Battle Over Cities

张雯琪

Date: 2020-01-03

Chapter 1: Introduction

This project needs to find all the cities with the maximum cost if destroyed.

To solve this problem, it comes as the following steps. First divide all the cities apart, and for each city, find out all the additional connected cities. Secondly for each city that is not connected yet, find out the smallest cost when added and update the connected sequence, thus getting the cost of rebuilt the city net. Lastly all we need to do is to find out the maximum cost and output the cities.

Chapter 2: Algorithm Specification

Get the cost of destroying each city and pick up those with the maximum cost. In the getcost process, first find out all the already connected city sequence, then for each dropped city, find the road with the minimum cost to join the sequence, thus we have the minimum cost for each destroyed city.

2.1 Data structure

Record all the roads in an array of structure

```
struct RoadRecord
{
    int C1, C2;
    int Cost;
    int Status;
}Road[maxn];
```

2.2 Function analysis

2.2.1 getcost()

```
Function getcost(city a)
{
  Costsum <- 0
  For each valid road do
      Add the two cities into the connected sequence
  End for
  If the connected sequence is completed then
      Return costsum
  End if
  Quick sort // sort all the cities in the sequence in ascending order
  For each city that has not been connected do
      Costsum <- costsum + add(city)
  End for
  Return costsum
}
```

2.2.2 add()

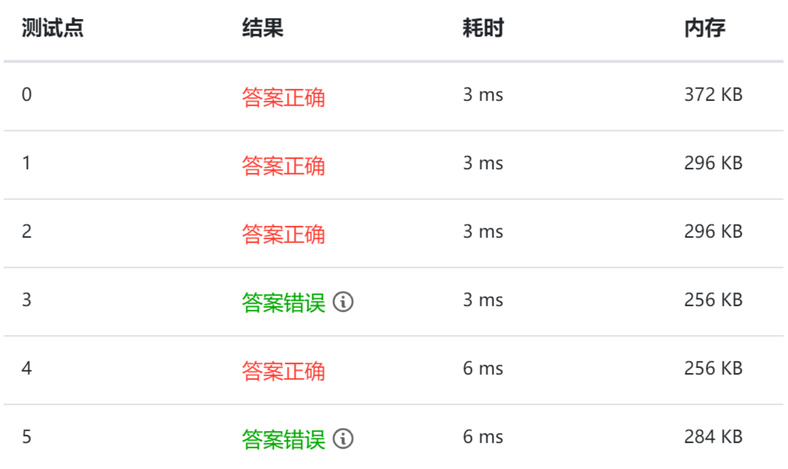
```
Function add(city a)
{
  Costrecord <- 0
  For each valid road to add city a to the connected sequence do
      Find the road with the minimum cost
      Costrecord <- costrecord + cost
  End for
  Return costrecord
}
```

2.2.3 Quick sort using C standard library function

void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))

```
qsort(cityseq, citynum, sizeof(int), cmp);
qsort(city, num, sizeof(int), cmp);
int cmp(const void* a, const void* b)
{
    return (*(int *)a - *(int *)b);
}
```

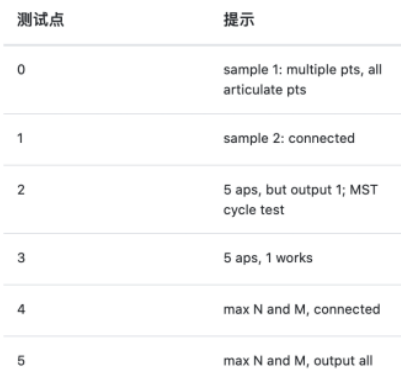
Chapter 3: Testing Results



测试点	结果	耗时	内存
0	答案正确	3 ms	372 KB
1	答案正确	3 ms	296 KB
2	答案正确	3 ms	296 KB
3	答案错误 ⓘ	3 ms	256 KB
4	答案正确	6 ms	256 KB
5	答案错误 ⓘ	6 ms	284 KB

For testing point 3 and 5, it's a pity that there are some mistakes, I tried but failed to figure them out.

Reference:



测试点	提示
0	sample 1: multiple pts, all articulate pts
1	sample 2: connected
2	5 aps, but output 1; MST cycle test
3	5 aps, 1 works
4	max N and M, connected
5	max N and M, output all

Chapter 4: Analysis and Comments

4.1 Time complexity

Function add()	-	O(NM)
Function getcost()	-	O(M+NlogN)
Quick sort algorithm	-	O(NlogN)

Thus the total time complexity is O(NM+N^2logN)

4.2 Space complexity

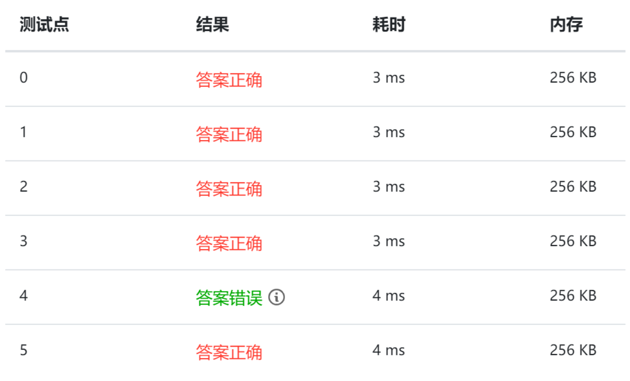
O(N+M), as for only one-dimension array.

4.3 Further improvements

One test case is that if all the cities can not be connected, then we have the maximum cost, but when I put

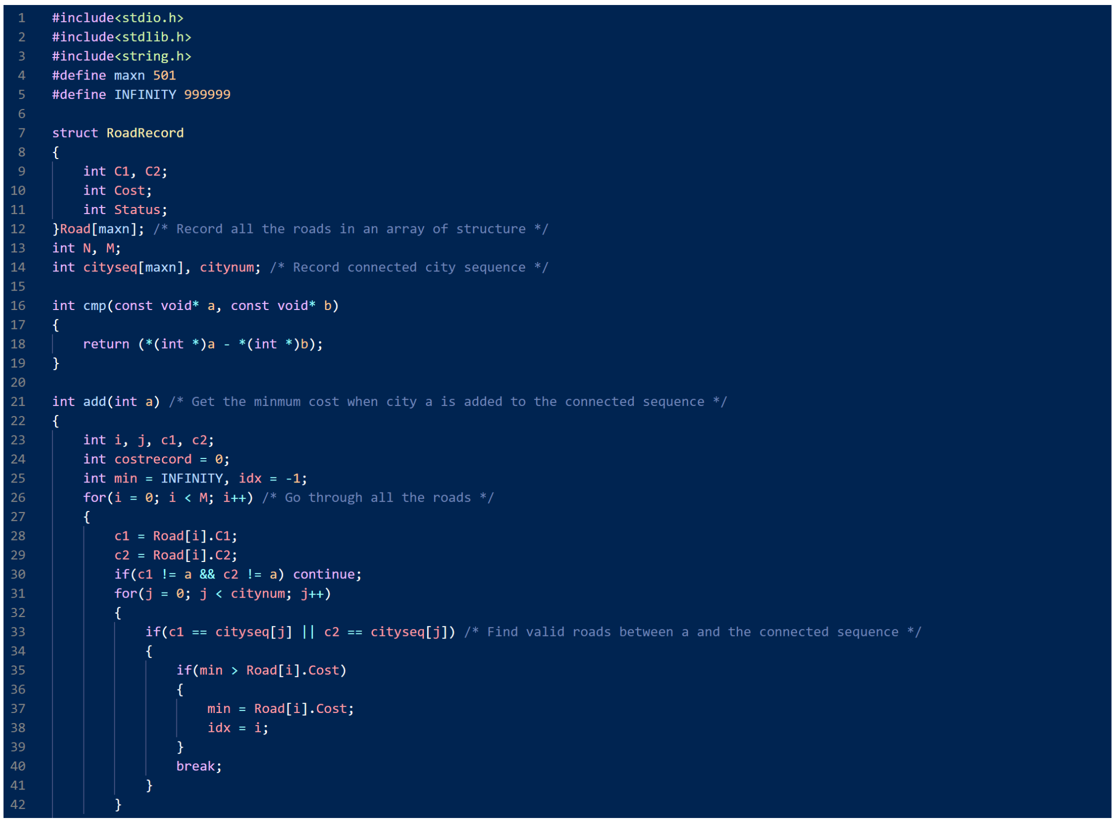
```
if(citynum == N-1) return costsum;
else return INFINITY;
```

into the function, it's really annoying that new mistakes occur.

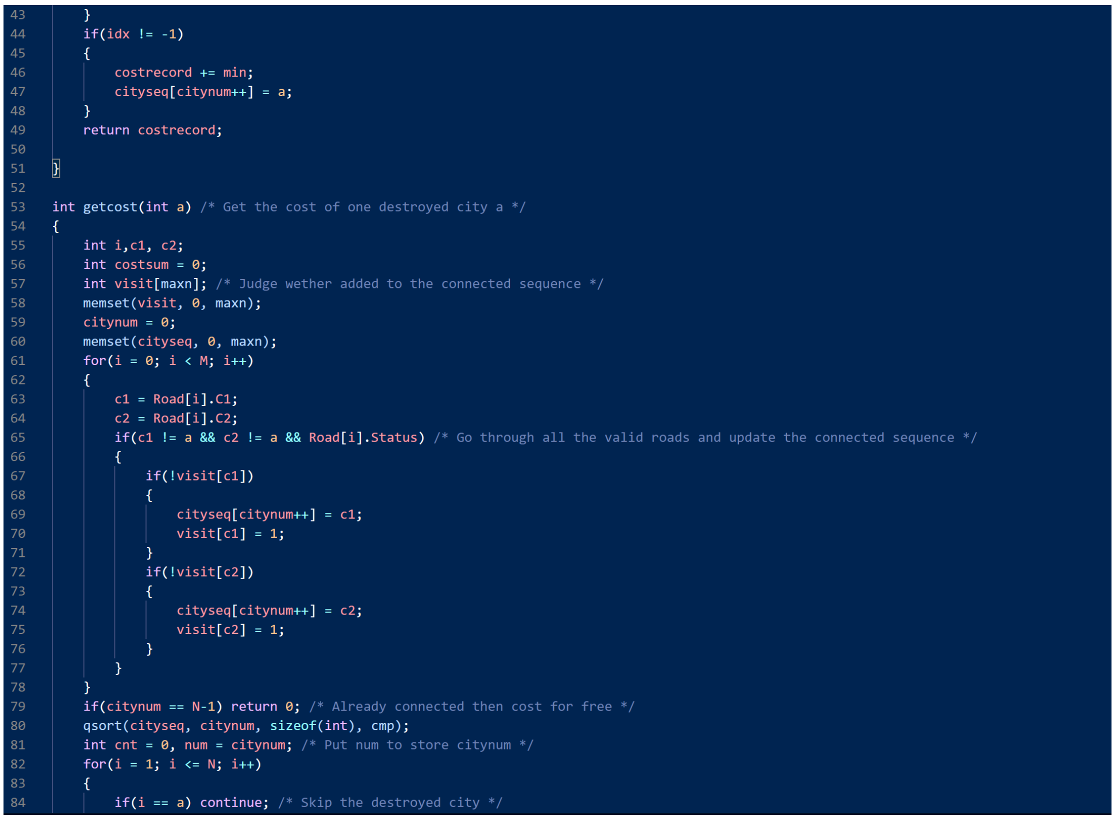


测试点	结果	耗时	内存
0	答案正确	3 ms	256 KB
1	答案正确	3 ms	256 KB
2	答案正确	3 ms	256 KB
3	答案正确	3 ms	256 KB
4	答案错误	4 ms	256 KB
5	答案正确	4 ms	256 KB

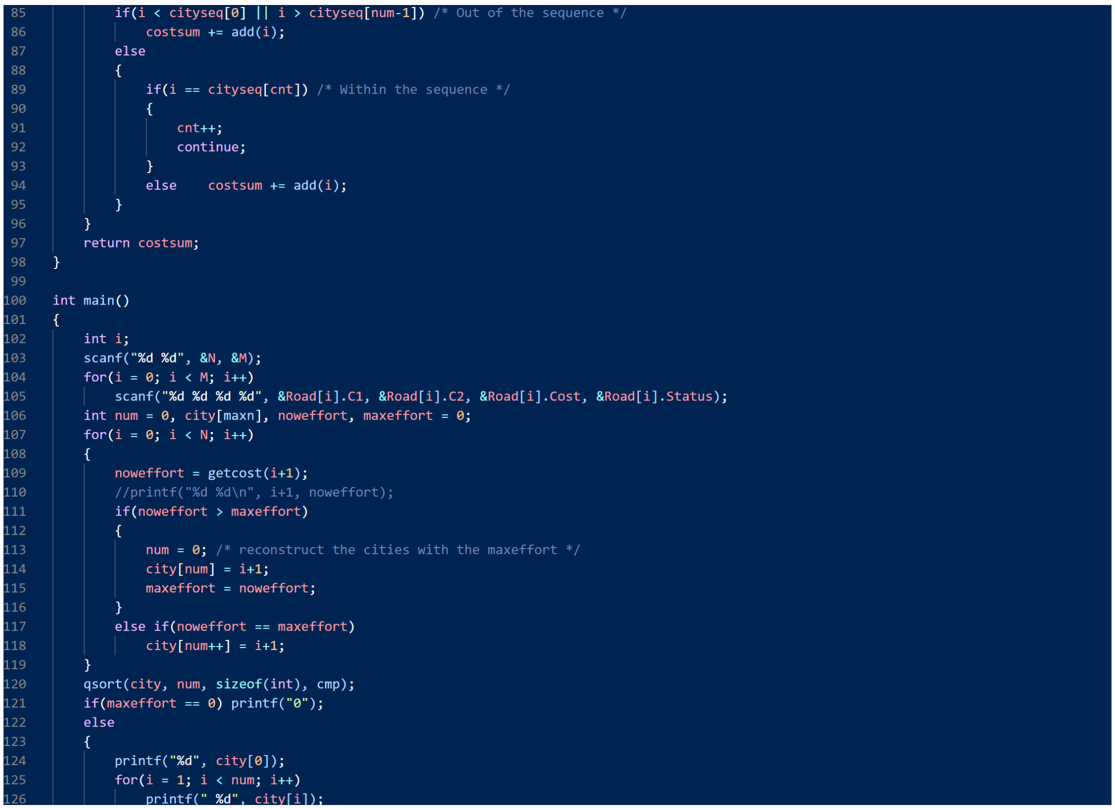
Appendix: Source Code (in C)



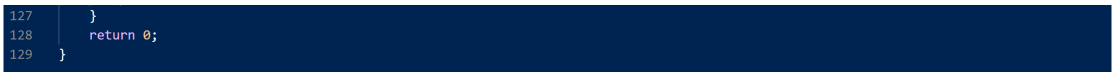
```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define maxn 501
#define INFINITY 999999

struct RoadRecord
{
    int C1, C2;
    int Cost;
    int Status;
}Road[maxn]; /* Record all the roads in an array of structure */
int N, M;
int cityseq[maxn], citynum; /* Record connected city sequence */

int cmp(const void* a, const void* b)
{
    return (*(int *)a - *(int *)b);
}

int add(int a) /* Get the minmum cost when city a is added to the connected sequence */
{
    int i, j, c1, c2;
    int costrecord = 0;
    int min = INFINITY, idx = -1;
    for(i = 0; i < M; i++) /* Go through all the roads */
    {
        c1 = Road[i].C1;
        c2 = Road[i].C2;
        if(c1 != a && c2 != a) continue;
        for(j = 0; j < citynum; j++)
        {
            if(c1 == cityseq[j] || c2 == cityseq[j]) /* Find valid roads between a and the connected sequence */
            {
                if(min > Road[i].Cost)
                {
                    min = Road[i].Cost;
                    idx = i;
                }
                break;
            }
        }
```



```c
    }
    if(idx != -1)
    {
        costrecord += min;
        cityseq[citynum++] = a;
    }
    return costrecord;

}

int getcost(int a) /* Get the cost of one destroyed city a */
{
    int i,c1, c2;
    int costsum = 0;
    int visit[maxn]; /* Judge wether added to the connected sequence */
    memset(visit, 0, maxn);
    citynum = 0;
    memset(cityseq, 0, maxn);
    for(i = 0; i < M; i++)
    {
        c1 = Road[i].C1;
        c2 = Road[i].C2;
        if(c1 != a && c2 != a && Road[i].Status) /* Go through all the valid roads and update the connected sequence */
        {
            if(!visit[c1])
            {
                cityseq[citynum++] = c1;
                visit[c1] = 1;
            }
            if(!visit[c2])
            {
                cityseq[citynum++] = c2;
                visit[c2] = 1;
            }
        }
    }
    if(citynum == N-1) return 0; /* Already connected then cost for free */
    qsort(cityseq, citynum, sizeof(int), cmp);
    int cnt = 0, num = citynum; /* Put num to store citynum */
    for(i = 1; i <= N; i++)
    {
        if(i == a) continue; /* Skip the destroyed city */
```



```c
        if(i < cityseq[0] || i > cityseq[num-1]) /* Out of the sequence */
            costsum += add(i);
        else
        {
            if(i == cityseq[cnt]) /* Within the sequence */
            {
                cnt++;
                continue;
            }
            else    costsum += add(i);
        }
    }
    return costsum;
}

int main()
{
    int i;
    scanf("%d %d", &N, &M);
    for(i = 0; i < M; i++)
        scanf("%d %d %d %d", &Road[i].C1, &Road[i].C2, &Road[i].Cost, &Road[i].Status);
    int num = 0, city[maxn], noweffort, maxeffort = 0;
    for(i = 0; i < N; i++)
    {
        noweffort = getcost(i+1);
        //printf("%d %d\n", i+1, noweffort);
        if(noweffort > maxeffort)
        {
            num = 0; /* reconstruct the cities with the maxeffort */
            city[num] = i+1;
            maxeffort = noweffort;
        }
        else if(noweffort == maxeffort)
            city[num++] = i+1;
    }
    qsort(city, num, sizeof(int), cmp);
    if(maxeffort == 0) printf("0");
    else
    {
        printf("%d", city[0]);
        for(i = 1; i < num; i++)
            printf(" %d", city[i]);
```



```c
    }
    return 0;
}
```

Declaration

I hereby declare that all the work done in this project titled "Bonus2_BattleOverCities" is of my independent effort.