Parallel Processing

#4 Homework Solutions

محمد حسین خوشه چین - ۹۹۲۱۰۱۶۴

۲۶ اردیبهشت ۱۴۰۰

توضیحات

الگوریتم نوشته شده بر اساس الگوریتم مقاله[۱] می باشد. سیستم عامل سیستم بنده Pop OS می باشد که یک توزیع از سیستم عامل Ubuntu محسوب می شود. میزان حافظه سیستم 8 Gig و پردازنده از نوع Intel Core i5 می باشد. زبان پیاده سازی این الگوریتم Java و زبان اسکریپ نوشته شده bash می باشد. اندازه گیری زمان اجرای برنامه ها توسط تایمر موبایل انجام شده است بنابراین ممکن است تا ۲ ثانیه خطا داشته باشد.

مسئله نگاشت کاهش الگوریتم پوسته محدب

در ابتدا یک برنامه به نام MakeFile می نویسیم که یک میلیارد مختصات در بازه $(0, 1000)^2$ به شکل تصادفی ایجاد کند. سپس مختصات هر نقطه را در یک خط جدا به صورت csv در فایلی به نام out.txt ذخیره می کنیم. کد این برنامه در شکل ۱ آمده است. اجرای این برنامه به

```java
import java.lang.Math;
import java.io.IOException;
import java.io.FileWriter;
public class MakeFile{

    public static void main(String []args) throws IOException{
        FileWriter fw = new FileWriter("out.txt");
        for(int i = 0 ; i<1000000000 ; i++){
            int a = (int)(Math.random()*(1000-0+1)+0);
            int b = (int)(Math.random()*(1000-0+1)+0);
            fw.write(a+","+b+System.getProperty( "line.separator" ));
        }
        fw.close();
    }
}
```

شکل ۱

مدت ۳ دقیقه ۳۹ ثانیه طول کشید و فایل out.txt با حجم ۷.۸ Gig گیگ ساخته شد. حال طبق الگوریتم باید تمام نقطه ها ابتدا بر اساس محور x مرتب کنیم و در صورتی که x ها برابر بود بر اساس y ها مرتب کنیم. برای انجام این مرتب سازی از یک دستور لینوکسی استفاده می کنیم. این دستور در شکل ۲ آمده است. در دستور sort از چند سوییچ استفاده شده است. سوییچ g

```
kc@pop-os:~$ sort -g out.txt -n out1.txt
```

شکل ۲

بدین معناست که سطر های فایل را بر اساس مقادیر عددی و به صورت lexicography مرتب

کند. سوییچ o بدین معناست که خروجی این مرتب سازی در فایلی به نام out1.txt ذخیره شود. اجرای این دستور به مدت ۲ ساعت و ۵۸ دقیقه و ۵۳ ثانیه به طول انجامید.

حال می خواهیم این فایل را به فایل هایی بشکنیم که هر کدام حداکثر ۵ میلیون مختصات دارد به بیان دیگر این فایل را به ۲۰۰ فایل تقسیم کنیم که هر کدام ۵ میلیون مختصات را در خودشان دارند و بعد از آن الگوریتم مان را بر روی این ۲۰۰ فایل اجرا کنیم. با استفاده از یک دستور لینوکسی می توانیم این فایل را بشکنیم. این دستور در شکل ۳ آورده شده است. در

```
kc@pop-os:~$ split --verbose -l5000000 out1.txt out.
```

شکل ۳

دستور split از چند سوییچ استفاده شده است. سوییچ verbose بدین معناست که زمانی که هر فایلی به عنوان خروجی ساخته می شود نامش در خروجی چاپ شود. سوییچ l بدین معناست که هر فایل حداکثر ۵ میلیون سطر داشته باشد. بعد از این سوییچ ها نام فایل ورودی یعنی out1.txt را می نویسیم و سپس پیشوند فایل هایی که قرار است بسازیم را مشخص می کنیم که پیشوند فایل های ما out می باشد. اجرای این دستور به مدت ۵ دقیقه و ۴۸ ثانیه به طول انجامید. در انتها ۲۰۰ فایل ساخته شد. حالا باید الگوریتم مان را پیاده بکنیم و سپس بر روی تمام این ۲۰۰ فایل اجرا کنیم. پیاده سازی این الگوریتم در برنامه MapReduce انجام شده است که در شکل ۴ و ۵ آورده شده است.

```java
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.util.Arrays;
public class MapReduce{

    public static void main(String []args) throws FileNotFoundException, IOException{
        BufferedReader csvReader = new BufferedReader(new FileReader(args[0]));
        int n = Integer.parseInt(args[1]);
        Point  p[] = new Point[n];
        String row="";
        for(int i = 0 ; i< n ; i++){
            row = csvReader.readLine();
            String[] data = row.split(",");
            Point o = new Point();
            o.x = Integer.parseInt(data[0]);
            o.y = Integer.parseInt(data[1]);
            p[i] = o;
        }
        csvReader.close();
        Point temp[] = new Point[n*2];
        int l = 0;
        int u = 0;
        for(int i=0 ; i<n ; i++){
            while(l>=2 && determinan(temp[l-2],temp[l-1],p[i]) <= 0){
                l--;
            }
            temp[l] = p[i];
            l++;
        }

        u = l+1;
```

شکل ۴

```
36          for(int i=n-2 ; i>=0 ; i--){
37              while(l>=u && determinan(temp[l-2],temp[l-1],p[i]) <= 0){
38                  l--;
39              }
40              temp[l] = p[i];
41              l++;
42          }
43          if(l > 1){
44              temp = Arrays.copyOfRange(temp,0, l-1);
45          }
46          BufferedWriter writer = new BufferedWriter(new FileWriter("final.txt", true));
47          for(int i = 0 ; i<temp.length ; i++){
48              if(temp[i] != null){
49                  writer.write(System.getProperty( "line.separator" )+temp[i].x+","+temp[i].y);
50              }
51          }
52          writer.close();
53      }
54
55      public static int determinan(Point a, Point b, Point c){
56          int d = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
57          return d;
58      }
59  }
```

شکل ۵

پیش از تحلیل برنامه نوشته شده باید اشاره کرد که در این الگوریتم لازم است که لیست نقاط در ابتدا مرتب شوند و چون ما با استفاده از دستور لینوکسی این کار را انجام داده ایم دیگر درون برنامه مان عمل مرتب سازی انجام نمی دهیم. از خط شماره ۱ تا ۲۳ برنامه مان فایل ورودی را با کمک args[0] میخوانیم و به ازای هر سطر آن که معرف یک مختصات است یک object از تایپ Point تعریف می کنیم و متغیرهای x و y اش را مقدار دهی می کنیم سپس این object ها را در یک آرایه به نام p که از تایپ Point است ذخیره می کنیم. طول این آرایه برابر با تعداد خطوط برنامه است که از متغیر args[1] خوانده می شود. کلاس Point تنها دو attribute به نام های x و y دارد. کد کلاس Point در شکل ۶ آورده شده است.

```
1 public class Point{
2
3     public int x;
4     public int y;
5 }
```

شکل ۶

الگوریتم ای که استفاده کرده ایم در دو مرحله پوسته محدب را محاسبه می کند و از مرتبه O(n) می باشد. در مرحله اول پوسته Upper را و در مرحله دوم پوسته lower را پیدا میکند. جزئیات این مراحل بدین صورت است که دو لیست که در ابتدا تهی هستند در نظر میگیرد. یکی برای محاسبه پوسته Upper و دیگری برای محاسبه پوسته lower. ما در کدمان به جای دو لیست از یک لیست استفاده می کنیم . در خط ۲۴ کد شکل ۴ یک آرایه به طول دو برابر تعداد مختصات مان تعریف می کنیم. الگوریتم در مرحله اول لیست نقاط را(p) از سمت چپ به سمت راست پیمایش می کند. در هر مرحله اندازه طول لیست upper را بررسی می کند . اگر کوچکتر از ۲ بود آن نقطه مشاهده شده را به لیست upper اضافه می کند. اما اگر طول لیست بیشتر از ۲ بود آنگاه درون یک حلقه می آید دو عنصر آخر لیست را با نقطه مشاهده شده بررسی می کند. دنبال این است که بفهمد اگر نقطه مشاهده شده را به انتهای لیست اضافه کند

آنگاه این سه نقطه با یکدیگر یک حرکت پادساعتگرد تشکیل میدهند یا خیر. به بیان دیگر اگر عنصر یکی مونده به آخر لیست را به عنصر آخر لیست وصل کنیم و سپس عنصر آخر لیست را به نقطه مشاهده شده وصل کنیم آیا یک فرو رفتگی به داخل تشکیل می دهند یا خیر و یا حتی هر ۳ به روی یک خط قرار میگیرند یا خیر . این کار را با استفاده از قضیه دترمینان می توان انجام داد. اگر دترمینان این سه نقطه را محاسبه کنیم و جواب منفی شود یعنی آن فرو رفتگی به داخل وجود دارد. اگر جواب دترمینان صفر شود یعنی هر سه نقطه روی یک خط قرار دارند. اگر جواب بزرگتر از صفر شود یعنی فرو رفتگی به سمت بیرون است و یا به بیان دیگر با وصل کردن این سه نقطه یک حرکت ساعتگرد شکل می گیرد و این حالت مجاز است و مشکلی ندارد. حال ما درون آن حلقه چک می کنیم. تا زمانی که طول لیست بزرگتر از ۲ است و حاصل دترمینان نقطه مشاهده شده و دو عنصر آخر لیست کوچکتر مساوی صفر است آنگاه درون حلقه بمان و در هر دور عنصر آخر لیست را حذف کن. زمانی که شرط حلقه نقض شد آنگاه نقطه مشاهده شده را به انتهای لیست اضافه کن. این مرحله در کدمان از خط ۲۵ تا ۳۳ انجام می شود. در مرحله دوم لیست نقاط را از سمت راست به چپ بررسی می کنیم و دقیقا همان اعمالی که در مرحله یک انجام دادیم را مجدد تکرار می کنیم. این مرحله در کدمان از خط ۳۴ تا ۴۲ انجام می شود. پس از پایان اجرای این مرحله حال باید نقاط تکراری پوسته محدب را حذف کنیم و عنصر آخر دو لیست را حذف کنیم و سپس این دو لیست را به یکدیگر بچسبانیم. چون ما از یک آرایه به جای دو لیست استفاده کرده ایم نیاز به چسباندن نداریم و حواسمون بود که لیست دوم را در آرایه از خانه ای شروع کنیم که برابر با خانه آخر لیست اول است. حال تنها باید خانه آخر آرایه را حذف کنیم و عناصر تکراری آن را هم دور بریزیم که این کار را در خط ۴۳ تا ۴۵ کد مان انجام داده ایم. در ادامه از خط ۴۶ تا ۵۳ خروجی پوسته محدب را درون فایل final.txt ذخیره می کنیم به طوریکه در هر بار اجرا خروجی در ادامه فایل ذخیره می شود و این یعنی دیگر نیاز نیست تا فایل های خروجی را بهم بچسبانیم.

حال باید یک شل اسکریپ بنویسیم که برنامه ما را به ازای تعداد فایل هایی که داریم اجرا کند و هر فایل را به عنوان ورود به آن بدهد. برای اینکار یک اسکریپت در فایل test.sh می نویسیم. کد این فایل در شکل ۷ قابل مشاهده است.

```
1 #!/bin/bash
2
3 for f in out*; do
4 java MapReduce "$f" "5000000"
5 done
```

شکل ۷

در این کد میگوییم که به ازای تمام فایل هایی که با out شروع می شوند برنامه را اجرا کن و به عنوان ورودی نام آن فایل و تعداد خطوط آن را بده. تنها باید حواسمان باشد که آن فایل out1.txt که همه مقادیر را داشت را حذف کنیم. حال برای اجرای این شل اسکریپت مطابق

شکل ۸ عمل می کنیم. اجرای این اسکریپت به مدت ۳۱ دقیقه و ۲۵ ثانیه به طول انجامید و

```
kc@pop-os:~/MapReduce/TM$ ./test.sh
```

شکل ۸

فایل final.txt در انتهای کار ۱۳۹۶ خط داشت. حال که همه داده ها را تجمیع کردیم کافیست که یک بار دیگر این برنامه را بر روی فایل تجمیع شده اجرا کنیم تا حاصل پوسته محدب نهایی بدست آید. حال برای اجرای نهایی ابتدا باید فایل final.txt را مانند شکل ۹ مرتب کنیم. اجرای این دستور تنها یک ثانیه به طول انجامید. حال با دستوری مانند شکل ۱۰ برنامه

```
kc@pop-os:~/MapReduce/FM$ sort -g final.txt -o end.txt
```

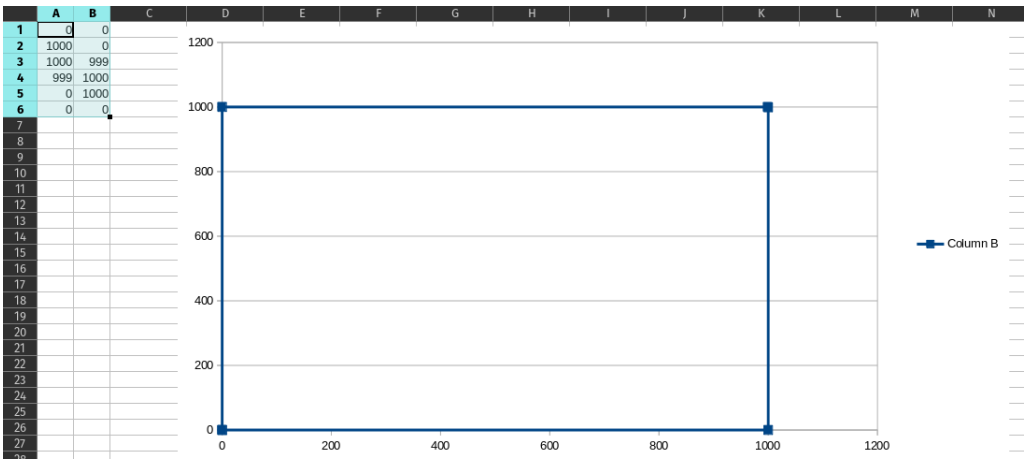
شکل ۹

MapReduce را بر روی فایل end.txt اجرا می کنیم تا پوسته محدب نهایی بدست بیاید و در فایل final.txt ذخیره شود منتها قبل از اجرا باید محتویات داخل فایل final.txt را خالی کنیم. اجرای این دستور نیز تنها یک ثانیه به طول انجامید. نمودار پوسته محدب این اجرا در شکل ۱۱

```
kc@pop-os:~/MapReduce/FM$ java MapReduce end.txt 1396
```

شکل ۱۰

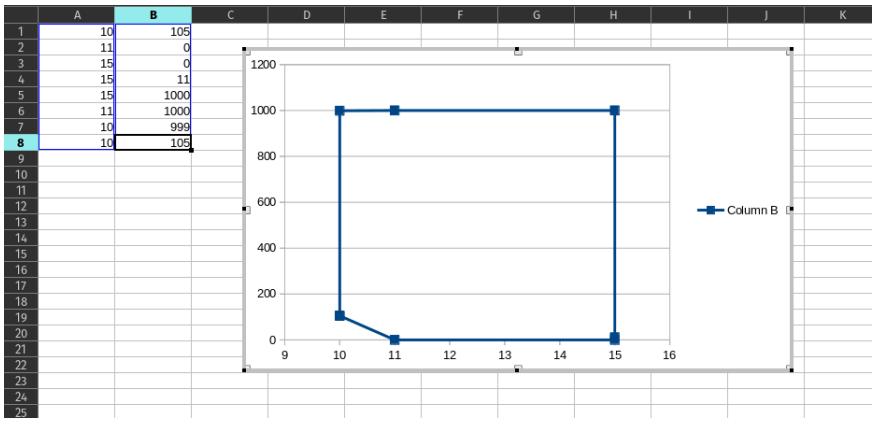
قابل مشاهده می باشد.



شکل ۱۱

حال می خواهیم زمان اجرا را بر اساس m=5000000 محاسبه کنیم. زمان اجرای ساخت فایل یک میلیارد خطی برای تمام ماشین ها یکی است پس از آن صرف نظر می کنیم. زمان

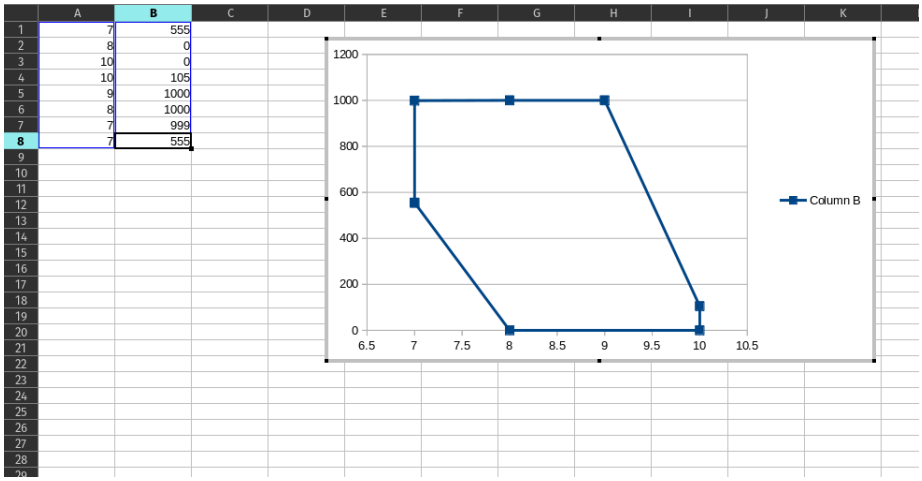
تقسیم کردن یک فایل به ۲۰۰ فایل برابر با ۵ دقیقه و ۴۸ ثانیه. زمان اجرای اسکریپت برابر با ۳۱ دقیقه و ۲۵ ثانیه اما چون میخواهیم آن را پارالل در نظر بگیریم به طور نمونه اجرای برنامه بر روی یکی از فایل ها را محاسبه می کنیم که برابر با ۴ ثانیه می باشد. در شکل ۱۲ می توانید پوسته محدب این اجرای نمونه را مشاهده کنید. زمان مرتب سازی فایل ادغام شده ۱ ثانیه و زمان اجرای الگوریتم روی آن ۱ ثانیه می باشد . بنابر این در مجموع ۵ دقیقه و ۵۴ ثانیه زمان برد تا بتوانیم مسئله را حل کنیم.



شکل ۱۲

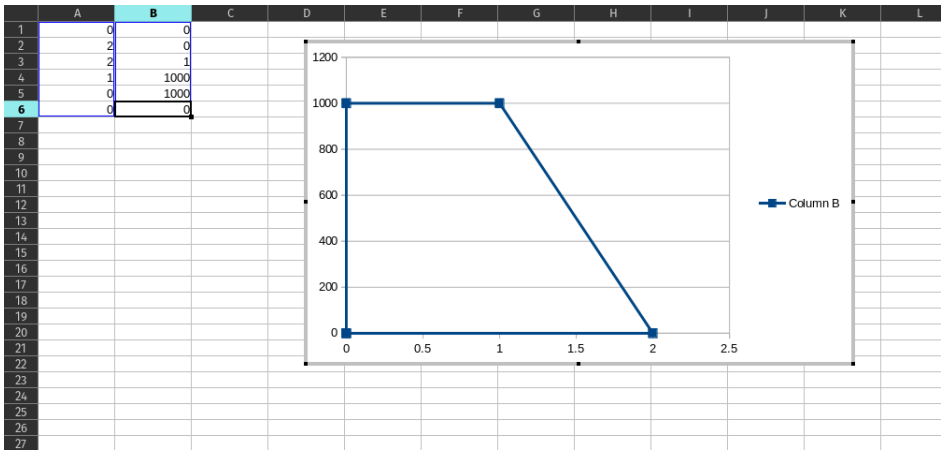
حال اگر بخواهیم مسئله را برای m=2500000 حل کنیم همه مراحل مانند قبل خواهد بود با این تفاوت که در هنگام استفاده از دستور split باید فایل را به فایل ها ۲ و نیم میلیون خطی بشکنیم و در هنگام نوشتن اسکریپت باید به عنوان ورودی ۲۵۰۰۰۰۰ را به جای ۵۰۰۰۰۰۰ بدهیم. بعد از اجرای دستور split چهارصد فایل ساخته می شود. زمان اجرای این دستور برابر با ۵ دقیقه و ۳۵ ثانیه می باشد. زمان اجرای اسکریپت برابر با ۱۶ دقیقه و ۱۲ ثانیه می باشد که در نهایت فایل final.txt را با تعداد ۲۵۹۸ خط تولید می کند اما چون زمان اجرا برنامه بر روی یکی از فایل ها برای ما مهم است بنابراین زمان یک اجرا را در نظر میگیریم که برابر با ۲ ثانیه می باشد. در شکل ۱۳ می توانید پوسته محدب این اجرای نمونه را مشاهده کنید. زمان مرتب سازی این فایل برابر با یک ثانیه و زمان اجرای الگوریتم بر روی آن نیز برابر با یک ثانیه خواهد بود. پس در مجموع ۵ دقیقه و ۳۹ ثانیه طول می کشد تا الگوریتم اجرا شود. پوسته محدب خروجی این الگوریتم نیز دقیقا برابر با شکل ۱۱ می شود.

حال اگر بخواهیم مسئله را برای m=2000000 حل کنیم همه مراحل مانند قبل خواهد بود با این تفاوت که در هنگام استفاده از دستور split باید فایل را به فایل ها ۲ میلیون خطی بشکنیم و در هنگام نوشتن اسکریپت باید به عنوان ورودی ۲۰۰۰۰۰۰ را به جای ۵۰۰۰۰۰۰ بدهیم. بعد از اجرای دستور split پانصد فایل ساخته می شود. زمان اجرای این دستور برابر با ۵ دقیقه و ۱۶



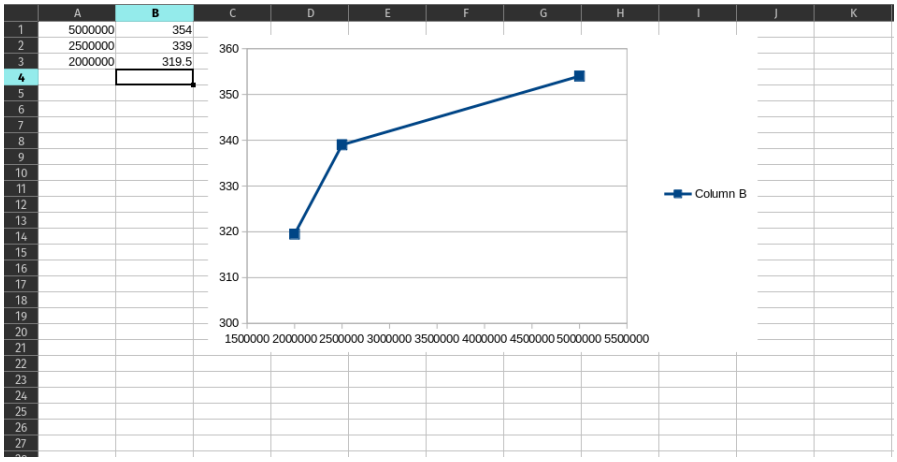
شکل ۱۳

ثانیه می باشد. زمان اجرای اسکریپت برابر با ۱۵ دقیقه و ۴۱ ثانیه می باشد که در نهایت فایل final.txt را با تعداد ۲۹۹۷ خط تولید می کند اما چون زمان اجرا برنامه بر روی یکی از فایل ها برای ما مهم است بنابراین زمان یک اجرا را در نظر میگیریم که برابر با ۵.۱ ثانیه می باشد. در شکل ۱۴ می توانید پوسته محدب این اجرای نمونه را مشاهده کنید. زمان مرتب سازی این فایل برابر با یک ثانیه و زمان اجرای الگوریتم بر روی آن نیز برابر با یک ثانیه خواهد بود. پس در مجموع ۵ دقیقه و ۵.۱۹ ثانیه طول می کشد تا الگوریتم اجرا شود. پوسته محدب خروجی این الگوریتم نیز دقیقا برابر با شکل ۱۱ می شود.



شکل ۱۴

در شکل ۱۵ می توانید نمودار زمان بر حسب m را ملاحظه کنید. توجه داشته باشید که زمان بر حسب ثانیه محاسبه شده است.



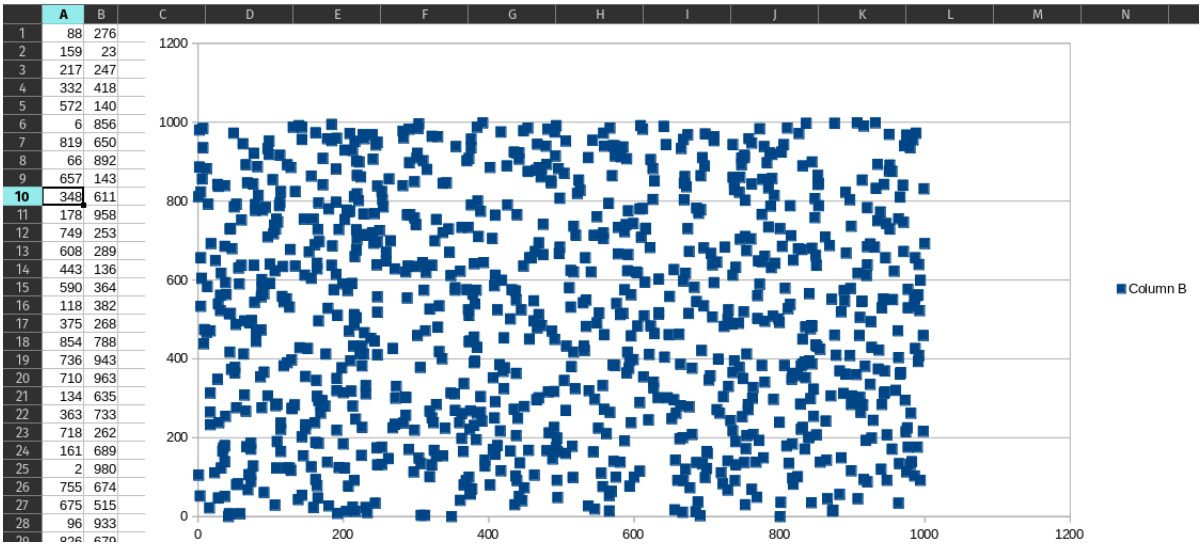
شکل ۱۵

در انتها به عنوان نمونه با استفاده از برنامه MakeFile یک فایل با ۱۰۰۰ نقطه می سازیم سپس آن را مرتب می کنیم و برنامه MapReduce را بر روی آن اجرا می کنیم تا پوسته محدب آن نقاط را بیابیم. در شکل ۱۶ دستورات لازم برای انجام این کار مشخص شده است.

```
kc@pop-os:~/Sample 1000$ java MakeFile
kc@pop-os:~/Sample 1000$ sort -g out.txt -o end.txt
kc@pop-os:~/Sample 1000$ java MapReduce end.txt 1000
```

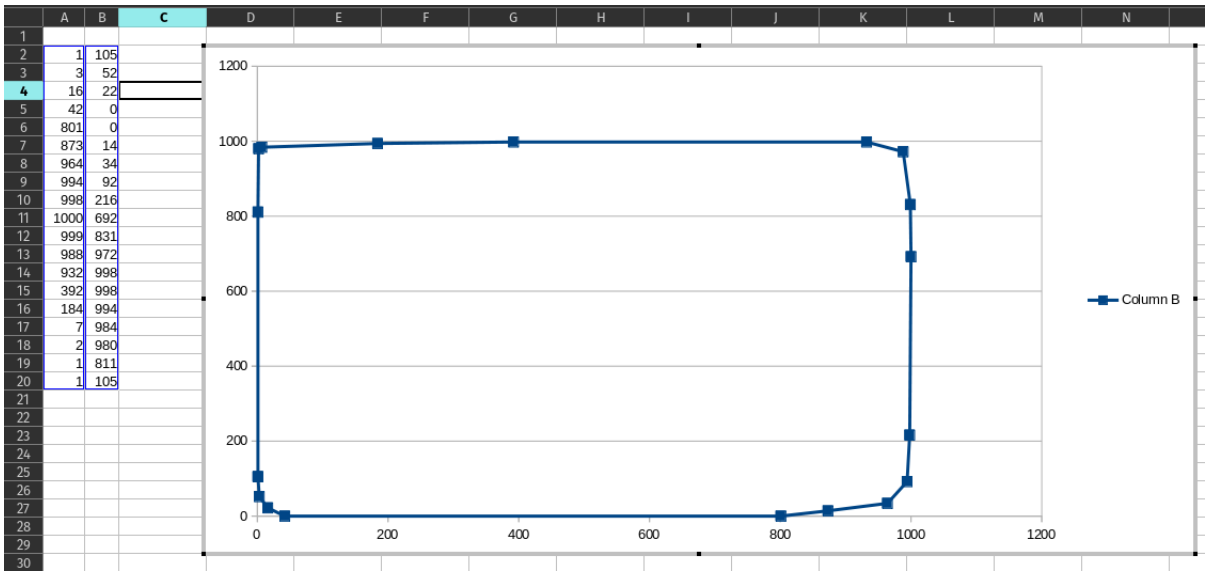
شکل ۱۶

در شکل ۱۷ می توانید نمودار این ۱۰۰۰ نقطه را ملاحظه کنید.



شکل ۱۷

در شکل ۱۸ نیز پوسته محدب این نقاط را ملاحظه می کنید.



شکل ۱۸

مراجع

[1] A. M. Andrew. Another Efficient Algorithm for Conex Hulls in Two Dimensions, Info. Proc. Letters9, 216-219, 1979.