萌新赛题解

七夕

考察最小生成树算法。

将各个城市看作点，城市之间以铁路或城际公交相连，城际公交免费乘坐，铁路需要买票，问需要买多少张票才能保证见到女朋友。可以将城际公交看作边权为0的边，铁路看作边权为1的边，使用最小生成树算法即可在确保边权最小的情况下使得图连通。

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <time.h>
#include <string>
#include <map>
#include <stack>
#include <vector>
#include <set>
#include <queue>
#define inf 0x3f3f3f3f
#define mod 10000
typedef long long ll;
using namespace std;
const int N=10005;
const int M=100005;
struct Edg {
    int  v,u;
    int w;
} edg[M];
bool cmp(Edg g,Edg h) {
    return g.w<h.w;
}
int n,m,maxn,cnt,k;
int parent[N];
int a[N];
void init() {
    for(int i=0; i<n; i++)parent[i]=i;
}
void Build() {
    int  u,v;
    while(k--){
        scanf("%d%d",&u,&v);
        edg[++cnt].u=u;
        edg[cnt].v=v;
        edg[cnt].w=0;
    }
```

```
    while(m--){
        scanf("%d%d",&u,&v);
        edg[++cnt].u=u;
        edg[cnt].v=v;
        edg[cnt].w=1;
    }
    sort(edg,edg+cnt+1,cmp);
}
int Find(int x) {
    if(parent[x] != x) parent[x] = Find(parent[x]);
    return parent[x];
}
void Union(int x,int y) {
    x = Find(x);
    y = Find(y);
    if(x == y) return;
    parent[y] = x;
}
void Kruskal() {
    int sum=0;
    int num=0;
    int u,v;
    for(int i=0; i<=cnt; i++) {
        u=edg[i].u;
        v=edg[i].v;
        if(Find(u)!=Find(v)) {
            sum+=edg[i].w;
            num++;
            Union(u,v);
        }
        if(num>=n-1) {
            printf("%d\n",sum);
            break;
        }
    }
}
int main() {
    scanf("%d%d%d",&n,&k,&m);
    if(m==0)printf("0\n"),exit(0);
    if(n==1)printf("0\n"),exit(0);
    cnt=-1;
    init();
    Build();
    Kruskal();
    return 0;
}
```

带路

考察搜索算法。

从任一可达点出发，用dfs遍历找到最远的路，最后输出即可。

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

int n, m;
char graph[1005][1005];
bool visited[1005][1005];

int dfs(int i, int j) {

    if (i < 0 || i >= n || j < 0 || j >= m || visited[i][j] || graph[i][j] == '#') {
        return 0;
    }

    visited[i][j] = true;

    int cnt = 0;
    cnt = max(cnt, dfs(i+1, j));
    cnt = max(cnt, dfs(i-1, j));
    cnt = max(cnt, dfs(i, j+1));
    cnt = max(cnt, dfs(i, j-1));

    visited[i][j] = false;

    return cnt + 1;
}

int find_max_reachable() {
    int max_count = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (!visited[i][j] && graph[i][j] == '.') {
                max_count = max(max_count, dfs(i, j));
            }
        }
    }

    return max_count;
}

int main() {
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; ++i) {
        cin >> graph[i];
    }
    int result = find_max_reachable();
    printf("%d\n", result);

    return 0;
}
```

斗牛

简单的模拟题。按照规则逐步模拟，先判断是否存在三张牌之和为10的倍数，再计算点数，点数相同再通过花色比较即可。

```
#include <iostream>
#include <vector>
#include <string>
#include <map>

using namespace std;

int get_score(string card) {
    if (card[card.size() - 1] == 'J' || card[card.size() - 1] == 'Q' ||
card[card.size() - 1] == 'K' || card.find("10") != string::npos)
        return 10;
    else if (card.find('A') != string::npos)
        return 1;
    else
        return card[card.size() - 1] - '0';
}

int calculate_score(vector<string> cards) {
    int total = 0;
    for (int i = 0; i < cards.size(); i++)
        total += get_score(cards[i]);
    return total;
}

int calculate_niu(vector<string> cards) {
    for (int i = 0; i < 5; i++) {
        for (int j = i+1; j < 5; j++) {
            for (int k = j+1; k < 5; k++) {
                int total = calculate_score(cards) - get_score(cards[i]) -
get_score(cards[j]) - get_score(cards[k]);
                if (total % 10 == 0) return ((calculate_score(cards) - total) - 1) %
10 + 1;
             }
        }
    }
    return 0;
}

int compare(vector<string> game1, vector<string> game2) {
    int niu1 = calculate_niu(game1);
    int niu2 = calculate_niu(game2);

    if (niu1 != niu2)
        return niu2 - niu1;

    if (niu1 == 10)
```

```
        return 0;

    int score1 = calculate_score(game1);
    int score2 = calculate_score(game2);

    if (score1 != score2)
        return score2 - score1;

    map<string, int> suit_order;
    suit_order["heitao"] = 4;
    suit_order["hongtao"] = 3;
    suit_order["meihua"] = 2;
    suit_order["fangkuai"] = 1;

    string suit1;
    string suit2;
    suit1 = game1[0].substr(0, game1[0].length()-1);
    suit2 = game2[0].substr(0, game2[0].length()-1);

    return suit_order[suit2] - suit_order[suit1];
}

int main() {
    int n;
    cin >> n;
    vector< vector<string> > games(n, vector<string>(5));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < 5; j++)
            cin >> games[i][j];

    int max_index = 0;
    for (int i = 1; i < n; i++) {
        if (compare(games[i], games[max_index]) < 0)
            max_index = i;
    }
    cout << max_index << endl;
    return 0;
}
```

撸铁

这个问题本质就是字符串的不同子串的个数，考察后缀数组算法。

后缀数组实际上就是把字符串的所有**后缀**按**字典序**排序后得到的数组。

注意到所有后缀的前缀集合就是的子串集合。考虑先将所有的后缀排序。对于第一个后缀，不同的前缀就是它本身的长度，对于第二个后缀，不同的前缀数量就是它本身的长度减去它和第一个后缀的最长公共子串。

对于后面的所有后缀，根据$lcp(i,j)=\min(lcp(i,k),(k,j))(i\le k\le j)$，$\forall j<i-1$必然有$lcp(i,i-1)>lcp(i,j)$。意思是 i 和前面所有后缀的最长公共前缀就是和前一个后缀的最长公共前缀。

记 $len[i]$表示排序后第 i 个后缀的长度。那么总的不同子串数就是$\sum len[i]-height[i]$。

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int MAXN=1e6+10;
int sa[MAXN],rank[MAXN],rsort[MAXN],y[MAXN],wr[MAXN],height[MAXN];
char a[MAXN];
bool cmp(int a,int b,int len)
{
    return wr[a]==wr[b] && wr[a+len]==wr[b+len];
}
void get_SA(int m,int n)
{
    for(int i=1;i<=n;i++) rank[i]=a[i-1];
    for(int i=1;i<=n;i++) rsort[rank[i]]++;
    for(int i=1;i<=m;i++) rsort[i]+=rsort[i-1];
    for(int i=n;i>0;i--) sa[rsort[rank[i]]--]=i;
    int len=1,p=0;
    while(p<n)
    {
        int k=0;
        for(int i=n-len+1;i<=n;i++) y[++k]=i;
        for(int i=1;i<=n;i++) if(sa[i]>len) y[++k]=sa[i]-len;
        for(int i=1;i<=n;i++) wr[i]=rank[y[i]];
        memset(rsort,0,sizeof(rsort));
        for(int i=1;i<=n;i++) rsort[wr[i]]++;
        for(int i=1;i<=m;i++) rsort[i]+=rsort[i-1];
        for(int i=n;i>0;i--) sa[rsort[wr[i]]--]=y[i];
        for(int i=1;i<=n;i++) wr[i]=rank[i];
        p=1;rank[sa[1]]=1;
        for(int i=2;i<=n;i++)
        {
            if(!cmp(sa[i],sa[i-1],len)) p++;
            rank[sa[i]]=p;
        }
        m=p;len<<=1;
    }
}
void get_height(int n)
{
    int k=0,j;
    for(int i=1;i<=n;i++)
    {
        j=sa[rank[i]-1];
        if(k) k--;
        while(a[j+k-1]==a[i+k-1]) k++;
        height[rank[i]]=k;
    }
}
ll solve(int n)
{
    ll ans=0;
    for(int i=1;i<=n;i++)
```

```
        ans+=n+1-sa[i]-height[i];
    return ans;
}
int main()
{
    int len;
    int fd;
    scanf("%s",a);
    len = strlen(a);
    get_SA(300,len);
    get_height(len);
    printf("%lld",solve(len));
    return 0;
}
```

XOR的艺术

线段树+懒标记模板题。

懒标记是标记这个区间是否翻转，两次翻转等于不翻转，所以懒标记的更新是异或操作。

维护任意一个区间的1的个数sum。

```
#include <bits/stdc++.h>
#define maxx 300020
#define ll long long
using namespace std;
int n,m,p,l,r,a[maxx];
char s[maxx];
ll add[maxx<<2],sum[maxx<<2];
//add是懒标记
void pushup(int rt)//向上更新
{
    sum[rt]=sum[rt<<1]+sum[rt<<1|1];
}
void bulid(int l,int r,int rt)//建树
{
    if(l==r)
    {
        sum[rt]=a[l];//赋值
        return ;
    }
    int mid=(l+r)>>1;
    bulid(l,mid,rt<<1);
    bulid(mid+1,r,rt<<1|1);
    pushup(rt);
}
void pushdown(int rt,int len)//懒标记下传
{
    if(add[rt])
    {
        add[rt<<1]^=1;
```

```
        add[rt<<1|1]^=1;
        sum[rt<<1]=(len-(len>>1))-sum[rt<<1];
        sum[rt<<1|1]=(len>>1)-sum[rt<<1|1];
        add[rt]=0;
    }
}

//L、R是要查询的区间，l、r是给定的区间。update同理。
ll query(int L,int R,int l,int r,int rt)//分区间查询
{
    if(l>=L&&r<=R)return sum[rt];
    pushdown(rt,r-l+1);
    int mid=(r+l)>>1;
    ll tot=0;
    if(L<=mid) tot+=query(L,R,l,mid,rt<<1);
    if(mid+1<=R) tot+=query(L,R,mid+1,r,rt<<1|1);
    return tot;
}
void update(int L,int R,int l,int r,int rt)//更新
{
    pushdown(rt,r-l+1);
    if(l>=L&&r<=R)
    {
        add[rt]^=1;
        sum[rt]=r-l+1-sum[rt];
        return;
    }
    int mid=(l+r)>>1;
    if(L<=mid) update(L,R,l,mid,rt<<1);
    if(mid+1<=R) update(L,R,mid+1,r,rt<<1|1);
    pushup(rt);
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) cin>>s[i];//先读入字符
    for(int i=1;i<=n;i++) a[i]=s[i]-'0';//再转换成数字
    bulid(1,n,1);
    for(int i=1;i<=m;i++)
    {
        scanf("%d%d%d",&p,&l,&r);
        if(p==0)
            update(l,r,1,n,1);
        if(p==1)
            cout<<query(l,r,1,n,1)<<'\n';
    }
    return 0;
}
```

调手表

简单的BFS模板题。

用一个标记数组表示到达每一个点数的最小步数是否已经出现。

用ans不断维护最小步数的最大值。

每一个状态扩展出两个状态：即 x+1 和 x+k （mod n）

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int n, k, t[N], x, Y1, y2, ans;
bool bj[N];
queue <int> q;
int main()
{
    scanf("%d %d", &n, &k);
    bj[0] = true;
    q.push(0);
    while(!q.empty())
    {
        x = q.front();
        q.pop();
        ans = max(ans, t[x]);
        Y1 = (x + k) % n, y2 = (x + 1) % n;
        if(!bj[Y1])
        {
            t[Y1] = t[x] + 1, bj[Y1] = true;
            q.push(Y1);
        }
        if(!bj[y2])
        {
            t[y2] = t[x] + 1, bj[y2] = true;
            q.push(y2);
        }
    }
    printf("%d", ans);
    return 0;
}
```

跑路

如果不存在空间跑路器的话，这道题就是一个简单的最短路问题，并且每条边的长度都是1。

那么如果存在空间跑路器，无非就是对于任意两个可以1s到达的点之间连上边。

这里用到倍增的思想，首先寻找到所有可以1s到达的点，并将其距离设置为1。

然后使用floyd，求解最短路即可。

具体分析见代码注释。

```
#include<bits/stdc++.h>
```

```cpp
using namespace std;
int dis[60][60],n,m;
bool G[60][60][65];
/*以上是变量说明部分，dis[i][j]表示i到j的路径/边的长度
G[i][j][k]表示，i到j是否存在一条长度为2^k的路径
如果有，为true，没有就是false*/
void init()
{
    memset(dis,0x3f,sizeof(dis));
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        dis[x][y]=1;
        G[x][y][0]=true;
        /*初始化，x到y的路径（边）最短是1，也就是x到y存在
        一条长度为2^0的路径（边）*/
    }
}
void work()//此函数对G和dis做预处理
{
    for(int k=1;k<=64;k++)
    //对于本题的数据，2^64已经足够。
        for(int i=1;i<=n;i++)
            for(int t=1;t<=n;t++)
                for(int j=1;j<=n;j++)
                    //枚举三个点
                    if(G[i][t][k-1]&&G[t][j][k-1])
                    /*如果i到t存在一条2^k-1长度的路径
                    并且t到j存在一条2^k-1长度的路径
                    就说明i到t，t到j都可以一秒到达，
                    路程*2刚好是2的幂，也可以一秒到达*/
                    {
                        G[i][j][k]=true;
                        //标记从i到j存在一条长度为2^k的路径
                        dis[i][j]=1;
                        //i到j距离可以一秒到达
                    }
}
void floyd() //Floyd图论求最短路。
{
    for(int k=1;k<=n;k++)
        //这里的注意点：枚举中间点的循环放在最前面
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
                //松弛操作。
}
int main()
{
    init();
```

```
    work();
    floyd();
    printf("%d",dis[1][n]);
    return 0;
}
```

题解

单调队列，维护一个单调递增的双端队列。

（1）当队首元素已经不在所求的区间范围内时，则弹出队首元素。

（2）当需要插入新元素时，不断比较队尾元素和新元素，若队尾元素较大，则弹出。反复执行直到队尾元素小于新元素。

最后输出队首元素，就是当前区间的最小值了。

```
#include <bits/stdc++.h>
using namespace std;
int n, m;
int q1[1000010];//q为a数组内元素的下标
int a[1000010];

void min_deque() {
    int h = 1, t = 0;
    for (int i = 1; i <= n; i++) {
        while (h <= t && q1[h] + m <= i) h++; //如果队首元素已经不在区间内，弹出
        while (h <= t && a[i] < a[q1[t]]) t--; //如果队尾元素大于新元素，弹出
        q1[++t] = i;//新元素入队
        if(i >= m) printf("%d\n", a[q1[h]]);//输出当前区间的最小值
    }
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    min_deque();
    return 0;
}
```

传纸条

考虑三维dp，对于每次转移，这两位同学的纸条走的步数总是相等的，也就是应该总有i+j = k+l = step。所以三维dp我们考虑枚举走的步数，同时剩下枚举第一个人第二个人的横坐标或者纵坐标。四维dp可以通过，但数据严格一点应该就会超时。

```
#include <bits/stdc++.h>
using namespace std;
int f[60][60][60][60], a[60][60];
```

```
int n ,m;
int main(){
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
                cin >> a[i][j];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            for (int k = 1; k <= n; k++)
                for (int l = j + 1; l <= m; l++)
                    f[i][j][k][l] = max({f[i][j-1][k-1][l],f[i-1][j][k][l-1],f[i][j-
1][k][l-1],f[i-1][j][k-1][l]})+a[i][j]+a[k][l];
    cout << f[n][m-1][n-1][m] << endl;
    return 0;
}
```

借教室

利用差分数组存每天的教室使用情况，然后求前缀和，如果发现不符合要求，就从后往前撤回订单，直到每天都符合要求，那么我们撤回的最后一个即为ans

```
#include <bits/stdc++.h>
const int I=1000005;
int a[I],c[I],l[I],r[I],n,m,x=-1;
long long cf[I];
long long sum;
bool flag=1;
using namespace std;
int main(){

    ios::sync_with_stdio(false);//输入优化
    cin.tie(0);cout.tie(0);

    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    for(int i=1;i<=m;i++){//差分数组
        cin>>c[i]>>l[i]>>r[i];
        cf[l[i]]+=c[i];
        cf[r[i]+1]-=c[i];
    }
    int j=m;//从后往前推，只要发现删除哪个后正好符合要求，则其为ans
    for(int i=1;i<=n;i++){
        sum+=cf[i];//计算每个教室的使用情况
        if(sum>a[i]){//
            while(sum>a[i]){//从后往前撤回消息
                cf[l[j]]-=c[j];
                cf[r[j]+1]+=c[j];
```

```
                if(l[j]<=i&&i<=r[j])//PAT：如果一个请求包含了第i个教室，则删除它会影响sum的
值
                    sum-=c[j];
                j--;
            }
            if(flag)x=j,flag=0;//更新x
            else
                x=min(x,j);
        }
    }
    if(x==-1)cout<<"0";
    else cout<<"-1"<<endl<<x+1;
    return 0;
}
```

旅行家的预算

本题考查贪心，策略如下：假设加满油可以找到一个油价小于当前站点油价的加油站:那么就加油使可以达到该站点，不然说明可达范围内所有点油价都高于本身站点，分两种情况，包括终点：直接把油加到可以到达终点；不包括终点时油加满后到达后面站点中油价最小的那个

```
#include<bits/stdc++.h>
using namespace std;
struct point{
    // l 表示加油站到起点所需的油量
    // v 表示该加油站的油价
    double l, v;
}way[10];
bool cmp(point p1, point p2){
    return p1.l < p2.l;
}
int main(){
    double D1, C, D2, P, sum = 0;
    int N;
    bool flag = 1;
    cin >> D1 >> C >> D2 >> P >> N;
    for(int i = 1; i <= N ; i++) {
        cin >> way[i].l >> way[i].v;
        way[i].l /= D2;
    }
    sort(way+1, way+N+1, cmp);

    way[0].l = 0;
    way[0].v = P;
    way[N+1].l = D1/D2;
    way[N+1].v = 0;

    int p = 0;
    //表示油箱里剩的油
```

```
    double last = 0;
    while(p < N+1){
        int des = -1;
        int minn = -1;
        for(int i = p + 1; i <= N + 1; i++){
            if(way[i].l - way[p].l > C) break;
            if(way[i].v < way[p].v){
                des = i;
                break;
            }
            else {
                if(minn == -1)minn = i;
                else {
                    if(way[i].v < way[minn].v)minn = i;
                }
            }
        }
        if(des != -1){
            // 表示发现了比自身油价低的加油站
            double cost = way[des].l - way[p].l;
            if(last >= cost) last -= cost;
            else {
                sum += (way[des].l - way[p].l - last) * way[p].v;
                last = 0;
            }
            p = des;
        }
        else {
            if(minn == -1){
                flag = 0;
                break;
            }
            else{
                // 表示没有找到比自身油价低的加油站
                double cost = way[minn].l - way[p].l;
                if(way[N+1].l - way[p].l < C){
                    sum += (way[N+1].l - way[p].l - last) * way[p].v;
                    p = N+1;
                }
                else {
                    sum += (C - last) * way[p].v;
                    last = C - (way[minn].l - way[p].l);
                    p = minn;
                }
            }
        }
    }
    if(flag) printf("%.2lf",sum);
    else cout << "No Solution";
    return 0;
}
```

细胞分裂

本题考查数论，题目即要求最小的正整数k使得有一个$i, 1-n$，满足$m_1^{m_2}|s_i^k$，首先对m_1进行质因数分解，可以证明s_i必须整除m_1，后面考察两者的质因数分解形式，维护k的最小值即可

```
#include<bits/stdc++.h>
#define _for(i,a,b) for(int i=a; i<b; i++)
using namespace std;

int n, m1, m2, S[10010], com=1;
int fac_m[20], degree_m[20], cnt;//分别记录m质因子、次数和总个数 m = m1^m2
int degree_s[20];
const int N = 44722;
int pri[N], len_pri = 0;
bool is_pri[N];
void Euler(int n){
    memset(is_pri+2, 1, sizeof(is_pri)-2);
    for(int i=2; i<=n; i++){
        if(is_pri[i]) pri[len_pri++]=i;
        for(int j=0; j<len_pri && i*pri[j]<=n; j++){
            is_pri[i*pri[j]] = 0;
            if(i%pri[j] == 0) break;
        }
    }
}

//将x分解成质因子相乘的形式
int fac(int x){
    int tmp = 1;
    _for(i,0,cnt){
        degree_s[i] = 0;
        while(x%fac_m[i] == 0){
            x /= fac_m[i];
            degree_s[i]++;
        }
        tmp = max(tmp, (int)ceil(degree_m[i]*1.0/degree_s[i]));
    }
    return tmp;
}

void inti(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m1 >> m2;
    _for(i,0,n) cin >> S[i];
    Euler(N);
}
```

```
int main(){
    inti();
    int ans = -1;
    if (m1==1)cout << 0;
    else {
        // 分解m1
        int sqrtm = (int)sqrt(m1);
        for(int i=0; pri[i]<sqrtm; i++){
            if(m1%pri[i] == 0){
                fac_m[cnt] = degree_m[cnt] = 0;
                fac_m[cnt++] = pri[i];
                while(m1%pri[i] == 0){
                    m1 /= pri[i];
                    degree_m[cnt-1]++;
                }
            }
        }
        if(m1!=1){
            fac_m[cnt++] = m1;
            degree_m[cnt-1]++;
        }
        _for(i,0,cnt){
            degree_m[i] *= m2;
            com *= fac_m[i];
        }

        _for(i,0,n){
            if(S[i]%com == 0){
                int tmp = fac(S[i]);
                if(ans == -1)ans = tmp;
                else ans = min(tmp,ans);
            }
        }
        cout << ans;
    }
    return 0;
}
```