

# توضیحات پروژه

## بخش اول

در این بخش هدف رنگ آمیزی فرش است بطوریکه حداقل رنگ استفاده شود و هیچ دو ناحیه ای هم رنگ نباشد. در این بخش ما ناحیه های فرش را به راس های یک گراف تشبیه کردیم و همسایگی دو یال را، به وجود یک یال بین این ناحیه تعبیر کردیم. اینگونه میتوان این مسئله را با کمک گرفتن از الگوریتم رنگ آمیزی گراف حل کرد. در فایل graphColor.txt ابتدا سایز گراف (تعداد راس ها) گرفته میشود و سپس در هر خط هر دو یالی که باهم همسایه اند نوشته میشوند.

## بخش دوم

برای پیدا کردن فرش های مشابه به هم از الگوریتم Sequence alignment استفاده شده در کلاس Sequence alignment دو ارایه به عنوان ورودی وارد می شوند که با هم مقایسه می شوند در نهایت یک نمره از نظر شباهت محاسبه می شود و بازگشت داده می شود. به طوری که اگر دو ایتیم با هم برابر باشد ۱ به کل نمره اضافه می شود و اگر نباشد ۱ از نمره کم می شود

```
this.MATCH = 1;
this.MISMATCH = -1;
this.INDEL = -1;
```

در متد findSolution یک ماتریس ساخته می شود به طوری که یکی از ارایه ها به عنوان col ها و یکی به عنوان row ها انتخاب می شود و اولین row and col امتیاز دهی می شوند به طوری که هر مدام یکی کمتر از ایتیم قبلی خود هستند

```
int[][] solution = new int[strand1.length + 1][strand2.length + 1];
solution[0][0] = 0;

for (int i = 1; i < strand2.length + 1; i++) {
    solution[0][i] = solution[0][i - 1] + INDEL;
}

for (int i = 1; i < strand1.length + 1; i++) {
    solution[i][0] = solution[i - 1][0] + INDEL;
}
```

بعد تمام ایتیم های دو ارایه ورودی را با هم مقایسه می کنیم و بیشترین مقدار

• Position to the left -1

• Position above -1

Position top-left + 1 •

را در ماتریس جواب قرار می دهیم

```
for (int i = 1; i < strand1.length + 1; i++) {
    for (int j = 1; j < strand2.length + 1; j++) {

        int matchValue;

        if (strand1[i - 1] == strand2[j - 1])
            matchValue = MATCH;
        else
            matchValue = MISMATCH;

        solution[i][j] = max(solution[i][j - 1] + INDEL, solution[i - 1][j] + INDEL,
                               solution[i - 1][j - 1] + matchValue);
    }
}
```

در نهایت نمره شباهت این دو در خانه

```
solution[solution.length - 1][solution[0].length - 1]
```

قرار می گیرد

چون نقشه فرش ها یک ماتریس است با استفاده یک الگوریتم آنها را به یک آرایه یک بعدی تبدیل می کنیم

## بخش سوم

### گزارش کار الگوریتم

بخش منطقی الگوریتم در کلاس BuyCarpet و بخش ارتباط با کاربر در کلاس App و در فانکشن buyCarpet نوشته شده است. در ادامه توابع کلاس BuyCarpet توضیح داده خواهد شد.

### تابع min

در این تابع یک آری لیست از موجودیت carpet داده میشود و به عنوان خروجی شی ای از نوع carpet که کمترین قیمت را دارد برگردانده میشود. ابتدا در متغیر answer اولین شی که در آری لیست است ذخیره می شود در این بخش کاربر به ما مقدار پولی که می تواند برای خرید فرش بپردازد را می دهد و ما باید بیشترین فرشی که کاربر میتواند خریداری کند را به او اعلام کنیم. همانطور که در داک پروژه نوشته شده در حل این بخش از الگوریتم کوله پشتی استفاده شده است. در الگوریتم کوله پشتی دو مقدار wieght و price داریم، ولی اینجا فقط مقدار weight که همان پول وارد شده توسط کاربر است را داریم. در نتیجه نیازی به ماتریس نیست و می توان با یک آرایه یک بعدی به جواب رسید. به طور خلاصه الگوریتم

نوشته در BuyCarpet با استفاده از فرش های موجودی که در لیست آرایه ای carpets ذخیره شده و همچنین پول دریافتی از کاربر گرفته میشود لیستی از فرش هایی که مجموع قیمتشان کمتر مساوی پول کاربر است را آرایه می دهد. لیست فرش ها را میتوان داخل فایل carpet.txt مشاهده کرد. این الگوریتم بترتیب از ارزان ترین فرش ها تا گرانترین فرش ها پیش میرود و تا وقتی مجموع قیمت فرش های انتخاب شده به دارایی کاربر نرسد هربار ارزان ترین فرش باقی مانده در لیست فرش ها به غیر از فرش های انتخاب شده را به لیست اضافه میکند.

## بخش چهارم

در این بخش برای پیدا کردن مسیر از الگوریتم floyed استفاده شده است. در کلاس FindPath این الگوریتم پیاده سازی شده است

سازنده کلاس آرایه های مورد نیاز را برای این الگوریتم ایجاد می کند

```
FindPath(int V, int[][] graph) {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            dis[i][j] = graph[i][j];

            if (graph[i][j] == INF)
                Next[i][j] = -1;
            else
                Next[i][j] = j;
        }
    }
    floyd(V);
}
```

در اخر سازنده متد floyd فراخوانی می شود تا ماتریس Next و Des کامل شود

```
private void floyd(int V) {
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dis[i][k] == INF ||
                    dis[k][j] == INF)
                    continue;

                if (dis[i][j] > dis[i][k] +
                    dis[k][j]) {
                    dis[i][j] = dis[i][k] +
                        dis[k][j];
                    Next[i][j] = Next[i][k];
                }
            }
        }
    }
}
```

```

    }
}
} k

```

و متد printPath برای پرینت فاصله بین دو نقطه است که در با یم حلقه که چک می کنند مه ایا به نقطه مورد نظر رسیده است یا خیر و در همین حین خانه هایی که رد کرده را چاپ می کند

```

void printPath(int u,
    int v) {
    if (Next[u][v] == -1)
        return;

    System.out.print(u + " -> ");

    while (u != v) {
        u = Next[u][v];
        System.out.print(u + " -> ");
    }
    System.out.println("END");
}

```

مرتبه این الگوریتم از  $O(n^3)$  است

سایت هایی که از انها ایده گرفته شد:

- <https://www.geeksforgeeks.org/>
- [https://en.wikipedia.org/wiki/Sequence\\_alignment](https://en.wikipedia.org/wiki/Sequence_alignment)
- <https://globin.bx.psu.edu/courses/fall2001/DP.pdf>