@shihou22 2019年05月27日に更新

ABC - 091- A&B&C

AtCoder java8 競技プログラミング

AtCoder ABC 091 A&B&C

AtCoder - 091

2019/05/27

問題の名称を修正

C問題の `int[][] ab` および `int[][]cd` の生成個所のコードを修正

A - Two Coins

```
private void solveA() {
    Scanner scanner = null;
    int a = 0;
    int b = 0;
    int c = 0;

    try {
        scanner = new Scanner(System.in);
        a = scanner.nextInt();
        b = scanner.nextInt();
        c = scanner.nextInt();

        if (c <= (a + b)) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
```

```
        } finally {
            if (scanner != null) {
                scanner.close();
            }
        }
    }
```

B - Two Colors Card Game

古いソースなのでコード汚いけど。。。

1. 青いカードを調べて文字列ごとにカウントする
2. 赤い文字列を調べてカウントした文字列と比較し、同じだったらマイナスする

```
    private void solveB() {
        Scanner scanner = null;
        int n = 0;
        int m = 0;

        try {
            scanner = new Scanner(System.in);
            n = scanner.nextInt();

            Map<String, Integer> nArray = new HashMap<String, Integer>();
            for (int i = 0; i < n; i++) {
                String key = scanner.next();
                if (!nArray.containsKey(key)) {
                    nArray.put(key, 1);
                } else {
                    int count = nArray.get(key);
                    nArray.put(key, ++count);
                }
            }

            m = scanner.nextInt();

            for (int i = 0; i < m; i++) {
                String key = scanner.next();
                if (nArray.containsKey(key)) {
                    int count = nArray.get(key);
                    nArray.put(key, --count);
                }
            }
```

```
            int maxCount = 0;
            for (Iterator<Integer> iterator = nArray.values().iterator(); iterator.hasNex
                maxCount = Math.max(maxCount, iterator.next());
            }

            System.out.println(maxCount);

        } finally {
            if (scanner != null) {
                scanner.close();
            }
        }
    }
```

C - 2D Plane 2N Points

- 解説を参考に、問題文を図示してみると理解しやすい
  - 青い点より左下 $(x, y- > 0, 0)$ にある赤い点とペアを組ませる
  - **一番多くペアを組むには？**
    - `青い点をX軸順にソートして、Xが小さい方から判定していく`
    - $(x, y- > 0, 0)$ に近い赤い点ほど、青い点とペアを組みやすい
      - 青い点の(x, y)に近い赤い点をペアにする
        - x,yのどちらを優先するか？
          - 青い点が$(10, 10)$にあり赤い点が$(8, 9), (9, 8)$にあるときどちらと組ませるべきか？
            - y軸が大きい方を優先して組ませる
          - 青い点はX軸順にソートしてある。
            - 判定時に基準としている青い点(10,10)より小さいXの青い点は存在しないが、基準としている青い点(10,10)より小さいyの青い点は存在する可能性がある
  - ソートの仕方
    - 青い点：(x,y)->(0,0)に近い方が前に。ただし、YよりXが優先
      - X軸を優先して昇順ソート
      - Y軸は昇順
    - 赤い点：(x,y)->(0.0)に遠い方が前に。ただし、XよりYが優先
      - Y軸を優先して降順ソート

- X軸を優先すると、(9,8)(8,9)の時にループで(9,8)を選択してしまう
- 選択したいのは(8,9)

- X軸は降順

```
    private void solveC() {

        try (final Scanner scanner = new Scanner(System.in)) {
            final int n = scanner.nextInt();

            int[][] ab = Stream.generate(() -> new int[] { scanner.nextInt(), scanner.nex
                    .limit(n).sorted(new SortComparatorCForRed()).toArray(int[][]::new);
            //          int[][] ab = IntStream.range(0, n).collect(() -> new int[n][2],
            //                  (t, i) -> {
            //                      t[i][0] = scanner.nextInt();
            //                      t[i][1] = scanner.nextInt();
            //                  }, (t, u) -> {
            //                      Stream.concat(Arrays.stream(t), Arrays.stream(u));
            //                  });
            //
            //          Arrays.sort(ab, new SortComparatorCForRed());

            int[][] cd = Stream.generate(() -> new int[] { scanner.nextInt(), scanner.nex
                    .limit(n).sorted(new SortComparatorCForBlue()).toArray(int[][]::new);

            //          int[][] cd = IntStream.range(0, n).collect(() -> new int[n][2],
            //                  (t, i) -> {
            //                      t[i][0] = scanner.nextInt();
            //                      t[i][1] = scanner.nextInt();
            //                  }, (t, u) -> {
            //                      Stream.concat(Arrays.stream(t), Arrays.stream(u));
            //                  });
            //
            //          Arrays.sort(cd, new SortComparatorCForBlue());

            int[] memo = new int[n];

            int res = 0;
            for (int j = 0; j < cd.length; j++) {
                int bX = cd[j][0];
                int bY = cd[j][1];
                for (int k = 0; k < ab.length; k++) {
                    int rX = ab[k][0];
                    int rY = ab[k][1];
                    if (bX > rX && bY > rY && memo[k] != 1) {
                        memo[k] = 1;
                        res++;
```

```
                    //                      System.out.println(bX + ":" + bY + ":" +
                    break;
                }
            }
        }

        System.out.println(res);

    }

}

/*
 * int[][] X-> Y 順
 * int[][0]が小さいのが先(xが小さいのが先)
 * int[][1]が小さいのが先(yが小さいのが先)
 */
private static class SortComparatorCForBlue implements Comparator<int[]> {

    @Override
    public int compare(int[] o1, int[] o2) {
        if (o1[0] < o2[0]) {
            return -1;
        } else if (o1[0] > o2[0]) {
            return 1;
        } else {
            if (o1[1] < o2[1]) {
                return -1;
            } else if (o1[1] > o2[1]) {
                return 1;
            } else {
                return 0;
            }
        }

    }

}

/*
 * int[][] Y -> X順
 * int[][1]が大きいのが先(yが大きいのが先)
 * int[][0]が大きいのが先(xが大きいのが先)
 *
 */
private static class SortComparatorCForRed implements Comparator<int[]> {

    @Override
```

```
        public int compare(int[] o1, int[] o2) {
            if (o1[1] < o2[1]) {
                return 1;
            } else if (o1[1] > o2[1]) {
                return -1;
            } else {
                if (o1[0] < o2[0]) {
                    return 1;
                } else if (o1[0] > o2[0]) {
                    return -1;
                } else {
                    return 0;
                }
            }

        }

    }
```


@shihou22

暇なおじさん

フォロー

関連記事 Recommended by LOGLY

電気工事士の資格を取ってコンセントやスイッチを改造しようぜ

by rukihena

SQL素人でも分かるテーブル結合(inner joinとouter join)

by naoki_mochizuki

Linuxの権限確認と変更（超初心者向け）

by shisama

一番分かりやすい OAuth の説明

by TakahikoKawasaki

セキュリティの専門スキルを身につけてキャリアアップ！

PR パソナテック

この記事は以下の記事からリンクされています

 **AtCoder の進捗リスト** からリンク　1 year ago

コメント

この記事にコメントはありません。

あなたもコメントしてみませんか :)

ユーザ登録

すでにアカウントを持っている方はログイン