contest 8题解

未分类

D题 小熊与团间距离

这是一道图上的问题，需要先对图这种数据结构以及图的遍历有一定了解，对图上的最短路算法比较熟悉
可以先去xjoi 3-1练习基本的图与树上的题目

dijkstra算法的基本思路是给每个点设置一个d值表示起点到每个点的最短路，一开始d[i] = inf, d[s] = 0;因为只有起点是确定最短路的，然后每次在未确定最短路的点集中寻找一个d值最小的点，假设叫做x，那么x这个点的最短路就不可能再被别的点更新了，因为别的点的最短路都大于等于它，所以我们把x标记掉，然后枚举与x相邻的点，用x的距离值加上相邻边的权值去更新相邻点的d值
重复n-1次就能得出所有点的最短路

本题在最短路模型的基础上稍加变形，因为前K个点两两之间是有边的，但是我们无法真的去建立这么多边，我们观察到如果某个点到达这K个点然后再从这K个点的另一个点出来，无论从哪个点出来，经过的距离值一定是题目中告诉我们的x，所以我们可以增加一个额外的中转节点T，让前K个点跟T之间都建立一条双向边，距离为x/2,这样的话就只需要建立K条双向边即可转换问题。由于x/2可能是浮点数，因此我们将所有边权都扩大两倍，最后跑出最短路后再除以2就可以了。
代码：

```
#include<bits/stdc++.h>
using namespace std;
const int N=100005;
const int inf=-1;
vector<pair<int,int> > e[N];
int T,n,m,k,s,x;
long long d[N];
inline void solve()
{
```

```
    priority_queue<pair<long long,int> > q;//大根堆，会先按照第一维关键字排序
    for(int i=1;i<=n;i++)
    {
        d[i]=inf;//为了避免inf太大爆炸，设置inf = -1
    }
    d[s]=0;
    q.push(make_pair(0,s));//初始化将起点扔进堆里
    while(!q.empty())
    {
        pair <long long,int> cur = q.top();
        cur=q.top();
        q.pop();
        cur.first=-cur.first;
        int u = cur.second;
        if(cur.first > d[u]) continue;
        int sz=e[u].size();
        for(int i=0;i<sz;i++) //更新u的邻接点的最短距离值
        {
            int v = e[u][i].first;
            int w = e[u][i].second;
            if(d[v]==inf||d[u]+w<d[v])
            {
                d[v]=d[u]+w;
                q.push(make_pair(-d[v],v));//我们要求最小值，所以变成相反数扔进去
            }
        }
    }
    for(int i=1;i<=n-1;i++)
    {
        printf("%lld ",d[i]/2);
    }
    printf("\n");
}
int main()
{
    scanf("%d",&T);
    while(T--)
    {

        scanf("%d%d%d%d%d",&n,&k,&x,&m,&s);
        for(int i=1;i<=n+1;i++)//清空邻接表
        {
            e[i].clear();
```

```
            }
            while(m--)
            {
                int a,b,c;
                scanf("%d%d%d",&a,&b,&c);
                e[a].push_back(make_pair(b,2*c));//加双向边
                e[b].push_back(make_pair(a,2*c));
            }
            n++;//新建一个点,与前K个点连边
            for(int i=1;i<=k;i++)
            {
                e[n].push_back(make_pair(i,x));
                e[i].push_back(make_pair(n,x));
            }
            solve();
        }
    }
```

I题 烹饪计划

这是一道比较明显的二分查找的题目
二分答案mid之后，对于每一段连续的1 或者0的个数超过mid的区间，我们需要花费最少的修改次数去搞定，比如111111111，mid=3，我们需要修改成111011101，即每4个修改一个，最后根据总的修改次数是否<=K来判断二分的条件是否成立，由于mid=1的情况比较特殊，比如00011000，按照二分的逻辑，需要的最小代价为3，但是实际上需要修改四次，但是mid=1的时候相当于都改成01010101或者10101010，直接特判断即可
代码

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1000010;
char s[N];

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n, k;
```

```
        scanf("%d%d", &n, &k);
        scanf("%s", s);
        int cost0 = 0, cost1 = 0;
        for (int i = 0; s[i]; i++) {
            if (i & 1) {
                cost0 += (s[i] == '0');
                cost1 += (s[i] == '1');
            } else {
                cost0 += (s[i] == '1');
                cost1 += (s[i] == '0');
            }
        }
        int mi = min(cost0, cost1);

        int l = 1, r = n, best = -1;
        while (l <= r) {
            int mid = (l + r) >> 1;
            int cost = 0;
            if (mid == 1) {
                cost = mi;
            } else {
                int cnt[2] = {0};
                for (int i = 0; s[i]; i++) {
                    if (i == 0) {
                        cnt[s[i] - '0']++;
                    } else {
                        if (s[i] == s[i - 1]) {
                            cnt[s[i]-'0']++;
                        } else {
                            int len = cnt[!(s[i]-'0')];
                            cost += len / (mid + 1);
                            cnt[s[i]-'0'] = 1;
                            cnt[!(s[i]-'0')] = 0;
                        }
                    }
                }
                int mx = max(cnt[0],cnt[1]);
                cost += mx / (mid+1);
            }
            if (cost <= k) {
                best = mid;
                r = mid - 1;
            } else {
                l = mid + 1;
            }
```

```
            }
            printf("%d\n", best);
        }
        return 0;
    }
```