گزارش پروژه درس طراحی الگوریتم دکتر باقری

امید سیفان

نحوه‌ی کارکرد برنامه

برنامه در اول کل فایل ورودی را می‌خواند. سپس خط به خط با فرض نداشتن جهت وارد لیست همسایگی می‌شود. برای لیست همسایگی از HashMap<Integer, HashMap<Integer, Edge>> استفاده شده است. دلیل انتخاب آن قابلیت دسترسی با $O(1)$ به کلید هاست.

هر Edge شامل ۲ راس و ۱ نمره‌ی یال هست که در اول برابر صفر گذاشته شده است. سپس نمره‌ی همه‌ی یال ها محاسبه می‌شود. برای محاسبه درجه هر راس کافی است که سایز HashMap مربوطه را به دست آوریم. برای محاسبه Z هم کافی است که لیست همسایه های رئوس i و j را اشتراک بگیریم. سپس لیست همه‌ی یال ها را به دست می‌آوریم که زمان محاسبه $O(n)$ دارد (البته که با ساختمان داده‌ی جدا برای نگهداری یال ها می‌توانیم زمان را خطی کنیم).

با استفاده از الگوریتم ورودی یال ها را مرتب می‌کنیم و یال با کوچکترین نمره را انتخاب می‌کنیم. پس از آن از HashMap ها یال موردنظر را حذف می‌کنیم. سپس راس های مجاور یال حذف شده را به دست می‌آوریم و نمره یال هایشان را به‌روز می‌کنیم (می‌توان برای بهبود، فقط یال های مجاور یال حذف شده را به‌روز کرد).

سپس با DFS چک می‌کنیم که پیماش تا چند راس امکان پذیر است. اگر تعداد آن برابر کل رئوس بود، الگوریتم تمام نشده و باید مراحل را دوباره تکرار کنیم. پس از اینکه الگوریتم تمام شد، رئوسی که توانستیم با DFS پیماش کنیم را داخل گروه A و بقیه را داخل گروه B میریزیم و در فایل خروجی ذخیره می‌کنیم.

چالش ها

۱- برای بهبود سرعت محاسبات و کش کردن نتایج از Java ۸ و Stream ها استفاده شده است. برای مثال:

```
private List<Integer> getAllNeighbors(int i, int j) {
    List<Integer> result =
            Stream.concat(
                    this.adjacency.get(i).values().stream(),
                    this.adjacency.get(j).values().stream()
            ) Stream<Edge>
            .map(x -> x.to) Stream<Integer>
            .distinct()
            .collect(Collectors.toList());
    result.add(i);
    result.add(j);
    return result;
}
```

۲- سرعت QuickSort به شدت پایین بود که دلیل آن برابر بودن تعداد زیادی از نمره های یال ها بود. برای حل آن، در صورتی که مقدار مورد بررسی با پاشنه برابر بود، با احتمال ۵۰درصد در دسته بندی های متفاوت قرار می‌گیرد.

```
if (arr[j].score <= pivot.score) {
    i++;

    Edge swapTemp = arr[i];
    arr[i] = arr[j];
    arr[j] = swapTemp;
}
```

```
if (arr[j].score < pivot.score || (arr[j].score == pivot.score && Sorts.rand.nextInt( bound: 2) == 1)) {
    i++;

    Edge swapTemp = arr[i];
    arr[i] = arr[j];
    arr[j] = swapTemp;
}
```

۳- در گـام اول QuickSort بـه صـورت بـازگشـتی پیاده شـده بـود که بـه مشکل StackOverFlow بـر می‌خورد و برای درست شدن این مشکل از Stack استفاده شد که باند های بالا و پایین مرتب سازی را در آن ذخیره و برداشت می‌کنیم. راه حل نهایی به شکل زیر پیاده شد:

```
private static void quickSort(Edge[] arr) {
    Stack<Integer> stack = new Stack<>();
    stack.push( item: 0);
    stack.push( item: arr.length - 1);

    while (!stack.isEmpty()) {
        int end = stack.pop();
        int begin = stack.pop();

        if (end - begin < 2) {
            continue;
        }

        int partitionIndex = partition(arr, begin, end);

        if (partitionIndex - 1 > begin) {
            stack.push(begin);
            stack.push( item: partitionIndex - 1);
        }

        if (partitionIndex + 1 < end) {
            stack.push( item: partitionIndex + 1);
            stack.push(end);
        }
    }
}
```

خروجی تست کیس ها

خروجی های زیر با MergeSort پیاده شده اند.

```
DFS: 10000
Deleting Edge{from=2987, to=6475, score=0.125}
Neighbors: 26
Neighbor Edges: 561
[Iteration 5446] - Elapsed: 0.041292511s
DFS: 10000
Deleting Edge{from=2988, to=9477, score=0.125}
Neighbors: 47
Neighbor Edges: 947
[Iteration 5447] - Elapsed: 0.034775698s
DFS: 10000
Deleting Edge{from=2990, to=7793, score=0.125}
Neighbors: 31
Neighbor Edges: 816
[Iteration 5448] - Elapsed: 0.034570639s
DFS: 9989
Saved Output
```

تست کیس ۱

```
DFS: 10000
Deleting Edge{from=622, to=4201, score=0.07142857142857142}
Neighbors: 37
Neighbor Edges: 848
[Iteration 11461] - Elapsed: 0.112904308s
DFS: 10000
Deleting Edge{from=623, to=4861, score=0.07142857142857142}
Neighbors: 38
Neighbor Edges: 670
[Iteration 11462] - Elapsed: 0.102759578s
DFS: 10000
Deleting Edge{from=624, to=5575, score=0.07142857142857142}
Neighbors: 40
Neighbor Edges: 1304
[Iteration 11463] - Elapsed: 0.106215068s
DFS: 10000
Deleting Edge{from=625, to=4412, score=0.07142857142857142}
Neighbors: 37
Neighbor Edges: 639
[Iteration 11464] - Elapsed: 0.11027531s
DFS: 9979
Saved Output
```

تست کیس ۲

```
DFS: 50000
Deleting Edge{from=17433, to=47583, score=0.125}
Neighbors: 47
Neighbor Edges: 1020
[Iteration 28564] - Elapsed: 0.220426147s
DFS: 50000
Deleting Edge{from=17434, to=49492, score=0.125}
Neighbors: 54
Neighbor Edges: 979
[Iteration 28565] - Elapsed: 0.249083516s
DFS: 50000
Deleting Edge{from=17435, to=39358, score=0.125}
Neighbors: 31
Neighbor Edges: 464
[Iteration 28566] - Elapsed: 0.238349869s
DFS: 50000
Deleting Edge{from=17439, to=19309, score=0.125}
Neighbors: 21
Neighbor Edges: 322
[Iteration 28567] - Elapsed: 0.266524116s
DFS: 50000
Deleting Edge{from=17440, to=49314, score=0.125}
Neighbors: 53
Neighbor Edges: 943
[Iteration 28568] - Elapsed: 0.213323183s
DFS: 49989
Saved Output
```

تست کیس ۳

مقایسه سرعت ها

	تست کیس ۱	تست کیس ۲	تست کیس ۳
خواندن فایل	۰.۴۱۲۷۱۶۰۸۶	۰.۵۷۱۳۴۷۲۲۳	۱.۰۹۵۲۱۱۵۹
محاسبه اولیه نمره یال ها	۰.۴۴۳۲۴۸۸۰۸	۰.۷۹۴۳۵۷۱۸	۱.۸۲۳۸۰۲۴۸۸
BubbleSort	۱۷۴.۷۲۷۲۷۷۹۹	۶۴۴.۲۰۱۹۳۱۰۵۶	بالای ۸۰ دقیقه
InsertionSort	۶۱.۸۴۸۴۷۹۷۳۱	۲۵۲.۳۶۷۱۰۲۵۸	۲۷۷۵.۶۷۴۱۰۸۸۳۳
MergeSort	۰.۰۴۵۳۷۹۴۴۲	۰.۰۸۰۸۸۳۸۳۳	۰.۲۴۸۰۷۷۷۲۷
QuickSort	۱۱.۱۹۶۷۰۷۷۴۶	۶.۹۹۷۸۲۴۶۰۲	۶۶۰.۵۷۴۳۳۷۳۹۶
QuickSort(Optimized)	۰.۰۶۹۹۱۷۳۰۵	۰.۱۰۹۱۵۶۴۹۲	۰.۴۴۹۹۹۱۷۷۹

با توجه به اینکه پیچیدگی زمانی BubbleSort و InsertionSort هر کدام در بهترین حالت $O(n)$ و برای MergeSort و QuickSort هر کدام در بهترین حالت $O(nlogn)$ است، نتایج منطقی است.