北陸先端科学技術大学院大学

情報科学研究科　上原隆平准教授

学生番号:0750011

氏名:奥村　浩徳

○副テーマ－区間グラフの平均頂点間距離を求めるプログラミング－

区間グラフとは、数直線に対応する一定区間の中で、ある「幅」を持つものを頂点と見るグラフの事です。そして2つの「幅」が共有する区間を持てばその2つの頂点は繋がっているとみなされます。2頂点の間は自由に移動できるので、グラフに向きはありません。

例として0から10までの整数区間があるとします。頂点Aは3から6までの幅であり、頂点Bは5から8までの幅であるとします。そうすると頂点AとBは5から6までの共有区間を持つので繋がっていることになります。

以下要素数を1000、スタート区間を0～100まで、長さを1～５１までに設定して、ランダムに頂点を発生させたときの区間グラフの平均長を求めるJavaでのプログラムです。

```
class intervalgraph {

  private int start;
  private int end;

  public intervalgraph(int st, int len)
  {
     start = st;
     end = len+st;
   }
  public void display(int j)
  {
    System.out.println("start: "+start+" ,end: "+end+" ,number: "+j);
  }
  public int getstart()
  {
    return start;
  }
  public int getend()
  {
    return end;
  }
```

}//このクラスではまず発生させた整数を用いて、スタート地点とエンド地点を持つ頂点を作り、それらに番号付けをしています。頂点のエンド地点はスタート地点の値と頂点の長さによって設定しています。

```
class Vertexlist {

      private intervalgraph[]a;

      private int nElems;

      private int[][]ElementMatrix;

      int farVertex = 0;

      int maxend;

      int max;

    public Vertexlist(int maxSize)
    {
        max = maxSize;
        a = new intervalgraph[max];
        nElems = 0;
        ElementMatrix = new int [max][max+1];
    }
    public void insert(int st,int len)
    {
        a[nElems]=new intervalgraph(st,len);
        nElems++;
    }
    public void displayVertex()
    {
        for(int j=0;j<nElems;j++)
            a[j].display(j);
     }
    public void insertsort1()
    {
        int in,out;
        for(out=1;out<nElems;out++)
        {
            intervalgraph temp = a[out];
            in = out;

            while(in>0 && (temp.getstart()<a[in-1].getstart()))
            {
                a[in] = a[in-1];
                --in;
            }
            a[in] = temp;
        }
    }
    public void insertsort2(int startrange)
    {   int in,out;
        for(out=1;out<nElems;out++)
            {
                        temp.getstart()==k && a[in-1].getstart() == k)
```

```
            intervalgraph temp = a[out];
                  in = out;
                        for(int k = 0; k<startrange ; k++)
                        {
                        while(in>0 && (temp.getend()<a[in-1].getend()) &&
                                    temp.getstart()==k && a[in-1].getstart() == k)
                        {
                              a[in] = a[in-1];
                              --in;
                        }
                        a[in] = temp;
                        }
               }
    }
```

//ここまでのプログラムの解説をします。コンストラクタの中では作った頂点を格納する、頂点数（いま仮にｎとします）の要素分の配列を作っています。さらに各頂点間の最短距離を求めるためにｎ×（ｎ＋１）の行列（ElementMatrix)を作っています。最初に作った頂点をinsert()で配列に次々に入れinsertsort1()で挿入ソートを使ってスタート地点の順に並び替えます。次にinsertsort2()でスタート地点を固定した状態で、長さの短いものから順に頂点をソートします。

```
public void adj()
    {
        for(int i=0;i<max;i++)
        { maxend = 0;
         for(int j=0;j<max;j++)
            { if(i==j)
                { ElementMatrix[i][j]=0; }
           else if(a[j].getstart()<a[i].getend() && a[j].getstart()>=a[i].getstart())
                 { ElementMatrix[i][j]=1;
                  if( a[j].getend()>maxend)
                     { maxend = a[j].getend();
                     ElementMatrix[i][max] = j; }
                  }
              }
         }
    }
    public int checkMatrix()
    {
        int check = 0;
        int i=0;
        while(i<max-1)
        {
            if(ElementMatrix[i][max]==i)
               {  check = 1;
                  break;
               }
            else
               { i++; }
        }
        return check;
    }
```

```
 public double getTotalSP()
{
   int totalsum =0;
   int sum;
       for(int i=max-2;i>=0;i--)
       {   sum =0;
             for(int j=i+1;j<max;j++)
             {
             if(ElementMatrix[i][j]==0)
            {ElementMatrix[i][j]=ElementMatrix[ElementMatrix[i][max]][j]+1;}
             sum+=ElementMatrix[i][j];
             }
        }
      totalsum+=sum;
   }
   return totalsum;
}
```

}//クラスVertexlistはここまでで終わりです。メソッドadj()で頂点i($1 \le i \le n$)を基準にして、相手の頂点j($i \le j \le n$)が自分と接しているかどうかを判定して、接していれば1、接していなければ0をElementMatrixの(i,j)成分に代入していきます。接しているかどうかは相手の頂点のスタート地点が自身の幅に含まれているかどうかで判定します。(i,n+1)には自分と接している頂点で一番遠くまで伸びているものの番号を代入します。そしてcheckMatrix()でグラフが繋がっているかどうか調べます。ElementMatrixの(i,n+1)($1 \le i \le n-1$)を見て、番号iが代入されていればその頂点は自分自身としか接していないのでグラフはそこで切れていることになります(ただし1番最後の頂点は除く)。その後にElementMatrixのn行目から1行目まで順に各頂点jへの最短距離を求めていきます。具体的には列jに1の値が入っていたらそれが最短距離となり、0が入っていたらn+1列目の値の頂点の行にジャンプして、その行のj列目の値に1を足したものが2頂点間の最短距離となります(ジャンプした先の頂点から頂点jまでの最短距離はすでに求まっている状態です)。最後にElementMatrixのn+1列目を除いた上三角行列の要素数の和を求めます。

```
class intervalgraphApp {

        public static void main (String args[])
        {
          int maxSize = 1000; //任意の要素数を代入してください
          int startrange = 100; //スタートの区間を代入してください
          int lengthrange = 50; //頂点の長さの幅を代入してください
          double sum = 0;
          double denominator = maxSize*maxSize;

        Vertexlist arr = new Vertexlist(maxSize);
        for(int j=0;j<maxSize;j++)
        {
            int x = (int)(Math.random()*startrange);
            int y = (int)(Math.random()*lengthrange+1);
            arr.insert(x,y);
        }

        arr.insertsort1();
        arr.insertsort2(startrange);
        arr.displayVertex();
```

```
        arr.adj();
        System.out.println("Vertex list: ");
        arr.displayVertex();
       if(arr.checkMatrix()==1)
                    { System.out.print("This graph is disconnected."); }
             else
             {
                   sum=arr.getTotalSP();
                System.out.println("Average shortest path: "+sum*2/denominator);
             }
       }
}
```

//ここでプログラムは終わりです。maxSize,startlange,lengthlange にはそれぞれ、要素数、スタート地点の区間、頂点が取りうる長さを入力してください。なお頂点の長さは最低でも1とするために、強制的に1を足してあります。あとは for()で要素の個数分だけ頂点を作って、それらに対して最短距離を求めるプログラムを実行します。そして ElementMatrix の(n+1 列目を除いた)上三角行列の和を2倍して、頂点の組み合わせの総数 n×n で割れば平均長が得られます。

☆プログラムの基本的なアルゴリズム

最初に述べたように、区間グラフに向きは存在しません。よってある頂点Aから頂点Bまでの最短距離は、頂点Bから頂点Aへの最短距離と等しいです。それを利用してプログラムの効率を上げるために、一方の頂点からもう一方の頂点までの最短距離の計算しか行っていません（後でそれらを2倍すればいいので）。

頂点Aから頂点Bへの最短経路の求め方は簡単で、まず頂点Aに接している頂点のうち頂点Bの方向に「最も遠くまで伸びている頂点」を選択します。さらにその頂点に接している頂点のうち「最も遠くまで伸びている頂点」を選択する、という動作を繰り返していけば頂点Bまで最も短い距離でたどり着くことができます。これは区間グラフに順序があるために可能な考え方です。よって必要な情報はある頂点に接している頂点のナンバーと、それらのうちで（正の方向あるいは負の方向に）最も遠くまで伸びている頂点のナンバーだけでいいことが分かります。このプログラムでは最初にスタート地点の小さいほうから順にソートをして、正の方向を自然な向きとして考えました。

ある頂点Cがべつの頂点Dに接しているかどうかの判定は、頂点Dの幅の中に頂点Cのスタート地点があるかどうかをみて行います。エンド地点での判断を行わなくていいのは、もし頂点Dの幅の中に頂点Cのエンド地点が含まれていれば、頂点Cの幅の中には頂点Dのスタート地点が必ず含まれているからです。

最初に考えた最短経路の値の求め方は、可変長リストを用意して目的の頂点に到達するまでに次々と「最も遠くまで伸びている頂点」のナンバーを入れていき、ある一つの最も長い頂点が目的の頂点と接していたら、リストの要素数に1を足して得るというものです。それら全てを足して2倍して、頂点全ての組み合わせの数で割ると区間グラフの平均最短距離が得られます。しかしこのやり方では重複して計算を行ってしまう場合が多々あるし、可変長リストの分だけ余分なメモリを食うという欠点がありました。

そこでもっと効率の良いアルゴリズムを考えて、頂点のソートをスタート地点で行った後にさらにエンド地点で行って、最短距離を一番最後の頂点から求めるというやり方にしました。そうすれば、ある頂点が目的の頂点までの最短距離を求めるときにジャンプする「最も遠くまで伸びている頂点」から目的の頂点までの計算はすでに行われている状態となり、それを利用する事ができます。このような効率化は区間グラフに順序が入っているから可能になります。

上のプログラムの出力結果は次の通りです。

1回目:1.694092	6回目:1.696848
2回目:1.69824	7回目:1.671848
3回目:1.686822	8回目:1.669742
4回目:1.685798	９回目:1.682198
5回目:1.694994	10回目:1.668502

これより上の設定では2頂点間の距離は 1.66~1.69 である事が分かります。つまり大体1ステップ2ステップ、多くても3ステップで2頂点が繋がっている状況となっています。これは ElementMatrix の n+1 列目を表示させてみれば分かりますが、幅の長い頂点がいくつか存在してそれらが任意の2頂点間の距離を縮めている結果であると言えます。