qnmbql的11.14日考试总结

T1 substr

题目描述

给定 n 个字符串 S1,S2,…,Sn ，要求找到一个最短的字符串 T，使得这 n 个字符串都是 T 的子串。

考场思路：大方向搜索+剪枝，预处理出所有子串两两之间首尾相接所能剩下的位数。同时处理出被其他字串所包含的子串，搜索时直接忽略，能省下大量时间。搜索过程中判断当前构造出的位数已经超过了已经搜出来的ans，直接退出。因为考场上忘记搜索完字典序的比较，WA了$20tps$,只T了一个点。

正解：状压DP

考场代码：

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
using namespace std;
int n,same[15][15];
string s[15];
int ans[15],ansnum=99999999;
int path[15],alre[15];
bool use[15],cnt;
void dfs(int pos,int num,int fa){
    if(pos>n-cnt){
        if(num<ansnum){
            for(int i=1;i<=n;i++){
                ans[i]=path[i];
            }
            ansnum=num;
        }
        if(num==ansnum){
            for(int i=1;i<=n;i++){
                if(ans[i]<path[i]){
                    break;
                }
                else{
                    ans[i]=path[i];
                }
            }
        }
        return ;
    }
    if(num>=ansnum) return ;
    for(int i=1;i<=n;i++){
        if(alre[i] || use[i]) continue;
        alre[i]=1; path[pos]=i;
```

```
        dfs(pos+1,num+(s[i].length()-same[fa][i]),i);
        alre[i]=0;
    }
}
int main(){
    freopen("substr.in","r",stdin);
    freopen("substr.out","w",stdout);
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>s[i];
    }
    sort(s+1,s+n+1);
    //for(int i=1;i<=n;i++) cout<<s[i]<<'\n';
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(i==j) continue;
            int len1=s[i].length(),len2=s[j].length();
            //cout<<len1<<" "<<len2<<'\n';
            int up=min(len1,len2),tot=0;
            for(int k=up;k>=1;k--){
                bool cg=1;
                for(int l=1;l<=k;l++){
                    if(s[i][len1-l]!=s[j][k-l]){
                        cg=0; break;
                    }
                }
                if(cg){
                    tot=k; break;
                }
            }
            if(len1==tot){
                use[i]=1; cnt++;
            }else{
                if(len2==tot){
                    use[j]=1; cnt++;
                }
            }
            same[i][j]=tot;
        }
    }
    dfs(1,0,0);
    cout<<s[ans[1]];
    for(int i=2;i<=n;i++){
        if(use[ans[i]]) continue;
        int len=s[ans[i]].length();
        for(int j=same[ans[i-1]][ans[i]];j<len;j++){
            cout<<s[ans[i]][j];
        }

    }
    return 0;
}
```

T2 flower

题目：

题目描述

小 C 在家种了 n 盆花，每盆花有一个艳丽度 ai.
在接下来的 m 天中，每天早晨他会从一段编号连续的花中选择一盆摆放在客厅，并在晚上放回. 同时每天有特定的光照强度 ki，如果这一天里摆放在客厅的花艳丽度为 x，则他能获得的喜悦度为 x mod ki.
他希望知道，每一天他能获得的最大喜悦度是多少.

考场思路：妥妥的跑了一波暴力后考虑分块优化，但是面对变化的k就束手无策了。

正解：考虑当 k 确定的时候如何求答案, 显然对于所有形如 $[ak,(a+1)k)$的值域区间, 最大值一定是最优的.进一步观察发现, 这样的区间总个数只有$klnk$ 个. 考虑分块, 那么我们可以在$O(n+klnk)$的时间复杂度内处理出一个块对于任意 k 的答案. 询问时复杂度是$O(mS)$的, 取$S=\sqrt{klnk}$ 可以达到最优复杂度 $O(n\sqrt{klnk})$ 难度在于具体实现

代码：

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN=100000;
int a[MAXN+5];
int n,m;
int ans[105][MAXN+5];
int tong[MAXN+5],base=1000;
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
//  freopen("flower2.in","r",stdin);
//  freopen("flower.out","w",stdout);
    cin>>n>>m;
    int num=(n-1)/base+1;
    for(int i=0;i<n;i++) cin>>a[i];
    for(int i=0;i<num;i++){
        int l=i*base,r=min(n,(i+1)*base);
        memset(tong,0,sizeof(tong));
        for(int j=l;j<r;j++) tong[a[j]]=a[j];
        for(int j=1;j<=MAXN;j++) tong[j]=max(tong[j-1],tong[j]);
        for(int k=1;k<=MAXN;k++){
            for(int a=0;a<=MAXN;a+=k){
                int up=min(MAXN,a+k-1);
                ans[i][k]=max(ans[i][k],tong[up]%k);
            }
        }
    }
    while(m--){
        int l,r,k;
```

```
        cin>>l>>r>>k;
        l--; r--;
        int maxx=0;
        int x=(l)/base+1,y=(r)/base;
        if(x>y){
            for(int i=l;i<=r;i++) {
                maxx=max(maxx,a[i]%k);
            }
        }else{
            for(int i=x;i<y;i++) maxx=max(maxx,ans[i][k]);
            for(int i=x*base-1;i>=l;i--) maxx=max(maxx,a[i]%k);
            for(int i=((y)*base);i<=r;i++) maxx=max(maxx,a[i]%k);
        }

        cout<<maxx<<'\n';
    }
    return 0;
}
```

T3 refract

题目：

题目描述

小 Y 十分喜爱光学相关的问题，一天他正在研究折射.
他在平面上放置了 n 个折射装置，希望利用这些装置画出美丽的折线. 折线将从某个装置出发，并且在经过一处装置时可以转向，若经过的装置坐标依次为 (x1, y1), (x2, y2), . . . (xk, yk)，则必须满足:
∀j ∈ (1, k], y[j]< y[j−1]
∀j ∈ (2, k], x[j−2] < x[j]< x[j−1] or x[j−1] < x[j] < x[j−2]
现在他希望你能告诉他，一共有多少种不同的光线能被他画出来，两种光线不同当且仅当经过的折射装置的集合不同. 你只需要告诉他答案对 10^9 + 7 取模后的结果.

考场思路：这只能是dp ，但是我一开始按照的y坐标排序，这导致转移的点的x坐标有上下两个界限，难以维护，最后没能实现代码。

正解：考虑按照x_i排序转移, 并记录$f(i,0/1)$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. $\forall y_j < y_i, f(i,0) \leftarrow f(j,1)$
2. $\forall y_j > y_i, f(j,1) \leftarrow f(k,0) | x_k > x_j and y_k < y_i$

   第二种情况可以前缀和优化, 复杂度 $O(n^2)$

代码

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#define ll long long
using namespace std;
const int MAXN=6005;
```

```cpp
const int p=1000000007;
int n;
struct things{
    int x,y;
}a[MAXN];
bool cmp(things a,things b){
    return a.x<b.x;
}
ll dp[MAXN][2],ans;
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    freopen("refract.in","r",stdin);
    freopen("refract.out","w",stdout);
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i].x>>a[i].y;
    sort(a+1,a+n+1,cmp);
    for(int i=1;i<=n;i++){
        dp[i][0]=dp[i][1]=1;
        for(int j=i-1;j>=1;j--){
            if(a[j].y>a[i].y){
                dp[j][1]+=dp[i][0]; dp[j][1]%=p;
            }
            if(a[j].y<a[i].y){
                dp[i][0]+=dp[j][1]; dp[i][0]%=p;
            }
        }
    }
    for(int i=1;i<=n;i++){
        ans=(ans+dp[i][0]+dp[i][1])%p;
    }
    cout<<(ans-n)%p;
    return 0;
}
```

T4 paint

题目：

题目描述

小 G 的喜欢作画，尤其喜欢仅使用黑白两色作画.
画作可以抽象成一个 r × c 大小的 01 矩阵. 现在小 G 构思好了了他的画作，准备动笔开始作画. 初始时画布是全白的，他每一次下笔可以将一个四联通的部分涂成黑色或白色.
你需要告诉他，在给出的构思下，他最少需要下笔多少次才能完成画作.

考场思路：死磕T2中，考场无思路

正解：不难证明猜到一个这样的结论: 存在一种最优方案使得每次操作的区域是上一次的子集且颜色与上一次相反. 考虑归纳证明, 记 S 为当前所有操作区域的并, T 为接下来一步的操作区域, 我们有:

\1. T 与 S 有交的情况一定可以转化成 T 被 S 包含的情况.

\2. T 与 S 交集为空时, 可以找一个连接 S 和 T 的集合 M 并操作 $S \cup T \cup M$ 并将之前的所有操作连接到更外的层以及外层的连接部分同时操作, 特殊处理最外层和第二层的情况.

\3. T 被 S 包含时, T 落在某个完整区域内时等价于情况二, 否则一定连接若干个同色块, 这些块可以同时处理, 步数一定不会更劣。

知道这个结论就比较好做了, 我们可以枚举最后被修改的区域, 这时答案就是将同色边边权当作 0, 异色边边权当作 1 后距离这个点最远的黑色点的距离（确保连到的是第一次操作的点）, 对所有　　点取最小值即可.（最后要加1！！（我们扩展的点本身自己翻转的步数））

代码：

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN=55*55;
int r,c;
bool flag[55][55];
int dx[5]={-1,1,0,0},dy[5]={0,0,-1,1};
int head[MAXN],tote;
struct edge{
    int u,v,nex,w;
}e[2*MAXN];
void add(int u,int v,int w){
    tote++;
    e[tote].u=u; e[tote].v=v; e[tote].w=w;
    e[tote].nex=head[u]; head[u]=tote;
}
#define num(i,j) ((i)*max(r,c)+(j))
int dis[MAXN+10][MAXN+10];
deque<int> q;
void bfs(int s){
//  memset(vis,0,sizeof(vis));
    for(int i=0;i<=MAXN;i++) dis[s][i]=-1;
    dis[s][s]=0;
    q.push_back(s);
    while(!q.empty()){
        int x=q.front();
        q.pop_front();

        for(int i=head[x];i;i=e[i].nex){
            int v=e[i].v;
            if(dis[s][v]!=-1) continue;
            if(!e[i].w) q.push_front(v);
            else q.push_back(v);
            dis[s][v]=dis[s][x]+e[i].w;
        }
    }

}
int minn=999999999;
int goal[MAXN],cnt;
```

```cpp
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    freopen("paint.in","r",stdin);
    freopen("paint.out","w",stdout);
    cin>>r>>c;

    for(int i=1;i<=r;i++){
        char s[60]; cin>>s;
        for(int j=1;j<=c;j++){
            flag[i][j]=(s[j-1]-'0');
            if(flag[i][j]){
                goal[++cnt]=num(i,j);
//              cout<<i<<" "<<j<<" "<<goal[cnt]<<'\n';
            }
        }
    }
//  for(int i=1;i<=cnt;i++) cout<<goal[i]<<" ";
    for(int i=1;i<=r;i++){
        for(int j=1;j<=c;j++){
            int di,dj;
            for(int k=0;k<4;k++){
                di=i+dx[k],dj=j+dy[k];
                if(0<di && di<=r && 0<dj && dj<=c){
                    if(flag[i][j]^flag[di][dj]){
                        add(num(i,j),num(di,dj),1);
                        add(num(di,dj),num(i,j),1);
                    }
                    else{
                        add(num(i,j),num(di,dj),0);
                        add(num(di,dj),num(i,j),0);
                    }
                }
            }
        }
    }

    for(int i=1;i<=r;i++){
        for(int j=1;j<=c;j++){
            int s=num(i,j);
            //cout<<"++";
            bfs(s);
            int maxx=0;
            for(int k=1;k<=cnt;k++){
                if(goal[k]==s) continue;
                maxx=max(maxx,dis[s][goal[k]]);
//              cout<<i<<" "<<j<<" "<<goal[k]<<" "<<dis[s][goal[k]]<<'\n';
            }
            minn=min(minn,maxx);
        }
    }
    cout<<minn+1;
    return 0;
}
```

总结

优秀的暴力在考场上是十分重要的！其次是问题的转化，如何建模并转化为易于解决的模型，有助于自己思路接近正解。还有就是DP状态的灵活变更转化，才能更快的推出正确的方程。大胆观察猜测结论，有助于思路的形成。