MAY 25, 2023

مستند پروژه درس طراحی الگوریتم دکتر ادیبی

کارخانه فرشبافی کوهدل

صابر سبزی

دانشکده مهندسی کامپیوتر دانشگاه اصفهان

مقدمه

آنچه در این پروژه خواهید دید و هدف آن

یک کارخانه فرشبافی نیاز به سامانه برای مدیریت و مکانیزه کردن کارهای کارخانه دارند. این سامانه شامل بخش مختلفی همچون طراحی، فروش، توزیع و ... میشود. هر بخش از این سامانه به طور مجزا توضیح داده شده است.در این پروژه الگوریتم های متفاوتی استفاده شده است که در طول ترم آموختیم. در این پروژه با الگوریتم های رنگ کردن گراف انواع سورت و جستجو و الگوریتم کوله پشتی , دایکسترا و فلوید نیز آشنا میشویم

در بخش اول به طراحی فرشها میپردازیم : 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 نواحي را مشخص کنیم :

```
--- KOOHDEL CARPET SHOP ----
*****************************
* 1 : DESIGN CARPETS    --> *
* 2 : SALE PART         --> *
* 3 : PROGRAMMER INFO   --> *
* 4 : EXIT !            --> *
*****************************
PLEASE ENTER YOUR CHOICE-->:
```

```
Enter The Number Of Carpet Rows
3
Enter The Number Of Carpet Columns
3
Enter Carpet Matrix Data :
1 0 0
1 0 1
0 1 0
Minimum number of colors required for carpet: 2
Colors assigned to regions:
Region 1: Color 1
Region 2: Color 2
Region 3: Color 1
```

در اینجا با وارد کردن سطر و ستون های ماتریکس ما که همان فرش است دیتای مورد نیاز آن را میدهیم سپس برنامه به ما حداقل رنگ مورد نیاز و رنگ هر یک از نواحي را مشخص میکند,

در ادامه توضیح میدهیم که کد آن چگونه نوشته شده و از چه الگوریتمی برای آن استفاده شده است :

```java
class GraphColoring {
    // تابع رنگ کردن گراف
    1 usage  shahinsabzi
    public static int[] graphColoring(int[][] graph) {
        int n = graph.length; // تعداد گره‌ها (نواحی)
        int[] colors = new int[n]; // آرایه‌ای برای نگه‌داری رنگ‌های اختصاص داده شده به گره‌ها

        // آرایه‌ای برای نگه‌داری رنگ‌های مجازی هر گره
        int[] availableColors = new int[n];
        Arrays.fill(availableColors, val: 0);

        // اختصاص رنگ به گره اول
        colors[0] = 1;

        // اختصاص رنگ به سایر گره‌ها
        for (int i = 1; i < n; i++) {
            Arrays.fill(availableColors, val: 0); // پاک کردن رنگ‌های مجازی

            // پیدا کردن رنگ‌های مجازی همسایه‌های گره فعلی و اضافه کردن آن‌ها به آرایه رنگ‌های مجازی
            for (int j = 0; j < n; j++) {
                if (graph[i][j] == 1 && colors[j] != 0) {
                    availableColors[colors[j] - 1] = 1;
                }
            }

            // پیدا کردن اولین رنگ مجازی برای گره فعلی
            // پیدا کردن اولین رنگ مجازی برای گره فعلی
            int color = 1;
            for (int j = 0; j < n; j++) {
                if (availableColors[j] == 0) {
                    color = j + 1;
                    break;
                }
            }

            colors[i] = color; // اختصاص رنگ به گره فعلی
        }

        // پیدا کردن تعداد رنگ‌های مجازی استفاده شده برای رنگ کردن گراف
        int numColors = 0;
        for (int i = 0; i < n; i++) {
            if (colors[i] > numColors) {
                numColors = colors[i];
            }
        }
        // برگرداندن تعداد کمترین رنگ‌های مورد نیاز و رنگ‌های اختصاص داده شده به هر یک از نواحی
        int[] result = new int[n + 1];
        result[0] = numColors; // تعداد کمترین رنگ‌های مورد نیاز
        for (int i = 0; i < n; i++) {
            result[i + 1] = colors[i]; // رنگ اختصاص داده شده به هر یک از نواحی
        }
        return result;
    }
}
```

ما در اینجا یک کلاس به اسم رنگ کردن گراف ساخته ایم و در آن آرایه ای برای تخصص رنگ های اختصاص داده شده به گره ها ساختیم سپس به اولین گره رنگ پیشفرضی دادیم با استفاده از حلقه ها و الگوریتم رنگ گراف کمترین رنگ مورد نیاز و رنگ هر یک از نواحي را مشخص کردیم و ریترن کردیم . پیچیدگی زمان کدی که برای این بخش از پروژه استفاده کردیم ان به توان دو است .

.است. برای تحلیل این مرتبه زمانی، میتوان ابتدا به خطوط کد دقت کرد O(n^2) در کد فوق، به صورت graphColoring مرتبه زمانی تابع
تعریف شده که برای نگهداری رنگهای اختصاص داده شده به colors قرار داده میشود. سپس یک آرایه به نام graph برابر با طول آرایه n ابتدا، متغیر
با طول برابر با تعداد گرهها تعریف میشود که برای نگهداری رنگهای مجاز در هر availableColors گرهها استفاده میشود. همچنین، یک آرایه دیگر به نام
.با مقدار صفر استفاده شده است availableColors برای پر کردن آرایه Arrays.fill گره استفاده میشود. در اینجا، از تابع
بار اجرا میشود. در هر n-1 از دومین گره شروع شده و به تعداد for پس از تعریف متغیرها، رنگ اول به گره اول اختصاص داده میشود. در ادامه، یک حلقه
با availableColors پاک شده و سپس برای هر همسایه گره فعلی، رنگ مربوط به آن همسایه در آرایه availableColors مرحله از این حلقه، ابتدا آرایه
مشخص میشود. در انتها، رنگ availableColors مقدار 1 مشخص میشود. سپس رنگ اولیه مجاز برای گره فعلی، با پیدا کردن اولین رنگ مجاز در آرایه
.پیدا شده به گره فعلی اختصاص داده میشود
دیگر، تعداد کمترین رنگهای مورد نیاز برای رنگ کردن گراف پیدا میشود و همچنین رنگهای اختصاص داده شده به هر for در انتها، با استفاده از یک حلقه
.یک از نواحی در یک آرایه جدید ذخیره میشوند
اگر در نظر بگیریم که در هر مرحله از حلقه دوم، یک حلقه دیگر وجود دارد که برای هر گره، همه همسایههای آن بررسی میشوند، میتوان گفت که پیچیدگی
بار اجرا میشود. بنابراین، تعداد کل عملیات انجام n بار اجرا میشود و حلقه دوم هم در هر بار اجرا، به n است. چرا که حلقه اول O(n^2) ،زمانی این تابع
شده در این تابع، برابر با
.است n^2

در بخش دوم پروژه که مربوط به بخش فروش است ابتدا به سراغ یافتن فرش مورد نظرمان بر اساس جستجو میان فرش های موجود (نقشه چند فرش کوچک ۳ در ۳) به طور پیشفرض در برنامه است , می‌رویم و درصد شباهت آن ها را پیدا میکنیم :

```
-------KOOHDEL CARPET SHOP------
*********************************
* 1 : SEARCH (BASED ON MAP)--> *
* 2 : BUY (BASED ON MONEY) --> *
* 3 : CLOSEST MARKET       --> *
* 4 : FIRST MENU           --> *
*********************************
PLEASE ENTER YOUR CHOICE: -->
```

```
Enter the dimensions of the carpet map (rows, columns):
3 3
Enter the elements of the carpet map (one row at a time):
1 0 1
1 1 0
0 1 0
Similarity with map 2: 44.721359549995796
1 1 0
0 0 1
0 1 0

Similarity with map 1: 59.999999999999986
1 0 1
0 1 0
1 0 1

Similarity with map 3: 59.999999999999986
1 1 1
1 0 1
0 0 0
```

ادامه توضیح می‌دهیم که کد آن چگونه نوشته شده و از چه الگوریتمی برای آن استفاده شده است :

```
class CarpetFactory {
    1 usage  shahinsabzi
    public static void main() throws Exception {...}

    // ----------------------------------------------------
    1 usage  shahinsabzi
    private static int[][] getInputMap() {...}

    // ----------------------------------------------------
    1 usage  shahinsabzi
    private static void printMap(int[][] map) {...}

    // ----------------------------------------------------
    1 usage  shahinsabzi
    public static Integer[] getSortedIndices(double[] arr) {...}

    // ----------------------------------------------------
    1 usage  shahinsabzi
    private static double calculateSimilarity(int[][] map1, int[][] map2) {...}

    // ----------------------------------------------------
    1 usage  shahinsabzi
    private static int[][][] getAvailableMaps() {...}


     shahinsabzi
    public static int[][] createRandomMatrix() {...}
}
// ----------------------------------------------------
```

تصویر بالا کلاسی که برای این بخش از پروژه ساخته شده است می‌بینید. تابع های مختلفی برای آن ساخته شده از جمله پرینت و نمایش نقشه ها, سورت کردن و گرفتن نقشه های موجود و ساخت نقشه های رندوم و تصادفی ۳۰۰ در ۴۰۰ همچنین از تابع گرفتن ورود برای نقشه هایی

که کاربر وارد می‌کند . حال, شکل زیر تابعی است است که برای پیدا کردن میزان شباهت نقشه های فرش ها کد آن زد شده است :

```java
private static double calculateSimilarity(int[][] map1, int[][] map2) {
    int dotProduct = 0;
    int norm1 = 0;
    int norm2 = 0;

    for (int i = 0; i < map1.length; i++) {
        for (int j = 0; j < map1[0].length; j++) {
            dotProduct += map1[i][j] * map2[i][j];
            norm1 += map1[i][j] * map1[i][j];
            norm2 += map2[i][j] * map2[i][j];
        }
    }

    double similarity = dotProduct / (Math.sqrt(norm1) * Math.sqrt(norm2));

    return similarity * 100;
}
```

در اینجا با گرفتن آرایه ای از دو نقشه موجود و ورودی در دو حلقه تو در تو درایه های هر کدامشان و میزان شباهتشان چک میشود . .پیچیدگی زمان کدی که برای این بخش از پروژه استفاده کردیم ان به توان دو است .

است. به دلیل دو حلقه تو در تویی که بر روی O(n^2) ،دو ماتریس ورودی n در n زمان اجرای این کد با ابعاد می‌شود. به علاوه، تعداد عملیات‌هایی که باید n^2 ابعاد ماتریس‌ها اجرا می‌شوند، تعداد کل عملیات‌ها برابر با است. بنابراین، مرتبه O(n^2) انجام شود، نیز در حد magnitudes و dot product برای محاسبه مجموع .است O(n^2) زمانی کل الگوریتم برابر با ان به توان دو است

در بخش سوم پروژه که مربوط به بخش فروش و شاخه خرید است ابتدا میزان دارایی و بوجه مان را وارد میکنیم سپس برای کاربر تعداد فرش و اینکه از کدام فرش می‌تواند خرید کند و برایش سود بیشتری دارد نمایش داده می‌شود.

```
-------KOOHDEL CARPET SHOP------
*********************************
* 1 : SEARCH (BASED ON MAP)--> *
* 2 : BUY (BASED ON MONEY) --> *
* 3 : CLOSEST MARKET       --> *
* 4 : FIRST MENU           --> *
*********************************
PLEASE ENTER YOUR CHOICE: -->
2
Enter your budget: ---> 500
You can afford the following carpets:
Carpet f- Price: 300 Area: 8
Carpet C- Price: 200 Area: 4
```

حال در ادامه کدی که برای این بخش از پروژه استفاده کرده ایم می‌بینیم:

```java
public static class Carpet {

    2 usages
    private String name;
    3 usages
    private int price;
    3 usages
    private int area;

    //---------------------------
    18 usages  shahinsabzi
    public Carpet(String name, int area, int price) {...}

    //---------------------------
     shahinsabzi
    public void setPrice(int price) { this.price = price; }

    //---------------------------
    3 usages  shahinsabzi
    public int getPrice() { return price; }

    //---------------------------
     shahinsabzi
    public void setArea(int area) { this.area = area; }

    //---------------------------
    2 usages  shahinsabzi
    public int getArea() { return area; }

    //---------------------------
```

برای این بخش برای فرش های موجود در فروشگاه یک کلاس ساخته ایم که , اتربیوت های نام و قیمت و مساحت دارند و ستر گتر های آن ها نیز گذاشته شده است.

```java
public static List<Carpet> findAffordableCarpets(int money, List<Carpet> carpets) {
    int n = carpets.size(); int[] prices = new int[n];
    int[] values = new int[n];
    for (int i = 0; i < n; i++) {
        Carpet carpet = carpets.get(i);
        prices[i] = carpet.getPrice();
        values[i] = carpet.getArea();
    }
    //--------------------------
    int[][] dp = new int[n + 1][money + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= money; j++) {
            if (i == 0 || j == 0) {...} else if (prices[i - 1] <= j) {
                dp[i - 1][j] = values[i - 1];
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
    //--------------------------
    List<Carpet> affordableCarpets = new ArrayList<>();
    int j = money;
    for (int i = n; i > 0 && j > 0; i--) {
        if (dp[i][j] != dp[i - 1][j]) {
            Carpet carpet = carpets.get(i - 1);
            affordableCarpets.add(carpet);
            j -= carpet.getPrice();
        }
    }
    return affordableCarpets;
```

با تابع بالا با گرفتن ورودی میزان بوجه و اررلیستی از فرش های موجود در آرگومان همانند الگوریتم کوله‌پشتی عمل می‌کند:

پیمایش اولیه لیست فرش‌ها برای ساختن دو آرایه به نام قیمت و مساحت، به ترتیب. با توجه به اینکه هر فرش شامل دو ویژگی مساحت و قیمت است، بنابراین پیمایش اولیه لیست فرش‌ها در مرتبه O(n) انجام می‌شود.

ساز نوع دو بعدی ایجاد می‌شود که سطرهای آن نشان‌دهنده تعداد فرش‌هایی dp در قسمت بعدی، یک جدول به نام هستند که در نظر گرفته شده و ستونهای آن نیز نشان‌دهنده پولی است که می‌توان برای خرید فرش‌ها داشت. در این بخش را به صورت مجزا dp اجرا می‌شود که هر یک از خانه‌های جدول O(n * money) از الگوریتم، یک حلقه دیگر در مرتبه انتخاب دو خانه از خط قبلی و خانه قبلی همان ،dp بررسی کرده و مقدار آن را محاسبه می‌کند. در هر خانه از جدول ستون جاری صورت می‌گیرد. پیچیدگی زمانی این بخش از الگوریتم O(n * money) است.

اجرا می‌شود که لیستی از فرش‌هایی که با بیشترین مساحت ممکن قابل O(n) در قسمت آخر، یک حلقه دیگر در مرتبه برای انتخاب فرش‌ها استفاده می‌شود. پیچیدگی زمانی این dp خرید هستند را به دست می‌آورد. در این حلقه، از جدول بخش از الگوریتم نیز O(n) است.

بنابراین، پیچیدگی زمانی کل این الگوریتم به صورت O(n * money) است.

مسیریابی به نزدیکترین فروشگاه کارخانه از آنجایی که کارخانه شعبات زیادی دارد این سامانه دارای بخشی است که کاربر میتواند با وارد کردن مختصات خود، نزدیکترین شعبه به خود را بیابد و مسیر رفتن به آن نقطه را پیدا کند. شهر فرضی ما شامل چهارراه هایی است که به یکدیگر متصل هستند. این نقاط به همراه خیابان های بین آنها از قبل به سیستم معرفی میشوند. . ورودی: کاربر نزدیکترین چهارراه نقشه به خود را به عنوان مختصات خود مشخص میکند. خروجی: آدرس نزدیک ترین شعبه و مسیر رسیدن به آن را به کاربر نشان میدهد. به این معنا که از کدام راس ها و به وسیله کدام یال ها به مقصد میرسد.

```
Enter your current location :
4
Vertex      Distance from Source
0       21
1       22
2       14
3       9
4       0
5       10
6       12
7       13
8       16
the answer is : eachNode{vertexNum=10, name=5, isDepartment=true}--- KOOHDEL CARPET SHOP ----
**************************
```

در صفحه بعد شما نحوه پیاده‌سازی کد های این بخش را مشاهده خواهید کرد.

این کد الگوریتم دیکسترا را برای پیدا کردن کوتاهترین مسیر بین یک گره مبدا و تمام گرههای دیگر در یک گراف با وزندار اجرا میکند. ورودیهای این الگوریتم شامل ماتریس مجاورت گراف با وزندار و گره مبدا میباشد.

ایجاد میشود و هر گره دارای شماره گره (vertices) در ابتدا، یک لیست حاوی تمام گرها از گره مبدا میباشد. پیمایش اولیه لیست گرها در (distance) و فاصله (vertexNum) انجام میشود O(V) مرتبه.

ایجاد میشود که نشاندهنده (shortest path treeset) sptSet سپس، یک آرایه به نام گرههایی است که در کوتاهترین مسیر از گره مبدا قرار دارند. این آرایه در ابتدا همهی خانههایش به false تنظیم میشود. پیچیدگی زمانی این قسمت از الگوریتم O(V) است.

سپس، فاصله گره مبدا به خودش برابر با صفر قرار داده میشود. پیچیدگی زمانی این قسمت از الگوریتم O(1) است.

بار اجرا میشود. در هر مرحله از این حلقه، یک گره V-1 در بخش بعدی، یک حلقه به تعداد مشخص sptSet با فاصله کمترین مسیر از گره مبدا برای پردازش انتخاب میشود و در

```
public ShortestPath() {

    eachNode v0 = new eachNode( vertexNum: 0,  isDepartment: false,  name: 0);
    eachNode v1 = new eachNode( vertexNum: 1,  isDepartment: true,  name: 1);
    eachNode v2 = new eachNode( vertexNum: 2,  isDepartment: false,  name: 2);
    eachNode v3 = new eachNode( vertexNum: 3,  isDepartment: false,  name: 3);
    eachNode v4 = new eachNode( vertexNum: 4,  isDepartment: false,  name: 4);
    eachNode v5 = new eachNode( vertexNum: 5,  isDepartment: true,  name: 5);
    eachNode v6 = new eachNode( vertexNum: 6,  isDepartment: false,  name: 6);
    eachNode v7 = new eachNode( vertexNum: 7,  isDepartment: true,  name: 7);
    eachNode v8 = new eachNode( vertexNum: 8,  isDepartment: false,  name: 8);

    vertices.add(v0);
    vertices.add(v1);
    vertices.add(v2);
    vertices.add(v3);
    vertices.add(v4);
    vertices.add(v5);
    vertices.add(v6);
    vertices.add(v7);
    vertices.add(v8);
```

```
void dijkstra(int graph[][], int src) {
    ArrayList<eachNode> dist = vertices;
    Boolean sptSet[] = new Boolean[V];

    //...
    for (int i = 0; i < V; i++) {
        dist.get(i).vertexNum = Integer.MAX_VALUE;
        sptSet[i] = false;
    }

    // Distance of source vertex from itself is always 0
    dist.get(src).vertexNum = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        //...
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        //...
        for (int v = 0; v < V; v++)

            //...
            if (!sptSet[v] && graph[u][v] != 0
                    && dist.get(u).vertexNum != Integer.MAX_VALUE
                    && dist.get(u).vertexNum + graph[u][v] < dist.get(v).vertexNum)
                dist.get(v).vertexNum = dist.get(u).vertexNum + graph[u][v];
    }
```

شده و فاصله های گره های مجاور آن گره با اعمال تغییرات لازم، به روزرسانی می‌شوند. بار اجرا می‌شود. بنابراین پیچیدگی V این قسمت با استفاده از یک حلقه داخلی، به تعداد .است O(V^2) زمانی این قسمت از الگوریتم

فراخوانی می‌شود که فاصله هر گره از گره مبدا را چاپ printSolution در نهایت، یک تابع .اجرا می‌شود O(V) می‌کند. این بخش از الگوریتم نیز در مرتبه

.می‌باشد O(V^2) بنابراین، پیچیدگی زمانی کل این الگوریتم برابر با

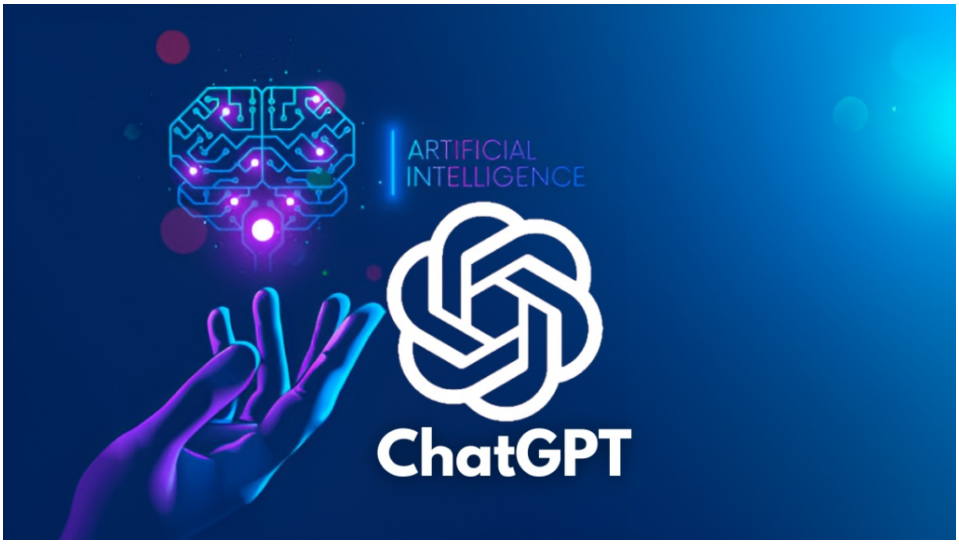
```java
class eachNode {
    14 usages
    int vertexNum;
    2 usages
    int name;
    3 usages
    Boolean isDepartment;

    9 usages  shahinsabzi
    public eachNode(int vertexNum, Boolean isDepartment, int name) {
        this.vertexNum = vertexNum;
        this.name = name;
        this.isDepartment = isDepartment;
    }

    shahinsabzi
    @Override
    public String toString() {
        return "eachNode{" +
                "vertexNum=" + vertexNum +
                ", name=" + name +
                ", isDepartment=" + isDepartment +
                '}';
    }
}
```

منابع

GeeksforGeeks



stack overflow