به نام خدای رنگین کمان


دانشگاه اصفهان

دانشکده مهندسی کامپیوتر


مستند پروژه نهایی درس «**طراحی الگوریتم**»

مدرس:جناب آقای دکتر پیمان ادیبی

# «کارخانه فرش بافی»


گرداوردندگان:

محمد حسینی

بهنود عبودیت


نیمسال دوم : 1401-1402

# کارخانه فرش بافی

یک کارخانه فرش بافی نیاز به سامانه برای مدیریت و مکانیزه کردن کارهای کارخانه دارند. این سامانه شامل بخش مختلفی همچون طراحی، فروش،توزیع و ... میشود.

### صفحه اول رابط کاربری

```
**Carpet-pet-pet System**

Menu:
1. Design New Carpet
2. List of Carpets
3. Search Carpet (By Map Pattern)
4. Buy Carpet (Based on the Amount of Money)
5. Directions to the Nearest Factory Store
0. Exit

Enter Index=>
```

- **طراحی:**

خانه قصد دارد فرشهای جدیدی وارد بازار کند از این رو می خواهد نقشه های جدیدی طراحی کند. آنها قصد دارند که فرشهایی ببافند که از اشکال هندسی تشکیل شده اند و این اشکال هندسی با یکدیگر همسایه هستند.هر کدام از این اشکال هندسی به یک رنگ است. هرچه تعداد رنگهای به کار رفته برای بافت فرش کمتر باشد هزینه فرش بافته شده کمتر است.از این جهت،کارخانه از شما میخواهد با دریافت اطلاعاتی از نواحی فرش،کمترین تعداد رنگ مورد نیاز و همچنین رنگ انتساب داده شده به هر یک از نواحی را برای آنها بیابید

💡 ما در این بخش از الگوریتم «Graph Coloring» استفاده کردیم.

پیچیدگی زمانی (time complexity) کد ارائه شده O(numOfPatterns^2)  است.
پیچیدگی حافظه:

1. آرایه‌ی 'result' اندازه‌ای برابر با numOfPatterns دارد، بنابراین O(numOfPatterns) حافظه استفاده می‌کند.
2. آرایه‌ی 'availableForPaint' نیز اندازه‌ای برابر با numOfPatterns دارد، بنابراین O(numOfPatterns) حافظه استفاده می‌کند.
3. حافظه‌ای که توسط شیء 'newCarpet' و ساختارهای داده مرتبط آن استفاده می‌شود، بستگی به جزئیات پیاده‌سازی کلاس 'Carpet' دارد.  بنابراین، نمی‌توانیم میزان دقیق حافظه مصرفی را برای این بخش تعیین کنیم.
4. لیست 'listOfVertices' از 'newCarpet' رئوس را ذخیره می‌کند و ویژگی 'color' را به هر رأس اضافه می‌کند. اگر تعداد رئوس همچنین برابر با numOfPatterns باشد، در این صورت حافظه مورد استفاده برای این لیست برابر با O(numOfPatterns) است.

ورودی:

عنوان ورودی، مشخص میکند که هر کدام از نواحی با کدام یک از نواحی دیگر همسایه است.

```
Enter Index=> 1

*Design Panel*
Enter the Price of Your Carpet=> 100
Number of Carpet's Patterns=> 5
Enter Relations Between Patterns.
[Patterns=>0...n-1] [Relations=> Ex: 0 5, And 0 For Exit]
Relation 1=> 0 1
Relation 2=> 0 2
Relation 3=> 1 2
Relation 4=> 1 3
Relation 5=> 2 3
Relation 6=> 3 4
Relation 7=> 0
```

خروجی:

حداقل تعداد رنگ مورد نیاز رنگ انتساب داده شده به هر یک از نواحی را به کاربر نشان میدهد.

```
Designing...
Your New Designed Carpet:
Price=> 100.0$
Pattern 0 --->  Color 0
Pattern 1 --->  Color 1
Pattern 2 --->  Color 2
Pattern 3 --->  Color 0
Pattern 4 --->  Color 1
```

## - فروش:

- **خرید بر اساس میزان پول**

این سامانه کاربر میتواند با وارد کردن حداکثر میزان پولی که دارد بیشترین تعداد فرشی که می تواند بخرد رابیابد. شما باید تمهیداتی بیاندیشید که با سریعترین روش ممکن کاربر بتواند این کار را عملی کند.

💡 ما در این بخش از الگوریتم «Knapsack Problem» استفاده کردیم.

پیچیدگی زمانی (time complexity) کد ارائه شده به صورت بازگشتی حساب می‌شود. زیرا تابع `maxNumOfCarpet` خود را به صورت بازگشتی فراخوانی می‌کند.

به طور کلی، می‌توان گفت که پیچیدگی زمانی این کد برابر است با O(2^n)، که n برابر با سایز `carpets` است. این به این معنی است که زمان اجرای کد به طور نمایی با افزایش تعداد اعضای (carpets) افزایش می‌یابد. زیرا هر مرحله از بازگشت، دو بار دیگر بازگشت صدا زده می‌شود که تعداد اعضای آرایه را یک واحد کاهش می‌دهد.

در مورد حافظه مصرفی، از تابع‌های بازگشتی استفاده می‌شود، اما آرایه `prices` با اندازه‌ی `carpets` ایجاد می‌شود و مجموعه 'resultOfCarpets' از نوع Set ایجاد می‌شود. بنابراین، حافظه مصرفی به طور تقریبی برابر است با O(n)، که n تعداد اعضای Set است.

ورودی:

کاربر مقدار پول خود را وارد میکند.

```
*Buy Carpet*
Enter Your Budget=> 1200
```

خروجی:

لیستی از فرشهایی که کاربر میتواند بخرد را نمایش میدهد.

```
Processing...
*Receipt*

#Carpet1:
Price=> 300.0$
Pattern 0 --->  Color 0
Pattern 1 --->  Color 1
Pattern 2 --->  Color 2
Pattern 3 --->  Color 0
#Carpet2:
Price=> 500.0$
Pattern 0 --->  Color 0
Pattern 1 --->  Color 1
Pattern 2 --->  Color 0
Pattern 3 --->  Color 2
Pattern 4 --->  Color 2
Pattern 5 --->  Color 1
#Carpet3:
Price=> 200.0$
Pattern 0 --->  Color 0
Pattern 1 --->  Color 1
Pattern 2 --->  Color 2
Pattern 3 --->  Color 0
Pattern 4 --->  Color 3
```

```
#Carpet4:
Price=> 100.0$
Pattern 0 --->  Color 0
Pattern 1 --->  Color 1
Pattern 2 --->  Color 0
Pattern 3 --->  Color 2
Pattern 4 --->  Color 0

Total Price=> 1100.0$
Remaining Budget=> 100.0$
```

- **مسیریابی به نزدیکترین فروشگاه کارخانه**

که کارخانه شعبات زیادی دارد این سامانه دارای بخشی است که کاربر می تواند با وارد کردن مختصات خود،نزدیکترین شعبه به خود را بیابد و مسیر رفتن به آن نقطه را پیدا کند. شهر فرضی ما شامل چه راه هایی است که به یکدیگر متصل هستند. این نقاط به همراه خیابانهای بین آنها از قبل به سیستم معرفی می شوند

💡 ما در این بخش از الگوریتم «Dijkstra's Algorithm» استفاده کردیم.

پیچیدگی زمانی (time complexity) کد ارائه شده برابر است با O(V^2)، که V تعداد رئوس گراف است. این مربوط به حلقه‌های دوتایی است که در کد وجود دارند. حلقه بیرونی به تعداد V بار اجرا می‌شود و حلقه داخلی نیز در بدترین حالت نیز به تعداد V بار اجرا می‌شود.

در مورد حافظه مصرفی، متغیرهای آرایه‌ای 'sptSet'، 'minDistances'، و 'parents' هر کدام با اندازه‌ی V ایجاد می‌شوند. بنابراین، حافظه مصرفی به طور تقریبی برابر است با O(V)، که V تعداد رئوس گراف است.

ورودی:

کاربر در شعبه ای که در ان قراردارد را به عنوان مختصات مبدا خود مشخص میکند.

```
*Nearest Store*
Enter Number of Stores=> 9
Enter Adjacency Matrix(Graph) of Stores:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0
Enter Source=> 0
```

خروجی:

آدرس نزدیکترین شعبه مسیر رسیدن به آن را به کاربر نشان میدهد. به این معنا که از کدام راه ها به وسیله کدام یالها به مقصد میرسد.

```
Processing...
The Nearest Store=> 1 [With 4 Distance]
Path=> 0->1->Done
Minimum Distances from Stores=> [0, 4, 12, 19, 21, 11, 9, 8, 14]
```

🧠 توضیحات بیشتر در ارائه ی حضوری پروژه به سمع و نظر علاقمندان میرسد.

🔗 لینک ریپازیتوری پروژه:

https://github.com/1mimhe/AlgorithmsProject-401_2-CarpetWeavingFactory