

**课 程 实 验 报 告**

**课程名称： 算法设计与分析实验**

**专业班级： ACM1501**

**学 号： U201514632**

**姓 名： 王宏斌**

**指导教师： 何琨**

**报告日期： 　2017年12月6日**

**计算机科学与技术学院**

# 一、 实验题目

1. Find the MST in a graph. Use "Union, Find" to implement the Kruskal algorithm.

2. Find the kth minimum element. Use divide and conquer to find the medium of the medium elements.

# 二、 设计思想

# 1. Kruskal 算法思想很简单，就是先将所有边按权重从小到大排序，然后从权重最小的边开始，通过find操作判断边的两个端点是否属于两个不同的集合，如果是则将边加入到MST的边集中，并将两个端点所在集合union。

# 2. 使用分治法找第k小，显然是通过快速选择，每次选取一个pivot，将数据分为大于pivot的集合S1(大小为size1)和小于pivot(大小为size2)的集合S2，如果size1 <= k，则在S2中通过快速选择继续找第k - size1小，否则在S1中通过快速选择继续找第k小。

# 三、 具体实现

# 1. Kruskal算法：

#include <iostream>

#include <algorithm>

#include <cstring>

using namespace std;

const int maxn = 1000;

struct Edge {

int x;

int y;

int weight;

} edges[maxn];

int parent[maxn];

int depth[maxn];

void makeSet() {

memset(parent, -1, sizeof(parent));

memset(depth, 0, sizeof(depth));

}

int find(int x) {

if (parent[x] == -1) {

return x;

}

parent[x] = find(parent[x]);

return parent[x];

}

void Union(int x, int y) {

int fx = find(x);

int fy = find(y);

if (depth[fx] > depth[fy]) {

parent[fy] = fx;

} else {

parent[fx] = fy;

if (depth[fx] == depth[fy]) {

++depth[fy];

}

}

}

int main() {

int n, m;

cin >> n >> m;

for (int i = 0; i < m; ++i) {

cin >> edges[i].x >> edges[i].y >> edges[i].weight;

}

sort(edges, edges + m, [](struct Edge a, struct Edge b) {

return a.weight < b.weight;

});

makeSet();

for (int i = 0; i < m; ++i) {

int fx = find(edges[i].x);

int fy = find(edges[i].y);

if (fx != fy) {

cout << edges[i].x << " -> " << edges[i].y << " : " << edges[i].weight << endl;

Union(edges[i].x, edges[i].y);

}

}

}

**2. quick select算法：**

#include <iostream>

using namespace std;

int partition(int \*input, int left, int right) {

int pivot = (input[left] + input[right]) / 2;

while (left < right) {

while (input[left] < pivot) {

++left;

}

while (input[right] > pivot) {

--right;

}

if (left < right) {

swap(input[left++], input[right--]);

}

}

return left;

}

int quick\_select(int \*input, int low, int high, int k) {

if (low >= high) {

return input[low];

}

int pi = partition(input, low, high);

int length = pi - low;

if (length < k) {

return quick\_select(input, pi, high, k - length);

} else {

return quick\_select(input, low, pi - 1, k);

}

}

int main() {

int n, k;

cin >> n >> k;

int input[n];

for (int i = 0; i < n; ++i) {

cin >> input[i];

}

cout << k << "th minimum: " << quick\_select(input, 0, n - 1, k) << endl;

return 0;

}

# 测试结果

# Kruskal算法：

1. Input:

|  |
| --- |
| 3 3 |
| 1 2 3 |
| 3 2 4 |
| 1 3 2 |

Output:

5

1. Input:

|  |
| --- |
| 5 6 |
| 4 5 7 |
| 4 2 2 |
| 1 2 1000 |
| 5 4 6 |
| 3 5 4 |
| 2 3 5 |

Output:

1011

**quick select 算法：**

1. Input:

|  |
| --- |
| 9 4 |
| 3 5 1 6 2 100 87 93 59 |

Output:

4th minimum: 5

2. Input:

|  |
| --- |
| 10 4 |
| 1 2 102 4 4 8 33 55 98 100 |

Output:

4th minimum: 4

# 总结与体会

通过union-find实现kruskal算法，我更加了解了union-find数据结构的使用，也熟悉了kruskal算法的实现，也在实现中进一步了解了union-find对kruskal算法的优化，使其时间复杂度由O(e^2)降为O(e loge)。通过divide-conquer实现找第k小，我更加熟悉了divide-conquer的思想及实现，不得不说，这次实验真的是收获颇丰。