به نام خدا

|  |  |  |
| --- | --- | --- |
| دانشگاه ايلام | ترم اول سال تحصيلي 98-97 | سري نهم تمرينات ساختمان داده ها |

1. متدی بنویسید که کوچکترین عنصر یک توده‌ی حداقل را حذف کند. فرض کنید که از نمایش آرایه‌ای استفاده می‌کنیم. پیچیدگی الگوریتم شما چقدر است؟
2. توده ی حداقل را با آرایه برای اعداد طبیعی پیاده کنید. سپس اعمال زیر را به ترتیب از چپ به راست انجام دهید.

add(4), add(18), add(11), add(2), add(31), add(42), delete(), delete(), add(13), add(5), add(1), delete()

1. یک کلاس جاوا به اسم Graph برای نمایش گرافهای بدون جهت پیاده کنید. متدهای زیر را پیاده کنید. جزء سازنده کلاس به صورت زیر است:

**public Graph(int n)**

این متد یک گراف با گرههای 0 تا n-1 می‌سازد که هیچ یالی ندارد.

کلاس همچنین دارای متدهای زیر است:

**public void addEdge(int i, int j)**

این متد یک یال بین نودهای i و j ایجاد می کند به شرطی که 0≤i<n و 0≤j<n و i≠j و قبلاً هیچ یالی بین i و j وجود نداشته باشد.

**public boolean isConnected()**

این متد چک می کند که آیا گراف یکپارچه است یا نه؟

**public void clear()**

این متد تمام یالهای گراف را پاک می‌کند.

**public String toString()**

این متد لیست مجاورت گراف را بر می گرداند. مثلاً، اگر کد زیر را اجرا کنیم:

**Graph g = new Graph(4);**

**g.addEdge(0,1); g.addEdge(0,2); g.addEdge(0,3);**

**g.addEdge(1,3); g.addEdge(2,3);**

**System.out.println(g);**

نتیجه ی زیر چاپ خواهد شد:

**From To**

**0: 1,2,3**

**1: 0,3**

**2: 0,3**

**3: 0,1,2**

**public void randomize(double p)**

این متد تمام یالهای گراف را حذف می‌کند. سپس، برای هر زوج (i,j) که 0 ≤ i < j ≤ n-1  باشد، یک عد تصادفی r بین ۰ و ۱ تولید می کنیم. اگر r ≤ p باشد، یک یال بین یالهای i و j اضافه می‌شود.

**public static void runTests(int size, double p0, double p1, int steps, int k)**

این متد الگوریتم زیر را اجرا می‌کند:

Let *d = (p1-p0)/steps* and g = new Graph(size).  
Then: for *p=p0,p0+d,p0+2d,…,p0+(steps-1)d,p1*,   
    count = 0;   
    repeat *k* many times   
        g.randomize();   
        if (g.isConnected()) count++;   
    print *p* and *(count / k)*

لذا، خروجی فراخوانی runTests(20,0.0,0.5,10,60); چیزی شبیه زیر خواهد بود:

**p = 0.000 fraction connected = 0.000**

**p = 0.050 fraction connected = 0.000**

**p = 0.100 fraction connected = 0.050**

**p = 0.150 fraction connected = 0.350**

**p = 0.200 fraction connected = 0.683**

**p = 0.250 fraction connected = 0.917**

**p = 0.300 fraction connected = 0.950**

**p = 0.350 fraction connected = 0.983**

**p = 0.400 fraction connected = 1.000**

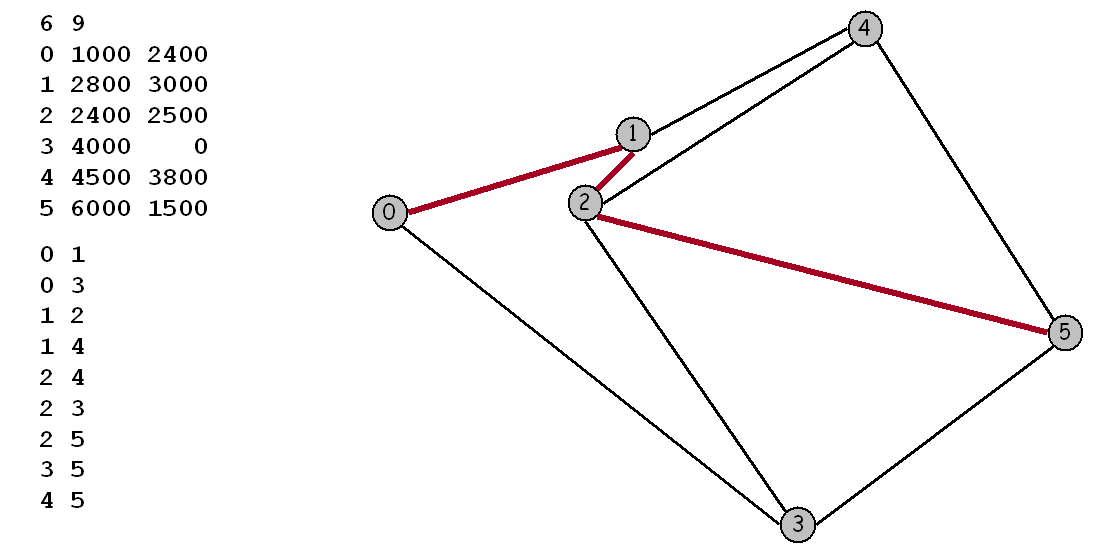
**p = 0.450 fraction connected = 1.000**

**p = 0.500 fraction connected = 1.000**

**public void DFS(double p)**

این متد پیمایش اول عمق را تولید می‌کند.

1. در این تمرین الگوریتم کوتاهترین مسیر دکسترا پیاده می گردد. نقشه شهرها در یک فایل داده می‌شود. این فایل شامل دو عدد است که تعداد نودها و یالها را مشخص می‌کند. سپس موقعیت هر گره داده شده است. در نهایت، یالها مشخص شده‌اند. به مثال زیر توجه کنید:



برنامه‌ی شما باید ابتدا اسم فایل را از کاربر بگیرد. سپس، محتویات فایل را در یک گراف بریزد. سپس، شماره‌ی نود منبع و مقصد را از کاربر بگیرید و کوتاهترین مسیر بین آنها را پیدا کنید. مثلاً، خطوط قرمز کوتاهترین مسیر بین نودهای ۰ و ۵ را نشان می‌دهند.