بسه تعالی

داکیومنت پروژه طراحی الگوریتم

استاد:دکتر ادیبی

اعضای گروه:

سهیل سلیمی
حسین آقایی

بهار ۱۴۰۱

توضیحات پروژه

بخش اول

در این بخش هدف رنگ آمیزی فرش است بطوریکه حداقل رنگ استفاده شود و هیچ دو ناحیه ای همرنگ نباشد.دراین بخش ما ناحیه های فرش را به راس های یک گراف تشبیه کردیم و همسایگی دو یال را، به

```
    Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i)
                adj[i] = new LinkedList();
        }
```

وجود یک یال بین این ناحیه تعبیر کردیم.اینگونه میتوان این مسیله 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.

برای پیدا کردن تعداد رنگ از متد coloring استفاده می کنیم که ابتدا یک ارایه با مقدار اولیه -۱ ایجاد می کنیم سپس در حلقه ای که تمام راس ها را طی می کنند چک می کنیم که ایا رنگی به ان اختصاص داده شده یا نه و سعی می کنیم رنگی که همسایه های ان ندارند را به ان اختصاص دهیم

```
void coloring() {
            int result[] = new int[V];
            Arrays.fill(result, -1);
            result[0] = 0;
            boolean available[] = new boolean[V];
            Arrays.fill(available, true);
            for (int u = 1; u < V; u++) {
                Iterator<Integer> it = adj[u].iterator();
                while (it.hasNext()) {
                    int i = it.next();
                    if (result[i] != -1)
                        available[result[i]] = false;
                }
                int cr;
                for (cr = 0; cr < V; cr++) {
                    if (available[cr])
                        break;
                }
```

```
                result[u] = cr; // Assign the found color

                Arrays.fill(available, true);
            }

            int min = -1 ;

            for (int u = 0; u < V; u++){
                if (min < result[u])
                    min = result[u];
                System.out.println("Vertex " + u + " ---> Color "
                        + result[u]);
            }

            System.out.println(min);
        }
```

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 اتیم 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 اریه 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 کی هر مدام یکی کمتر از ایتم 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 ایتم 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 شباهات 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 انها را به یک ارایه یک بعدی تبدیل می کنیم

بخش سوم

شرح کلی

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.

گزارش کار الگوریتم

بخش منطقی الگوریتم در کلاسBuyCarpet 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 تابه یک اری 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 قیمت
اولین شی که در اری لیست است ذخیره میشود.سپس داخل حلقه فور همه اشیا چک میشوند و اگر قیمت شی ای کمتر بود آن شی در answer ذخیره می شود.

private Long min(ArrayList carpets) { Long answer = Long.valueOf(carpets.get(0).price); for (int i = 0; i < carpets.size(); i++) { if (carpets.get(i).price < answer) answer = Long.valueOf(carpets.get(i).price); } return answer; }

تابع **getCarpet**

در این اری لیستی از موجودیت carpet و یک قیمت میگیریم.به عنوان خروجی carpet که ان قیمت را دارد برمیگردانیم.این کار را با یکحلقه فور ساده روی اری لیست انجام میدهیم. اگر همچین شی ای پیدا نشد مقدار null برگردانده میشود.

```
public Carpet getCarpet(Long price, ArrayList<Carpet> carpetsArr){
    for (int i = 0; i <carpetsArr.size() ; i++) {
     if (carpetsArr.get(i).price == price){
       return carpetsArr.get(i);

     }
    }
    return null;
  }
```

تابع **showLargestNumOfCarpet**

این تابع عملیات اصلی محاسبه را انجام میدهد. به عنوان ورودی لیستی از همه فرشهارا دریافت میکند. به طور خلاصه با استفاده از فرش های موجودی که در لیست آرایه ای carpets ذخیره شده وهمچنین پول دریافتی از کاربر گرفته میشود لیستی از فرش هایی که مجموع قیمتشان کمتر مساوی پول کاربر است را ارایه 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.

```
public ArrayList<String> showLargestNumOfCarpet(long money,ArrayList<Carpet> carpetsAr
    long totalPrice = 0;
```

```
    ArrayList<String> chosenCarpets = new ArrayList<>();
    while (totalPrice < money) {
      if (carpetsArr.size() == 0)
        break;
      Long minPrice = min(carpetsArr);
      if (minPrice + totalPrice <= money) {
        totalPrice += minPrice;
        Carpet carpetWithMinPrice = getCarpet(minPrice,carpetsArr);
        chosenCarpets.add(carpetWithMinPrice.name);
        carpetsArr.remove(carpetWithMinPrice);

      } else
        break;
    }
    return chosenCarpets;
  }
```

نمونه ورودی و خروجی

input:30000

output:[machine-carpet1, machine-carpet2]

تحلیل زمانی پروژه

پیچیدگی اصلی داخل حلقه وایل در تابع showLargestNumOfCarpet اتفاق میفتد.این حلقه در بدترین حالت به اندازه تعداد فرشها تکرار میشود و هر بار در دستور های وLong minPrice = min(carpetsArr) و Carpet carpetWithMinPrice = getCarpet(minPrice,carpetsArr) یک حلقه به تعداد فرشها پیمایش میشود اگر تعداد فرشها = n در نتیجه پیچیدگی این الگوریتم در بدترین حالت = O(n)

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 ارایه 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://globin.bx.psu.edu/courses/fall2001/DP.pdf