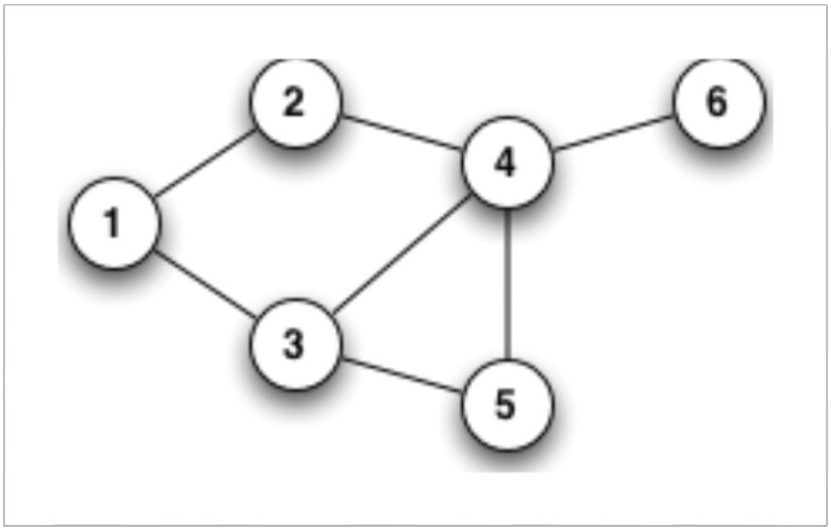
پروژه طراحی الگوریتم

استاد طباطبایی



Kruskal’s algorithm

TO FIND MINIMUM SPANNING TREE

امین ملک لو

961113077

در این الگوریتم، هدف پیدا کردن درخت پوشای مینیمم است. یعنی از گراف داده شده به عنوان ورودی، درختی پیدا کنیم که هم پوشا باشد و همه ی راس ها را شامل شود، و همچنین کمترین وزن ممکن را داشته باشد.

برای شروع ابتدا باید ورودی هارا از کاربر بگیریم و آنهارا به ترتیب غیرنزولی مرتب کنیم. همچنین روش اجرای برنامه و توابع توضیح داده خواهد شد.

ابتدا به بررسی کلی برنامه میپردازیم :

1 – ساختار کلی

برنامه به طور کلی از دو کلاس Matris و Kruskal تشکیل شده است

کلاس کروسکال کلاس اصلی برنامه است که main در آن قرار دارد و در آن یک شی از کلاس ماتریس ساخته میشود، کلاس ماتریس عملیات حفظ و حساب داده ها را انجام میدهد :

```
1
2   import java.util.Scanner;
3   import java.util.Vector;
4
5   > class Matris {···
219
220 > public class kruskal {···
```

2 – کلاس ماتریس

در ابتدا متغیر های اصلی این کلاس را میبینیم ، سپس رفتار توابع درون کلاس را بررسی میکنیم.

همانطور که در تصویر مشاهده میشود، اولین متغیر تعداد راس های گراف اولیه است که بسیار پر اهمیت است و در اکثر توابع از آن استفاده میشود.

سپس سه Vector برای گرفتن راس شروع و پایان و وزن یال ها به ترتیب با نام های Edges , dest , src طراحی کرده ایم.

Vector نوعی ساختمان داده مانند ArrayList است که قابلیت باز و منبسط شدن دارد، در آخر برای کمتر کردن حجم مورد استفاده ی برنامه، این سه را در یک آرایه

به اسم SDW (src , dest , weight) ذخیره میکنیم و آن را بر اساس وزن مرتب میکنیم.

```
5   class Matris {
6       private int numberOfVertices ;
7       Vector<Integer> Edges = new Vector<Integer>();
8       Vector<Integer> src = new Vector<Integer>();
9       Vector<Integer> dest = new Vector<Integer>();
10
11      //to import Sorce, Destination and Weigh vectors into one array
12      private int [][] SDW ;
13
14      // this array represents parent and children relations
15      // rows represent parent vertices ( except for [0])
16      // coloumns represent children vertices ( except for [0])
17      private int [][] parChild ;
18
19      // size array is meant to hold the size of each tree that each parent is holding
20      private int [] size ;
21
22
23      // to save SDW elements that do not create cycle for the final MST
24      private int [][] Final;
25
```

دیگر آرایه های مهم این برنامه آرایه ی parChild است که در آن نسبت فرزند و والد بودن بین راس ها مشخص میشود که برای تشخیص دور کاربرد دارد . و همچنین آرایه ی size که حجم هرکدام از زیردرخت ها را در خود ذخیره میکند و برای تشخیص دور کاربرد دارد.

آرایه ی Final که در تصویر قبل مشخص است درواقع درخت نهایی را در خود ذخیره میکند.

2 – (2) : توابع اصلی کلاس ماتریس

در این قسمت به توضیح توابع اصلی بر اساس ترتیب فراخوانی شدن آنها در main میپردازیم

1 – setNOV ( ) ;

به این علت که متغیر تعداد راس ها به صورت private تعریف شده، یک setter برای آن تعریف میکنیم :

```
30      public void setNOV () {
31
32          System.out.println("tedade ras haye graph ra vared konid:");
33          System.out.println(" deghat konid ke bishtar az 2 ras bashand");
34
35          this.numberOfVertices = scanner.nextInt();
36
37          for (; numberOfVertices <3;){
38              System.out.println("lotfan deghat konid tedade ras ha bishtar az 2 bashad");
39              this.numberOfVertices = scanner.nextInt();
40          }
41      }
```

2 – explain ( ) ;

این تابع درواقع توضیحات نحوه ی کار برنامه را به کاربر میدهد، یعنی نحوه ی وارد کردن راس ها و یال ها، و اتمام یال دهی به برنامه با وارد کردن عدد 99 به جای ورودی راس. اگر کاربر شرایط را مطالعه کند و ok وارد کند، خروجی تابع برابر true میشود و برنامه شروع به کار میکند.

```
95      public boolean explain () {
96          boolean stuation =false ;
97          String command ;
98          for (;!stuation;){
99          System.out.println("lotfan be tartibe 'Src' Enter 'Dest' Enter 'Weigh' vared konid: ");
100         System.out.println("deghat konid shomare ras ra dorost vared konid");
101         System.out.println(" 1 <= shomare ras <= tedade ras ha");
102         System.out.println("vazn (weight) nemitavanad manfi bashad");
103         System.out.println("baraye etmame meghdar dehi be jaye Src 99 ra vared konid");
104         System.out.println("dar soorate khandane raveshe kar 'ok' ra vared konid");
105         command = scanner.next();
106         if ( command.compareTo("ok") == 0)
107         stuation = true ;
108     }
109         return stuation;
110     }
```

3 – matrisGo ( ) ;

این تابع وظیفه ی یال گیری از کاربر را دارد تا زمانی که وروردی راس مقدار 99 داده شود.

```
44      public void matrisGo () {
45          boolean go = true ;
46          for (;go;){
47              if ( go ){
48                  for (Vi = 0 ; Vi < 1 || Vi > numberOfVertices ;)
49                  {
50                      System.out.print( "Src:\t");
51                      Vi = scanner.nextInt();
52                      if (Vi == 99) {
53                          go = false;
54                          break;
55                      }
56                  }
57              }
58              if (go){
59                  for (Vf = 0 ; Vf < 1 || Vi > numberOfVertices ;)
60                  {
61                      System.out.print( "Dest:\t");
62                      Vf = scanner.nextInt();
63                  }
64              }
65              if(go){
66                  for (weight = -1 ; weight < 0 ;)
67                  {
68                      System.out.print("Weight:\t");
69                      weight = scanner.nextInt();
70                      Edges.add(weight);
71                      src.add(Vi) ;
72                      dest.add(Vf) ;
73                  }
```

و همچنین در آخر این تابع، مقادیر سه Vector به آرایه ی دو بعدی SDW پاس داده میشوند و سه وکتور را null میکنیم.

4 – sortSDW ( ) ;

در این تابع همانطور که از نامش مشخص است، آرایه ی SDW را بر اساس وزن، از وزن کمتر به وزن بیشتر مرتب میکند.

// SDW [i] [0] = Sources, SDW [i] [1] = Destinations

// SDW [i] [2] = Weight pf Edges

```
114     public void sortSDW () {
115         int [] temp;
116         temp = new int [3] ;
117         for ( int i = 0 ; i < SDW.length ; i ++) {
118             for ( int j = i + 1  ; j < SDW.length ; j++ ){
119                 if (SDW[i][2] > SDW[j][2])
120                 {
121                     // to sort weight
122                     temp [2] = SDW[i][2];
123                     SDW[i][2] = SDW[j][2];
124                     SDW[j][2] = temp [2];
125 
126                     // to sync Sources with the Weight
127                     temp [0] = SDW[i][0];
128                     SDW[i][0] = SDW[j][0];
129                     SDW[j][0] = temp [0];
130 
131                     // to sync Destinations with Weight
132                     temp [1] = SDW[i][1];
133                     SDW[i][1] = SDW[j][1];
134                     SDW[j][1] = temp [1];
135                 }
136             }
137         }
```

روش پیاده سازی الگوریتم

پس از اینکه آرایه ی اصلی مرتب شد، دو مفهوم اصلی برای پیاده سازی الگوریتم تعریف میکنیم:

پرنت – سایز (والد – حجم)

در ابتدا هر راس پرنت خودش است و سایز همه ی راس ها برابر یک است .

یک یال (به همان ترتیب غیرنزولی) انتخاب میشود و راس شروع و پایان آن src و dest نام میگیرند. سپس بررسی میشود که سایز سرگروه کدام یک از دو راس بزرگتر است (سایز پرنت)، سایز هرکدام که بزرگتر بود به عنوان پرنت زیردرخت کوچک محسوب میشود، و زیردرخت کوچکتر را در زیردرخت بزرگتر قرار میدهیم، و پرنت زیردرخت بزرگتر، پرنت همه ی راس های زیر درخت کوچکتر میشود، همچنین سایز زیردرخت بزرگتر به علاوه ی سایز زیردرخت کوچکتر میشود و سایز زیردرخت کوچکتر برابر صفر میشود.

این عملیات در تابع merge اتفاق میفتد که در صفحه بعد بع بررسی این تابع خواهیم پرداخت.

5 – void merge (int p , int c) ;

// p is the parent index and c is the child index

```
157     public void merge ( int p , int c ) {
158
159         for ( int i = 1 ; i < numberOfVertices + 1 ; i ++ ) {
160             parChild[p][i] += parChild[c][i] ;
161             parChild [c][i] = 0 ;
162         }
163         size [p] += size [c] ;
164         size [c] = 0 ;
165
166     }
```

6 – int par (int childIndex) ;

این تابع اندیس یک راس را میگیرد و اندیس راس پرنت آن را برمیگرداند.

```
142     public int par ( int childIndex ) {
143
144         int c = childIndex ;
145
146         for ( int i = 1 ; i < numberOfVertices + 1 ; i ++ ) {
147             if ( parChild[i][c] == 1 ){
148                 return i ;
149             }
150         }
151         return 0 ;
152
153     }
```

7 – void krus ( ) ; => variables

این تابع، تابع اصلی برنامه است که انجام الگوریتم، با استفاده از توابع قبلی در آن انجام میگیرد.

در ابتدا سه آرایه ی parChild ، size و Final را در آن میسازیم، سپس چهار متغیر زیر را برای راحتی در نوشتن و محاسبه تعریف میکنیم:

Src: Source Vertex

parS: parent pf src (parS = par (Src) ;)

dest: Destination Vertex

parD: parent of dest

```
168     public void krus () {
169
170         parChild = new int [numberOfVertices+1][numberOfVertices+1];
171         size = new int [numberOfVertices+1] ;
172         Final = new int [numberOfVertices-1][3] ;
173
174         for ( int i = 1 ; i < size.length ; i ++ ){
175             parChild[i][i] = 1 ;
176             size[i] = 1 ;
177         }
178
179         // parent of src, parent of dest
180         int src , parS ;
181         int dest , parD ;
```

8 – void krus ( ) ;

طبق توضیحاتی که در سه صفحه قبل (صفحه هفت) داده شد الگوریتم را مینویسیم، اگر پرنت هر دو راس یعنی src و dest برابر بود یعنی دور ایجاد میشود، تایع این یال را تشخیص میدهد و آن را در Final قرار نمیدهد و مشخصات یال را چاپ میکند. پیچیدگی زمانی انجام الگوریتم با داشتن متغیر ها و پرنت ها برابر n است.
همچنین برای محاسبه ای اندیس پرنت n و merge کردن زیردرخت ها نیز o (n) است، ولی merge و par به صورت موازی انجام میشوند پس در کل میتوان گفت پیچیدگی زمانی این کد برابر o (n^2) میباشد.

```
183     for ( int i = 0 , j = 0 ; i < SDW.length && j < (numberOfVertices - 1) ; i++ )
184     {
185         src = SDW[i][0] ;
186         dest = SDW[i][1] ;
187         parS = par(src) ;
188         parD = par(dest) ;
189         if ( parS == parD ) {
190             //here
191             System.out.println("yal e " + src + " ==> " + dest + " ( Weight = " + SDW[i][2] + " )"+" dor
192         }
193         else {
194             if ( size [parS] >= size [parD] ) {
195                 merge(parS, parD);
196             }
197             if ( size [parD] > size [parS] ) {
198                 merge(parD, parS);
199             }
200             Final [j][0] = src ;
201             Final [j][1] = dest ;
202             Final [j][2] = SDW [i][2] ;
203             j ++ ;
204         }
205     }
206 
207 }
```

9 – void show ( ) ;

این تابع فقط جواب آخر (یعنی یال ها و مشخصات آنها) را با نمایش ترتیب رسم نشان میدهد:

```
209     public void show () {
210         System.out.println("\nminimum spanning tree looks something like this:\n");
211         System.out.println("================================");
212         System.out.println("index\tSrc\tDest\tWeight");
213         for ( int i = 0 ; i < Final.length ; i ++ ){
214             System.out.println( (i+1) + "\t" +Final[i][0] + "\t" + Final[i][1] + "\t" + Final[i][2]);
215         }
216         System.out.println("================================");
217     }
218 
219 }
```

10 – main ( ) ;

ساخت شی از کلاس ماتریس و فزاخوانی ها:

```
221   public class kruskal {
222 
          Run | Debug
223       public static void main(String[] args) {
224           Matris matrix = new Matris ();
225           matrix.setNOV();
226           boolean situation ;
227           Scanner scnr = new Scanner (System.in);
228           situation = matrix.explain();
229           if (situation)
230           {
231               matrix.matrisGo() ;
232           }
233           scnr.close();
234           matrix.krus();
235           matrix.show();
236 
237       }
238   }
```

نمونه ورودی و خروجی

برای اطمینان از کارکرد برنامه، گراف نه راسی زیر را طبق قوانین ورودی به برنامه میدهیم:

example for input graph :

Src	Dest	Weight
1	2	4
2	8	11
1	8	8
2	3	8
6	5	10
7	6	2
3	4	7
3	6	4
4	6	14
8	7	1
4	5	9
8	9	7
9	3	2
9	7	6

خروجی برنامه

در نهایت خروجی برنامه به شکل زیر خواهد بود:

```
Src:    99
yal e 9 ==> 7 ( Weight = 6 ) dor ijad mikonad
yal e 8 ==> 9 ( Weight = 7 ) dor ijad mikonad
yal e 1 ==> 8 ( Weight = 8 ) dor ijad mikonad

minimum spanning tree looks something like this:

==================================
index   Src     Dest    Weight
1       8       7       1
2       7       6       2
3       9       3       2
4       3       6       4
5       1       2       4
6       3       4       7
7       2       3       8
8       4       5       9
==================================
```

پایان