(self.colony, key=lambda ant: ant.cost)
        if self.total_best_ant_cost is not None:
            if self.total_best_ant_cost - new_best_ant.cost <
    IMPROVEMENT_THRESHOLD:
                stagnancy += 1
                if stagnancy >= STAGNANCY_THRESHOLD:
                    break
            else:
                stagnancy = 0

        if self.total_best_ant_cost is None or new_best_ant.cost < self.
    total_best_ant_cost:
            self.total_best_ant_cost = new_best_ant.cost
            self.total_best_ant_assignments = new_best_ant.assignments.copy
    ()

    return [self.total_best_ant_cost, self.total_best_ant_assignments]
```

همانطور که طبق کد واضح است برای شرط توقف از دو مفهوم بهبود و رکود کمک گرفتیم، به این صورت که اگر بهترین پاسخ بعد از تعداد مشخصی دور، هر بار بهبود اندکی داشته باشد، الگوریتم را پایان می‌دهیم.
نهایتا کد زیر را برای حل مسئله اجرا می‌کنیم.

```
ap = AP(FILE_PATH)

t_start = time.time()
```

```
answer = ap.solve()
t_end = time.time()

print(f'Took: {t_end - t_start:.5f} seconds')
print(f'Cost: {answer[0]}')
print('Assignments:', end=' ')
print(answer[1])
```

۱۱ گزارشات

حال کد بالا را برای هر یک از آزمایش‌های موجود در تمرین اجرا می‌کنیم و در هر مورد با آزمون‌وخطا پارامترها را طوری پیدا می‌کنیم که الگوریتم سریع‌تر به پاسخ برسد.

۱.۱۱ آزمایش اول

پارامترها را به صورت زیر مقداردهی می‌کنیم و کد را اجرا می‌کنیم.

```
FILE_PATH = 'job1.assign'

COLONY_POPULATION = 10
ALPHA = 2
BETA = 2
EVAPORATION_PARAMETER = .3
ELITISM = 1
IMPROVEMENT_THRESHOLD = 1
STAGNANCY_THRESHOLD = 5

PARALLEL = False
```

خروجی زیر را دریافت می‌کنیم.

```
Took: 0.00138 seconds
Cost: 26
Assignments: [3, 1, 2, 0]
```

۲.۱۱ آزمایش دوم

پارامترها را به صورت زیر مقداردهی می‌کنیم و کد را اجرا می‌کنیم.

```
FILE_PATH = 'job2.assign'

COLONY_POPULATION = 100
```

```
ALPHA = 2
BETA = 2
EVAPORATION_PARAMETER = .3
ELITISM = 1.5
IMPROVEMENT_THRESHOLD = 1
STAGNANCY_THRESHOLD = 20

PARALLEL = True
```

خروجی زیر را دریافت می‌کنیم.

```
Took: 11.41899 seconds
Cost: 317
Assignments: [90, 89, 61, 74, 68, 81, 49, 57, 22, 43, 21, 30, 87, 97, 69,
   84, 10, 60, 35, 26, 5, 72, 31, 24, 75, 9, 67, 91, 92, 46, 64, 77, 17,
   98, 25, 85, 42, 29, 13, 71, 79, 18, 86, 48, 12, 3, 27, 99, 95, 7, 82,
   39, 88, 20, 58, 16, 41, 37, 38, 52, 32, 14, 66, 59, 93, 65, 73, 62, 78,
   4, 63, 94, 70, 23, 76, 28, 83, 2, 33, 56, 40, 80, 0, 50, 47, 8, 15, 34,
   53, 44, 55, 6, 11, 1, 36, 45, 19, 54, 51, 96]
```

۳.۱۱ آزمایش سوم

پارامترها را به صورت زیر مقداردهی می‌کنیم و کد را اجرا می‌کنیم.

```
FILE_PATH = 'job3.assign'

COLONY_POPULATION = 200
ALPHA = 5
BETA = 5
EVAPORATION_PARAMETER = .1
ELITISM = 1.5
IMPROVEMENT_THRESHOLD = 1
STAGNANCY_THRESHOLD = 20

PARALLEL = True
```

خروجی زیر را دریافت می‌کنیم.

```
Took: 108.03951 seconds
Cost: 710
Assignments: [8, 89, 93, 176, 14, 104, 129, 15, 197, 40, 110, 128, 158,
   195, 115, 66, 83, 11, 27, 42, 168, 45, 140, 106, 48, 149, 67, 3, 150,
   87, 147, 16, 175, 23, 179, 72, 30, 142, 146, 170, 159, 32, 183, 56, 134,
    58, 18, 160, 92, 68, 49, 157, 112, 103, 105, 39, 122, 187, 113, 20,
   109, 85, 98, 52, 62, 165, 131, 55, 114, 190, 188, 73, 64, 29, 194, 82,
```

86, 31, 88, 4, 101, 196, 108, 26, 78, 107, 10, 51, 65, 41, 12, 198, 154,
6, 117, 34, 166, 178, 80, 189, 22, 74, 61, 35, 136, 148, 182, 63, 138,
156, 139, 124, 84, 24, 120, 0, 172, 19, 152, 81, 47, 191, 37, 192, 126,
164, 135, 33, 132, 174, 71, 143, 25, 2, 141, 57, 28, 38, 144, 145, 17,
7, 186, 180, 137, 184, 90, 169, 94, 130, 36, 99, 75, 69, 173, 162, 46,
97, 118, 76, 1, 44, 199, 121, 43, 177, 127, 79, 54, 21, 155, 123, 53,
119, 96, 70, 50, 116, 91, 102, 151, 171, 181, 167, 185, 161, 100, 125,
163, 5, 193, 77, 13, 59, 9, 111, 95, 153, 133, 60]

^