总结

11月12日

$T1\ desire$:

本蒟蒻并不会，这里不讲正解。讲讲错误思路~~水字数~~，考场上什么思路都没有，但看这个式子很像$Haffman$，~~可惜我不会~~，于是怎么办哪，摆烂。

$T2\ dealing$：

考场上有很多想法，比如并查集，这个事实上是对的，然而只是暴力的去做的话会TLE，于是想办法优化，就有了奇怪的线段树做法~~直接寄掉~~，由于想不出其他做法，最后就只打了暴力。

正解：

考虑倍增优化，说起来很离谱，这个倍增优化和普通的并查集没有太大区别，只是多了一步从高位向低位更新的过程。具体来说，在建的过程中，以$fa[i][j]$表示以i为起点，向后拓展2^j个的节点的父亲。和普通并查集没有区别，就是找父亲，更新，只是一次性跳的点为2^j个。考虑下放，显然对于$fa[i][j]$想要更新到$fa[i][j-1]$需要考率i和$i+2^{j-1}$，先找到$fa[i][j]$的最高父亲，设它为u，那么$i+2^{i-1}$的父亲显然为$u+2^{j-1}$。

代码：

```
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
int power(int a,int b){
    int res=1;
    for(;b;b>>=1){
        if(b&1)
            res=1ll*res*a%mod;
        a=1ll*a*a%mod;
    }
    return res;
}
int n,m;
int fa[2000086][25];
int Find(int x,int pos){
    return fa[x][pos]==x?x:fa[x][pos]=Find(fa[x][pos],pos);
}
void con(int x,int y,int pos){
    int u=Find(x,pos);
    int v=Find(y,pos);
    if(u!=v)
        fa[v][pos]=u;
}
void upd(){
    for(int i=20;i>=1;i--){
        for(int j=1;j<=n;j++){
            int u=Find(j,i);
            con(u,j,i-1);
            con(u+(1<<(i-1)),j+(1<<(i-1)),i-1);
        }
```

```
    }
}
void work(int x,int y,int len){
    for(int i=20;i>=0;i--){
        if((len>>i)&1){
            con(x,y,i);
            x+=(1<<i);
            y+=(1<<i);
        }
    }
}
bool vis[2000086];
int main(){
    freopen("dealing.in","r",stdin);
    freopen("dealing.out","w",stdout);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        for(int j=0;j<=20;j++)
            fa[i][j]=i;
    }
    for(int i=1;i<=m;i++){
        int x,y,len;
        scanf("%d%d%d",&len,&x,&y);
        if(x==y)continue;
        work(x,y,len);
    }
    upd();
    int ans=0;
    for(int i=1;i<=n;i++){
        int x=Find(i,0);
        if(!vis[x])
            ans++,vis[x]=true;
    }
    printf("%d",power(26,ans));
    return 0;
}
```

$T3\ lunatic$：

考场没有做这道题~~本蒟蒻在前两道题浪费了太多时间~~，读题后没读懂，就放掉了。其实相比前两道，这道题更具有可骗分性。

正解：

考虑如果一个线段包含另一个线段，那么这条线段就不会做出真正有用的贡献。因为当他和他所包含的线段在一起时，一定是以他的子线段为基准。考虑将和其他线段有包含关系的划分为集合A，其他线段为集合B。于是考虑dp,令$dp[i][j]$表示前i条个集合含了j条线段。显然当B集合最后的线段被含进去时，答案为$dp[i][n]$加上A集合内长度前$k-i$长的线段。考虑转移$dp[i][j]=max(dp[i-1][k]+r_{k+1})-l_i$，时间效率$O(n^2)$，考虑优化。对于所有的已确定的划分来说答案为$max(0,l_{i1}-r_{i2-1})+max(0,l_{i2}-ri3-1)+\ldots+max(0,l_{i_{m-1}}-r_{i_m})$。又由于答案的下界为前$k-1$长的线段长之和，当上式有一个小于零时，都不可能高于下界，所以可以去掉取最大值的过程。直接变为$l_{i1}-r_{i2-1}+\ldots l_{i_{m-1}}-r_{im}$，考虑以$l_i-r_{i-1}$为关键字排序，贪心就行。

代码：然而并没有打完。

$T4$ $season$：

正解不会，只会暴力。考虑线段树优化，开n个线段树，空间开不下，但能过$60pts$，然后最小生成树。

总结：考场上基本上是寄掉了，原因挺多：

1.对自己认识不清楚，老想打正解，结果暴力都没打完。

2.代码实现能力太差，有思路会打错。

3.不注意细节，会死在一些奇怪的地方，比如没删调试。

11月14日总结：

$T1$ $substr$：

考场上想了一个奇怪的贪心思路，希望能以每个字符串之间的重合度为关键字来贪心。事实上会被卡掉~~但好像水货数据可以过~~$80pts$。

正解：

考虑正确的搜索，我们对每个字符串进行状态压缩，建一棵$Trie$树，将最后一个字符状态定义为2^{i-1}，如果一个字符串包含了它，就说明状态包含它，最终状态为$(1<<n)-1$。先用ac自动机搞定匹配问题，然后拓展当前状态。

代码：

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<queue>
using namespace std;
int n;
char s[58];
int tr[686][28],tot;
int sta[686];
void Tire(int x){
    int now=0;
    int len=strlen(s+1);
    for(int i=1;i<=len;i++){
        if(!tr[now][s[i]-'A'])
            tr[now][s[i]-'A']=++tot;
        now=tr[now][s[i]-'A'];
    }
    sta[now]|=(1<<(x-1));
}
int fail[686];
void Fail(){
    queue<int>q;
    for(int i=0;i<26;i++){
        if(tr[0][i])
            q.push(tr[0][i]);
    }
    while(!q.empty()){
        int x=q.front();
        q.pop();
        for(int i=0;i<26;i++){
```

```
            if(tr[x][i]){
                fail[tr[x][i]]=tr[fail[x]][i];
                sta[tr[x][i]]|=sta[fail[tr[x][i]]];
                q.push(tr[x][i]);
            }
            else
                tr[x][i]=tr[fail[x]][i];
        }
    }
}
bool vis[686][5086];
int fa[3000086],ans[3000086],cnt;
char c[686];
void bfs(){
    tot=0;
    queue< pair< pair<int,int> ,int> >q;
    vis[0][0]=1;
    q.push(make_pair(make_pair(0,0),0));
    while(!q.empty()){
        int now=q.front().first.first,st=q.front().first.second;
        int x=q.front().second;
        q.pop();
        if(st==(1<<n)-1){
            while(x){
                c[++cnt]=ans[x]+'A';
                x=fa[x];
            }
            for(int i=cnt;i>=1;i--)
                putchar(c[i]);
            return;
        }
        for(int i=0;i<26;i++){
            if(!vis[tr[now][i]][st|sta[tr[now][i]]]){
                vis[tr[now][i]][st|sta[tr[now][i]]]=true;
                ans[++tot]=i;
                fa[tot]=x;
                q.push(make_pair(make_pair(tr[now][i],st|sta[tr[now][i]]),tot));
            }
        }
    }
}
int main(){
    freopen("substr.in","r",stdin);
    freopen("substr.out","w",stdout);
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%s",s+1);
        Tire(i);
    }
    Fail();
    bfs();
    return 0;
}
```

$T2\ flower$：

考场上敲了一个暴力，然后对于一些特殊点专门敲了一个线段树，骗了$40pts$。

正解：

在k确定的情况的下，在区间$a\times k$到$(a+1)\times k-1$之间，他们的最大值显然为最优解。可以考虑分块，由于区间只有$kink$个，取一个块的大小为$\sqrt{kink}$，大概是1000，对于每一块处理在取询问为k时的答案即可，最后在处理答案是只处理l到r中间的块，其余暴力处理。

代码：

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<cstring>
using namespace std;
int re(){
    int res=0,p=1;
    char c=getchar();
    while(c<'0'||c>'9'){
        if(c=='-')p=-1;
        c=getchar();
    }
    while(c>='0'&&c<='9'){
        res=(res<<3)+(res<<1)+(c^48);
        c=getchar();
    }
    return res*p;
}
void wr(int x){
    if(x<0)putchar('-'),x=-x;
    if(x>9)wr(x/10);
    putchar(x%10+'0');
}
const int siz=1000;
const int K=100000;
int n,m;
int a[100086];
int val[100086];
int block;
int ans[108][100008];
int main(){
    freopen("flower.in","r",stdin);
    freopen("flower.out","w",stdout);
    n=re();m=re();
    for(int i=1;i<=n;i++)
        a[i]=re();
    block=(n-1)/siz+1;
    for(int i=1;i<=block;i++){
        memset(val,0,sizeof val);
        for(int j=siz*(i-1)+1;j<=n&&j<=i*siz;j++)
            val[a[j]]=a[j];
        for(int j=1;j<=K;j++)
            val[j]=max(val[j],val[j-1]);
        for(int j=1;j<=K;j++)
            for(int k=0;k<=K;k+=j)
```

```
                ans[i][j]=max(ans[i][j],val[min(K,k+j-1)]-k);
    }
    for(int i=1;i<=m;i++){
        int l,r,k;
        l=re();r=re();k=re();
        int res=0;
        int st=(l-2)/siz+2,ed=r/siz;
        if(ed<st){
            for(int j=l;j<=r;j++)
                res=max(res,a[j]%k);
            wr(res);putchar('\n');
            continue;
        }
        for(int j=st;j<=ed;j++)
            res=max(res,ans[j][k]);
        for(int j=l;j<=(st-1)*siz&&j<=n;j++)
            res=max(res,a[j]%k);
        for(int j=(ed-1)*siz+1;j<=r&&j<=n;j++)
            res=max(res,a[j]%k);
        wr(res);putchar('\n');
    }
    return 0;
}
```

$T3\ refract$：

挺好的一道dp题，考场上看都没看，全卷前两道题了。

正解：考虑按照 x_i 排序转移, 并记录 $dp[i][0/1]$ 表示以第 i 个点为顶端接下来向

左或向右的折线方案数. 从左到右加点, 考虑前 i 个点构成的包含 i 点的折

线. 由于新点横坐标最大, 所以只可能在折线的第一位或第二位:

1.任意的$y_j \leq y_i$ 来说，$dp[i][0]+=dp[j][1]$

2.任意的$y_i \leq y_j$来说，$dp[j][1]+=dp[i][0]$.

代码：

```
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
int n;
pair<int,int>pos[6086];
long long dp[6086][3];
int main(){
    freopen("refract.in","r",stdin);
    freopen("refract.out","w",stdout);
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&pos[i].first,&pos[i].second);
    sort(pos+1,pos+n+1);
    for(int i=1;i<=n;i++){
        dp[i][0]=dp[i][1]=1;
        for(int j=i-1;j>=1;j--){
            if(pos[j].second<pos[i].second)
                dp[i][0]=(1ll*dp[i][0]+1ll*dp[j][1])%mod;
            else
```

```
                dp[j][1]=(1ll*dp[j][1]+1ll*dp[i][0])%mod;
        }
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        ans=(1ll*ans+1ll*dp[i][0]+1ll*dp[i][1]-1)%mod;
    }
    printf("%d\n",ans);
    return 0;
}
```

T4 paint:

考场没做，事实上这道题大概是今天最简单的一道题。

正解：

易证，当前操作所包含的连通块一定是上一次操作的子连通块。有了这个结论就很好做了，考虑枚举最后一次修改的地方，那么最高修改次数一定是最远的1所在的地方，这里的距离是指中间所经历的不同01串。

代码：

```
#include<bits/stdc++.h>
using namespace std;
int n,m;
int dir1[6]={0,1,0,-1},dir2[6]={1,0,-1,0};
char geo[58][58];
int dis[58][58];
int bfs(int s,int t){
    priority_queue< pair<int, pair<int,int> > >q;
    memset(dis,-1,sizeof dis);
    dis[s][t]=0;
    q.push(make_pair(0,make_pair(s,t)));
    int res=0;
    while(!q.empty()){
        int x=q.top().second.first,y=q.top().second.second;
        q.pop();
        if(geo[x][y]=='1')
            res=max(res,dis[x][y]);
        for(int i=0;i<4;i++){
            int tx=x+dir1[i],ty=y+dir2[i];
            if(tx<1||tx>n||ty<1||ty>m)continue;
            if(dis[tx][ty]==-1){
                if(geo[x][y]!=geo[tx][ty])
                    dis[tx][ty]=dis[x][y]+1;
                else
                    dis[tx][ty]=dis[x][y];
                q.push(make_pair(-dis[tx][ty],make_pair(tx,ty)));
            }
        }
    }
    return res;
}
int ans=1e9;
int main(){
```

```
    freopen("paint.in","r",stdin);
    freopen("paint.out","w",stdout);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        scanf("%s",geo[i]+1);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            ans=min(ans,bfs(i,j));
        }
    }
    printf("%d\n",ans+1);
    return 0;
}
```