مهلا شریفی

۹۸۳۱۰۳۵

تمرین عملی دوم

درس معماری افزاره‌های شبکه

فهرست

راهنمای اجرا:

```
#To run for specific number of time-slots and ports
run()
# depicting effect of number port number on throughput for specific number of time-slots
plot_throughput_vs_ports(31035)
```

یکی از دو خط ۱۳۱ یا ۱۳۳ باید کامنت شوند.(وابسته به اینکه کدام قسمت از تمرین را می‌خواهید اجرا کنید.)

سوئیچ:

```
class Switch:
    def __init__(self, num_ports):
        self.num_ports = num_ports
        self.current_match = list(range(num_ports))  # Initial state: input i connects to output i

        self.input_buffers = []
        for i in range(num_ports):
            # Create a row for each input port which shows length of its virtuall queue
            # (all virtuall queue are empty at first)
            buffer_row = [0 for _ in range(num_ports)]
            self.input_buffers.append(buffer_row)
```

سوئیچ سه فیلد دارد.

1 - تعداد پورت‌های ورودی و خروجی

2 - Match فعلی – همانطور که در دستور کار گفته شده، match اولیه دلخواه درنظر گرفته شده است به این شکل که پورت ورودی i به پورت خروجی i، match می‌شود.

3 - صف‌های مجازی هر پورت خروجی – چون نیازی به پیاده سازی بسته نبوده است، برای هر پورت ورودی یک لیست در نظر گرفته شده که اندیس iام این لیست، طول صف مجازی پورت خروجی i را در پورت ورودی مربوطه نشان می‌دهد. در ابتدا صف‌ها خالی هستند به همین دلیل مقادیر لیست‌ها صفر است.

اجرای شبیه‌سازی

```
90      def run_simulation(self, num_time_slots):
91          total_throughput = 0
92          print("Time Slot | Current Match | Throughput")
93          for t in range(num_time_slots):
94              self.update_input_buffers()
95              throughput = self.run_time_slot(t)/(self.num_ports)
96              total_throughput += throughput
97              print(f"{t+1:9} | {str(self.current_match):14} | {throughput}")
98              self.process_current_match()
99
100         average_throughput = total_throughput / num_time_slots
101         print(f"\nAverage Throughput over {num_time_slots} time slots: {average_throughput}")
102         return average_throughput
```

خطوط ۹۴ تا ۹۸ : در هر time-slot اجرا می‌شوند.

خط ۹۴: به ازای هر پورت ورودی یک بسته وارد می‌شود که با احتمال برابر متعلق به یکی از صف‌های مجازی پورت ورودی خواهد بود.

خط ۹۵: matching جدید انجام می‌شود.

خط۶: بسته‌هایی که match شده‌اند از صف ورودی حذف می‌شوند.

```
80      def update_input_buffers(self):
81          for in_port in range(self.num_ports):
82              self.input_buffers[in_port][random.randint(0, self.num_ports-1)] += 1
83
```

با وجود خط ۸۳ و قسمتی که مشخص شده است تضمین می‌شود که بسته‌ها با احتمال برابر به یکی از صف‌های خروجی تعلق می‌یابند.

```
50      def run_time_slot(self, t):
51          # Get all neighbor matchings
52          neighbors = self.get_neighbors(self.current_match)
53
54          # Start with the current match as the best match
55          max_weight = self.compute_weight(self.current_match)
56          best_match = self.current_match
57
58          # Check all neighbor matchings to find the best one
59          for neighbor in neighbors:
60              weight = self.compute_weight(neighbor)
61              if weight > max_weight:
62                  max_weight = weight
63                  best_match = neighbor
64
65          # Compute the Hamiltonian walk and its weight
66          hamiltonian = self.hamiltonian_walk(t)
67          hamiltonian_weight = self.compute_weight(hamiltonian)
68
69          # Check if the Hamiltonian walk is better than the current best
70          if hamiltonian_weight > max_weight:
71              best_match = hamiltonian
72
73          # Set the best matching as the current match
74          self.current_match = best_match
75
76          # Calculate throughput for the current match
77          throughput = self.calculate_throughput(self.current_match)
78          return throughput
```

Match جدید باید از انتخاب شود.

در خطوط ۵۲ تا ۷۱، از بین همسایه‌ها،خروجی قدم همیلتونی و match فعلی، matchای که بیشترین وزن را دارد، پیدا می‌شود. خروجی تابع run_time_slot، گذردهی برای match انتخاب شده است.

وزن‌دهی

```
29      def compute_weight(self, match):
30          weight = 0
31          for i, output in enumerate(match):
32              if self.input_buffers[i][output] > 0:
33                  weight+=1
34          return weight
```

وزن هر match می‌تواند با معیارهای مختلفی سنجیده شود. من از تعداد بسته‌هایی که به ازای آن match عبور می‌کنند (گذردهی)استفاده کرده‌ام. اگر صف مجازی پورت ورودی به ازای پورت خروجی match شده خالی نباشد، یک واحد به وزن match افزوده می‌شود. (خطوط ۳۲ و ۳۳)

همسایه‌ها

```
18      def get_neighbors(self, match):
19          # Generating neighbors by swapping pairs of connections
20          neighbors = []
21          for i in range(self.num_ports):
22              for j in range(i+1, self.num_ports):
23                  neighbor = match.copy()
24                  #swap values directly
25                  neighbor[i] , neighbor[j] = neighbor[j], neighbor[i]
26                  neighbors.append(neighbor)
27          return neighbors
```

خط ۲۳: از match فعلی یک کپی گرفته می‌شود.

خط ۲۵: محتوای دو خانه‌ی i و j با یکدیگر عوض می‌شوند و به این ترتیب یکی از همسایه‌ها ایجاد می‌شود. (باقی نیز به همین شکل ایجاد می‌شوند و به عنوان خروجی لیستی از همسایه‌های match ورودی باز گردانده می‌شود.)

قدم همیلتونی

```
36      def hamiltonian_walk(self, t):
37          perm_iter = permutations(range(self.num_ports))
38          selected_perm = next(islice(perm_iter, t, None), None)
39          return list(selected_perm) if selected_perm else list(range(self.num_ports))
40
```

خط ۳۷: برای به دست آوردن جایگشت‌ها از کتابخانه‌ی آماده‌ی itertools استفاده شده است.

خط ۳۸: permutations تمام جایگشت‌ها را باز می‌گرداند که به لحاظ محاسباتی استخراج عضو tام از لیست جایگشت‌ها زمان‌بر است. خط ۳۸ به این منظور است که جایگشت tام را مستقیما بگیریم و هربار نیازی به محاسبه‌ی کل جایگشت‌ها نداشته باشیم.

گذردهی

```
41      def calculate_throughput(self, match):
42          throughput = 0
43          for i, output in enumerate(match):
44              if self.input_buffers[i][output] > 0:  # Check if there is at least one packet to send
45                  throughput += 1
46          return throughput
```

خط ۴۴: به ازای match شدن هر یک از ورودی‌های i به خروجی j، اگر بسته‌ای برای ارسال به خروجی j در jامین صف مجازی بافر ورودی i وجود داشته باشد یکی به تعداد بسته‌هایی که به ازای این match می‌تواند ارسال شود افزوده می‌شود.

مصورسازی

```python
def plot_throughput_vs_ports(num_time_slots):
    port_numbers = [4, 5, 6, 7, 8]
    average_throughputs = []  #

    for num_ports in port_numbers:
        switch = Switch(num_ports)
        average_throughputs.append(switch.run_simulation(num_time_slots))

    # Plotting the results
    plt.figure(figsize=(10, 6))
    plt.plot(port_numbers, average_throughputs, marker='o', linestyle='-', color='b')
    plt.title('Average Throughput vs. Number of Ports')
    plt.xlabel('Number of Ports')
    plt.ylabel('Average Throughput')
    plt.xticks(port_numbers)
    plt.grid(True)
    plt.show()
```

طبق خواسته‌ی دستور کار پروژه میانگین گذردهی برای تعداد پورت ۵ و ۶ و ۷و ۸و ۹ روی نمودار رسم شده است. و نتیجه به ازای تعداد time-slot = 31035 به شرح زیر است: