1 实验题目

1.1 使用并查集实现 MST 的 Kruskal 算法。

输入一张无向连通图，求最小生成树。

输入：多组数据输入。

每一组数据第一行两个正整数 n、m，分别表示图的点数和边数。

接下来 m 行每行三个正整数 u、v、w，在点 u 和 v 之间有一条权值为 w 的边。

输出：每组数据输出一行一个整数 x，表示给定的图的最小生成树上所有边权之和。

输入数据限制：

n <= 100000

m <= 100000

w <= 20000

输入所给的点编号从 1 开始。

输入所给的图保证连通。

1.2 实现 O(n)复杂度的查找第 k 小的数。

输入：多组数据输入。

每一组数据第一行两个正整数 n、k，其中 n 表示数列 A 的长度，k 意义见上文。

接下来一行 n 个非负整数，表示数列 A 中各元素值。

输出：每组数据输出一行一个非负整数 x，表示数列 A 中第 k 小的元素值。

输入数据限制：k <= n，A 中所有元素均不大于 2000000000

输入中的 k 是从 1 开始的。

2 设计思想

1.1 题 Kruskal 算法使用的是 MST 的蓝规则，按照贪心的顺序逐步添加边。

具体思想：将边按照权值排序，从最小权值的边开始遍历，若将其加入最小生成森林后不构成环，则将其加入最小生成森林，当所有的边都被遍历过之后，就可以得到最

小生成森林。

可以使用并查集对 Kruskal 来实现 Kruskal 算法，每个集合代表一个森林，新加入边后，若新边的两个端点位于不同的森林则可以将他加入 MSF，否则不可以。

1.2 题查找第 K 小的数算法使用分治策略，每次选出一个点作为分组点，将数据分成比他大的，比他小的和与他相等的三组，然后根据每组数据的数目和 k 的值决定在那个分组中递归查找。

选什么样的点作为分组点是算法的关键，若按照随机选点的方法，最坏情况下算法复杂度将是 O(n^2)，但是期望情况下是 O(n)。如果选用如下的策略，算法在最坏情况下的复杂度也可以成为 O(n)。这种策略即为，想将数据每五个分为一组，组内排序，取每组的中值作为一个新组，在新组上递归调用算法查找中值，将找到的这个值作为分组点。然后进行操作与随机取点作为分组点的是一样的。

3 具体实现

1.1

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <stdio.h>
using namespace std;

class Graph{
   typedef pair<int, int> edge;
   public:
      Graph(int v, int e) : V(v), E(e), parent(v + 1) {
         for(int i = 0; i < v + 1; ++i){
            parent[i] = i;
         }
      }
      void addEdge(int u, int v, int w){
         edges.push_back({w, {u, v}});
      }
      int KruskalMST();
   private:
      int find(int u){
         if(u != parent[u]){
            parent[u] = find(parent[u]);
```

```
            }
            return parent[u];
        }
        void merge(int u, int v){
            int fu = find(u);
            int fv = find(v);
            parent[fu] = fv;
        }
    private:
        vector<pair<int, edge>> edges;
        int V, E;
        vector<int> parent;
};

int Graph::KruskalMST(){
    int mst_wt = 0;
    sort(edges.begin(), edges.end());
    for(auto i = edges.begin(); i != edges.end(); ++i){
        int u = i->second.first;
        int v = i->second.second;
        int w = i->first;

        int fu = find(u);
        int fv = find(v);
        if(fu != fv){
            mst_wt += w;
            merge(fu, fv);
        }
    }
    return mst_wt;
}


int main(){
    int V, E;
#ifdef LOCAL
    freopen("input.txt", "r", stdin);
#endif
    while(scanf("%d %d", &V, &E) == 2){
        Graph G(V, E);
        int u, v, w;
        for(int i = 0; i < E; ++i){
            scanf("%d %d %d", &u, &v, &w);
            G.addEdge(u, v, w);
```

```
            }
            printf("%d\n", G.KruskalMST());
        }
        return 0;
}
```

1.2

```
    #include <iostream>
    #include <vector>
    #include <algorithm>
    #include <stdio.h>
    using namespace std;

    void swap(vector<int> &data, int i, int j){
        int temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    }

    int findKthMin(vector<int> &data, int low, int high, int k){
        if(low == high) return data[low];
        int pivot = data[low];
        int size = high - low    + 1;
        int Lbase = low - 1, Lsize = 0, Rbase = high + 1, Rsize = 0, EquMid = 0, Ebase =
Rbase;

        for(int i = low; Lsize + Rsize + EquMid != size;){
            if(data[i] < pivot){
                i++;
                Lbase++;
                Lsize++;
            }
            else{
                swap(data, i, --Rbase);
                if(data[Rbase] == pivot){
                    swap(data, Rbase, --Ebase);
                    EquMid++;
                }
                else{
                    Rsize++;
                }
            }
        }

        if(Lsize >= k){
```

```
            return findKthMin(data, low, Lbase, k);
        }
        else if(Lsize + EquMid >= k){
            return pivot;
        }
        else{
            return findKthMin(data, Rbase, high - EquMid, k - Lsize - EquMid);
        }
    }

    int main(){
        int total, k;
    #ifdef LOCAL
        freopen("input.txt", "r", stdin);
    #endif
        while(scanf("%d %d", &total, &k) == 2){
            if(total == 0) break;
            vector<int> data(total);
            for(int i = 0; i < total; ++i){
                scanf("%d", &data[i]);
            }
            printf("%d\n", findKthMin(data, 0, data.size() - 1, k));
        }
        return 0;
}
```

4 测试结果

使用李真同学提供的测试样例，地址：

https://github.com/EAirPeter/2017Fall-Algo/tree/master/exp2

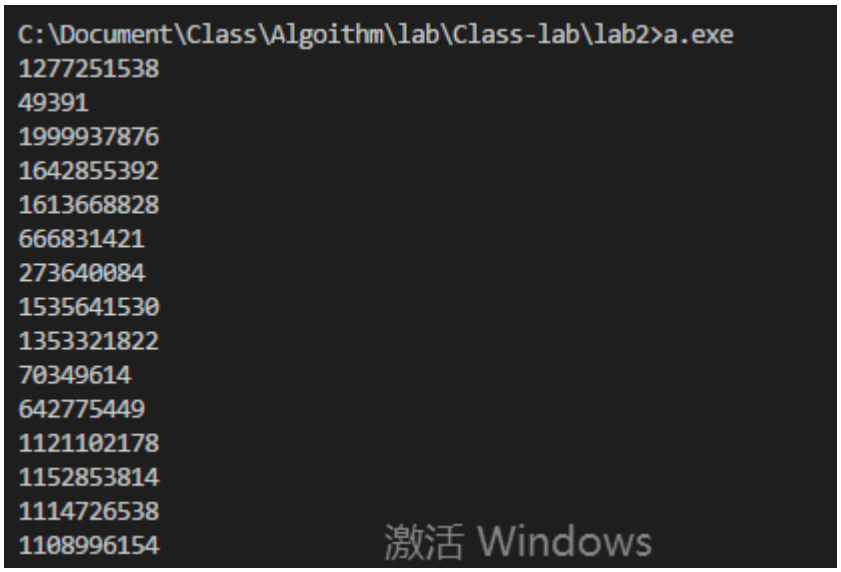
1.1 输出结果：

```
C:\Document\Class\Algoithm\lab\Class-lab\lab2>a.exe
812254142
12075079
115282
13275
1128
0
1000167548
1000390536
```

运行时间：

Real: 0m1.019s

1.2 测试结果



运行时间：

Real: 0m0.370s

使用 diff 与标准输出进行对比，没有错误。

5 总结与体会

由于之前做了很多并查集算法的训练，所以 1.1 题写起来很快，使用 stl 与 Union-Find 很快就完成了，但是做 1.2 题的时候遇到一些小问题。一开始没有将与分组点相等的值单独看作一个分组，导致在某些样例上表现十分差，找到问题修改后算法表现良好。