

《算法分析与设计》0504 课后作业

2154312 郑博远

1. 最大多位整数问题

使用贪心的策略，将数字首先进行排序，再将排序后的数字按照从大到小的顺序进行排列即可。需要注意的是，对数字 a、b 进行排序时的比较依据应该是，a、b 拼合后的数字与 b、a 拼合后的数字的大小关系。

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <string>
#include <vector>
#include <queue>
#include <stack>
#include <map>
#include <set>
using namespace std;

class Solution {
private:
    //将每个整数转为string方便比较 存入vector中
    vector<string> numarray;

    /**
     * @brief 将数字数组转为string数组
     * @param nums 输入的数组数组
     */
    void convert(vector<int>& nums)
    {
        for (int i = 0; i < nums.size(); i++)
            numarray.push_back(to_string(nums[i]));
    }

    /**
     * @brief 定义string比较大小
     */
};
```

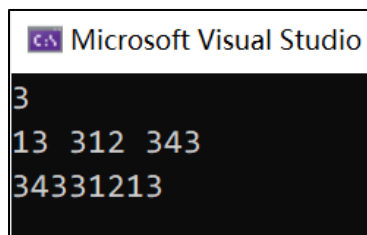
```

    * @param a 字符串a
    * @param b 字符串b
    */
    static bool LT(string a, string b)
    {
        return a + b < b + a;
    }
public:
    std::string largestNumber(std::vector<int>& nums)
    {
        string output;
        convert(nums);
        sort(numarray.begin(), numarray.end(), LT);
        for (int i = numarray.size() - 1; i >= 0 ; i--)
            output += numarray[i];
        return output;
    }
};

int main()
{
    int n;
    std::cin >> n;
    std::vector<int> nums(n);
    for (int i = 0; i < n; i++) {
        std::cin >> nums[i];
    }
    Solution s;
    std::cout << s.largestNumber(nums) << std::endl;
    return 0;
}

```

测试样例：

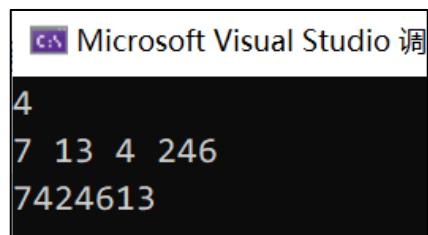


Microsoft Visual Studio

```

3
13 312 343
34331213

```



Microsoft Visual Studio 调

```

4
7 13 4 246
7424613

```

2. 克鲁斯卡尔最小生成树

根据克鲁斯卡尔算法进行最小生成树的生成，即每次选择不会形成环的权重最小边加入生成树中。使用并查集来记录顶点之间的关系，若一条边的两个顶点属于同一个集合则会形成回路。代码如下：

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// 记录边信息的结构体
struct edge {
    int start;
    int end;
    int weight;

    bool operator <(const edge& that)const
    {
        return weight < that.weight;
    }
};

// 查找根元素
int findroot(const vector<int>& root, int i)
{
    if (root[i] == i)
        return i;
    else
        return findroot(root, root[i]);
}

int main(){

    // 顶点个数与边的个数
    int vex_num, edge_num;

    cin >> vex_num >> edge_num;

    vector<edge> edges(edge_num);
    vector<int> root(edge_num);
```

```

for (int i = 0; i < edge_num; i++) {
    char vex;
    cin >> vex;
    // 默认输入下标从A开始
    edges[i].start = vex - 'A';
    cin >> vex;
    edges[i].end = vex - 'A';
    cin >> edges[i].weight;

    // 初始化root数组
    root[i] = i;
}

sort(edges.begin(), edges.end());

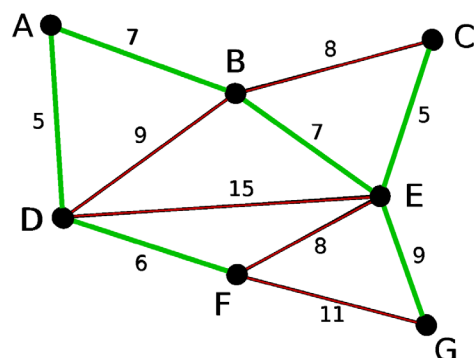
cout << endl;
for (int i = 0; i < edge_num; i++) {
    // 添加该边导致环 则跳过
    if (findroot(root, edges[i].start) == findroot(root, edges[i].end))
        continue;
    else {
        // 合并
        root[findroot(root, edges[i].start)] = edges[i].end;
        cout << char(edges[i].start + 'A') << " " << char(edges[i].end +
'A') << " " << edges[i].weight << endl;

    }
}

return 0;
}

```

测试样例：



Microsoft Visual Studio 调试控制台

```
7 11
A B 7
B C 8
A D 5
B D 9
D E 15
B E 7
D F 6
E F 8
F G 11
E G 9
C E 5

A D 5
C E 5
D F 6
A B 7
B E 7
E G 9
```