2026年2月7日SACCC五级题解

T1 紧挨(close)

考查知识点：枚举。

题目大意：在字符串S中寻找最近的“SACCC”子序列，输出其首尾所在位置差+1。

sol1 30pts

数据满足输入的字符串长度不超过 20，可以采用 0/1 枚举或者二进制枚举，如果枚举得到的子序列就是 “SACCC”，查看首位字符位置，刷新答案，时间复杂度为 $O(2^n)$。

sol2 100pts

剩余数据满足长度不超过 1000，由于 “SACCC” 的的长度只有 5，可以顺序遍历，碰到 ‘S’ 字符则开始寻找最接近的第一组 “SACCC” 即可，刷新答案，时间复杂度为 $O(n^2)$。

如果将题目数据规模继续扩大，还可以分别寻找 ‘S'、’A‘、’C ’的位置按照升序的顺序放入三个 vactor（a,b,c），枚举 a 中 ‘S’ 的位置 p，然后根据 p 分别到 b 和 c 中二分查找最近的 ‘A' 和最近的 3 个 ’C' ，刷新答案，时间复杂度为 $O(n*log^2n)$。

满分程序如下:

```cpp
#include <bits/stdc++.h>
using namespace std;
string s;
string t = "SACCC";
int main()
{
    freopen("close.in","r",stdin);
    freopen("close.out","w",stdout);
    cin >> s;
    int ans = s.size();
    for (int i = 0; i < s.size(); i++)
    {
        if (s[i] != 'S')
            continue;
        int now = 1; // 下一个要查询的字符为 t[now]
        for (int j = i + 1; j < s.size(); j++)
        {
            // 匹配上最近的一个
            if (s[j] == t[now])
                now++;
            if (now == 5)
            {
                // 到 j 的位置时五个字符都找到了
                // 即 s[i] ~ s[j] 这个子串中存在子序列 SACCC
                // 显然这是 i 开头最短的子串
                ans = min(ans, j - i + 1);
                break;
            }
        }
    }
    cout << ans << "\n";
    return 0;
}
```

T2 家庭(family)

考查知识点：模拟，闰年判断，日期计算，简单数论，前缀和。

题目大意：共n组数据，每组数据要求判断输入的三个日期是否合法，若不合法就输出“FALSE”。否则就输出由三人出生日期算出的 mod1，mod2之间所有合数之和。

sol1 40pts

输入判断完 “FALSE” 之后算出每一个人的 age 和 sec，然后求出 mod1 和 mod2。然后遍历 mod1，mod2 中每一个数，其中**对每一个数判断是否是合数**，代码如下：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int dpm[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};//每一个月的总天数
int y[5], m[5], d[5], MOD, sec[5];
//判断闰年函数
bool run(int x){return x % 400 == 0 || (x % 100 && x % 4 == 0);}
//函数getd表示从1899年12月31日（或者说1900年0月0日更好理解）到这一天的天数
int getd(int yr, int mo, int dy){
    int s = 0;
    for(int i = 1900; i < yr; i++) s += 365 + run(i);
    for(int i = 1; i < mo; i++) s += dpm[i] + (i == 2 && run(yr));
    return s + dy;
}
//判断合数函数
bool heshu(int n){
    for(int i = 2; i < n; i++){
        if(n % i == 0) return true;
    }
    return false;
}
void work(){
    //输入
    bool f = 1;
    for(int i = 1; i <= 3; i++){
        cin >> y[i] >> m[i] >> d[i];
        if(m[i] == 2 && f){
            if(d[i] > run(y[i]) + 28){
                cout << "FALSE" << endl; f = false;
            }
        }else{
            if(d[i] > dpm[m[i]] && f){
                cout << "FALSE" << endl; f = false;
            }
        }
    }
    cin >> MOD;
    //一定要等输入完再考虑返回主程序，因为是多组数据，每组数据必须全部读入完毕才能读入下一组数据。
    if(!f) return;
```

```
    //计算每一个人的sec和mod1，2
    for(int i = 1; i <= 3; i++) sec[i] = (getd(2026, 2, 7) - getd(y[i], m[i], d[i])) * 24 * 3600;
    int mod1 = ((sec[1] % MOD) * (sec[2] % MOD)) % MOD;//防止越界，简单数论的应用
    int mod2 = sec[3] % MOD;
    //mod1，2之间的合数之和
    int ans = 0;
    for(int i = min(mod1, mod2); i <= max(mod1, mod2); i++){
        if(heshu(i)){
            ans += i;
        }
    }
    cout << ans << "\n";
}
signed main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    freopen("family.in", "r", stdin);
    freopen("family.out", "w", stdout);
    int t; cin >> t;
    while(t--) work();
    return 0;
}
```

可以看到这个方法连样例都会 TLE，因为判断合数的效率太低了，所以需要改进算法。

sol2 70pts

仔细观察易得时间复杂度主要大在最后计算 mod1，mod2之间合数之和的时候。因此，我们可以使用**素数筛的方式预处理**来减小判断合数的时间复杂度，代码如下：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int dpm[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
int y[5],m[5],d[5],MOD,sec[5];
const int MAXMOD=1e7+5;
bool flag[MAXMOD];
bool run(int x){return x%400==0||(x%100&&x%4==0);}
//埃氏筛法求素数
void sieve(int n){
    flag[1] = false;
    for(int i = 2; i <= n; i++){
        if(!flag[i]){
            for(int j = i + i; j <= n; j += i){
                flag[j] = true;
            }
        }
    }
}
int getd(int yr,int mo,int dy){
    int s=0;
    for(int i=1900;i<yr;i++)s+=365+run(i);
    for(int i=1;i<mo;i++)s+=dpm[i]+(i==2&&run(yr));
    return s+dy;
}
void work(){
    bool f=1;
    for(int i=1;i<=3;i++){
        cin>>y[i]>>m[i]>>d[i];
        if(m[i]==2&&f){
            if(d[i]>run(y[i])+28){
                cout<<"FALSE"<<endl;f=false;//日期非法
            }
        }else{
            if(d[i]>dpm[m[i]]&&f){
                cout<<"FALSE"<<endl;f=false;//日期非法
            }
        }
    }
```

```
    cin>>MOD;
    if(!f)return;
    for(int i=1;i<=3;i++) sec[i]=(getd(2026,2,7)-getd(y[i],m[i],d[i]))*24*3600;
    int mod1=((sec[1]%MOD)*(sec[2]%MOD))%MOD;
    int mod2=sec[3]%MOD;
    int ans=0;
    for(int i=min(mod1,mod2);i<=max(mod1,mod2);i++){
        if(flag[i]){
            ans+=i;                 //判断合数可以直接根据flag数组读取
        }
    }
    cout<<ans<"\n";
}
signed main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    freopen("family.in","r",stdin);
    freopen("family.out","w",stdout);
    sieve(10000000);//通过筛法得到每个数字是否为合数
    int t;cin>>t;
    while(t--)work();
    return 0;
}
```

修改之后已经能在一秒内跑完样例了，但依旧还有数据点超时，还需要继续改进算法。

sol3 100pts

仔细观察易得程序执行的大量时间主要是消耗在最后计算 mod1，mod2之间合数之和的时候。因此，我们可以使用前缀和的方式预处理来减小判断合数的时间复杂度，小细节：前缀和求差时的下标访问要保证在非负数。代码如下：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int dpm[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
int y[5],m[5],d[5],MOD,sec[5];
const int MAXMOD=1e7+5;
bool flag[MAXMOD];
int sum[MAXMOD];
bool run(int x){return x%400==0||(x%100&&x%4==0);}
void sieve(int n){
    flag[1]=false;
    for(int i=2;i<=n;i++){
        if(!flag[i]){
            for(int j=i+i;j<=n;j+=i){
                flag[j]=true;
            }
        }
    }//埃氏筛法求素数
    for(int i=1;i<=n;i++) sum[i]=sum[i-1]+(flag[i]?i:0);//求前缀和
}
int getd(int yr,int mo,int dy){
    int s=0;
    for(int i=1900;i<yr;i++)s+=365+run(i);
    for(int i=1;i<mo;i++)s+=dpm[i]+(i==2&&run(yr));
    return s+dy;
}
void work(){
    bool f=1;
    for(int i=1;i<=3;i++){
        cin>>y[i]>>m[i]>>d[i];
        if(m[i]==2&&f){
            if(d[i]>run(y[i])+28){
                cout<<"FALSE"<<endl;f=false;
            }
        }else{
            if(d[i]>dpm[m[i]]&&f){
                cout<<"FALSE"<<endl;f=false;
            }
        }
```

```
    }
    cin>>MOD;
    if(!f)return;
    for(int i=1;i<=3;i++) sec[i]=(getd(2026,2,7)-getd(y[i],m[i],d[i]))*24*3600;
    int p=((sec[1]%MOD)*(sec[2]%MOD))%MOD;
    int q=sec[3]%MOD;
    int l=min(p,q),r=max(p,q);
    if(!l) l++;        //如果l为0，为防止下面l-1为负数下标访问，又因为0不是合数，直接将l变成1，不影响答案
    cout<<sum[r]-sum[l-1]<<"\n";//求前缀差获得答案
}
signed main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    freopen("family.in","r",stdin);
    freopen("family.out","w",stdout);
    sieve(10000000);
    int t;cin>>t;
    while(t--)work();
    return 0;
}
```

T3 选材(select)

题目大意：将最多36个个体分成两半（数量尽量接近），使得两半的和值之差的绝对值最小。

sol1 70pts

考查知识点：直接暴搜/二进制枚举即可。

sol2 100pts

考查知识点：搜索，枚举，折半枚举，二分。

设sum为所有数字之和。

注意到当$n = 36$时，2^n太大了。

但是$2^{\lfloor \frac{n}{2} \rfloor}$是完全可以的。

在这个方法中；

原题是：选出$\lfloor \frac{n}{2} \rfloor$个，使选的数的和和没选的数的和的差的绝对值最小。
我们改成是：选出$\lfloor \frac{n}{2} \rfloor$或$\lfloor \frac{n+1}{2} \rfloor$个，且满足选的数字之和不大于没选的数字和（即选的数的和$\leq \frac{sum}{2}$）。

那么我们可以先暴搜前$\lfloor \frac{n}{2} \rfloor$位，一共最多$2^{18} \approx 2.6 \times 10^5$种可能性。

因为选出的人数也有要求，所以将前半部分的所有情况按照选的个数区分。区分出$0 \sim \lfloor \frac{n}{2} \rfloor$ 共$\lfloor \frac{n}{2} \rfloor + 1$个集合，其中第$x(0 \leq x \leq \lfloor \frac{n}{2} \rfloor)$个集合存储的是前半部分中选出$x$个可以达到的所有的数字和。

然后暴搜后$n - \lfloor \frac{n}{2} \rfloor$位，对于每种方法，设$s$表示这个后半部分的方法的选出的数字之和，然后分成选$\lfloor \frac{n}{2} \rfloor$个和选$\lfloor \frac{n+1}{2} \rfloor$个两种情况，分别求出前半部分还需要多少个，然后在对应的集合里面二分出最大的数，满足加上s后$\leq \frac{sum}{2}$，也就是说在满足与后半部分相加仍$\leq \frac{sum}{2}$的情况下前半部分的数字和最大可以是多少。（选的数字的和比没选的数字的和少的情况下，选的数字之和越大，选的数字之和和没选的数字之和的差越小）最后再用这个最大的数字与s相加，得到选出的数字和，再和sum做差求出没选的数字和，最后选出和没选出的做差得到这个后半部分方案的最小答案。

本题的答案就是所有后半部分方案的答案的最小值。

代码如下：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
long long a[40];
vector<long long> mp1[41];
long long x(long long a, long long b, long long s)
{
        return s - 2 * a - 2 * b;
}
signed main()
{
        freopen("select.in", "r", stdin);
        freopen("select.out", "w", stdout);
    int n;
    cin >> n;
    long long sum = 0;
    for (int i = 1; i <= n; i++)
    {
            cin >> a[i];
            sum += a[i];
    }
    int x1 = n / 2, x2 = n - x1;
    for (int i = 0; i < (1ll << x1); i++)
    {
            long long s = 0, c = 0;
            for (int j = 0; j < x1; j++)
            {
                    if (i & (1ll << j))
                    {
                            s += a[j + 1];
                            c++;
                    }
            }
            mp1[c].push_back(s);
    }
    for (int i = 0; i <= x1; i++)
    {
            sort(mp1[i].begin(), mp1[i].end());
    }
```

```
    long long ans = 1e18;
    for (int i = 0; i < (1ll << x2); i++)
    {
            long long s = 0, c = 0;
            for (int j = 0; j < x2; j++)
            {
                    if (i & (1ll << j))
                    {
                    c++;
                            s += a[n - j];
                    }
            }
        int p1 = n / 2 - c, p2 = (n + 1) / 2 - c;
        if (!mp1[p1].empty() && s + mp1[p1][0] <= sum / 2)
        {
                    auto t1 = upper_bound(mp1[p1].begin(), mp1[p1].end(), sum / 2 - s);
                    auto t2 = t1; t2--;
                ans = min(ans, x(*t2, s, sum));
            }
            if (!mp1[p2].empty() && s + mp1[p2][0] <= sum / 2)
        {
                ans = min(ans, x(*t1, s, sum));
                    auto t2 = t1; t2--;
                ans = min(ans, x(*t2, s, sum));
            }
    }
    cout << ans << endl;
        return 0;
}
```

sol3 100pts

考察知识点：搜索，枚举，折半枚举，二分。

设sum为所有数字之和。

注意到当$n = 36$时，2^n太大了。

但是$2^{\lfloor \frac{n}{2} \rfloor}$是完全可以的。

设$m = \left\lfloor \frac{n}{2} \right\rfloor$。

那么我们一共需要选出m个数，同时这个m也表示把这个序列分成两半后前半段的长度。

那么我们可以先暴搜前m位，一共最多$2^{18} \approx 2.6 \times 10^5$种可能性。

因为选出的人数也有要求，所以将前半部分的所有情况按照选的个数区分。区分出$0 \sim m$ 共$m+1$个集合，其中第$x(0 \leq x \leq m)$个集合存储的是前半部分中选出x个可以达到的所有的数字和。

然后暴搜后$n-m$位，对于每种方法，设s，c分别表示这个后半部分的方法的选出数字之和及数字个数。

- 如果$c > m$说明选的个数已经超过了题目要求，直接跳过；
- 如果$c = m$说明已经选满了$\left\lfloor \frac{n}{2} \right\rfloor$个，直接根据$s$和$sum$求出这个后半部分的方案的答案。
- 如果$c < m$说明前半部分还要选$m-c$个，而且前半部分的数字总和需要尽可能接近$\frac{sum}{2} - s$，这样加上s就更接近sum的一半。而接近$\frac{sum}{2} - s$分为比它大和比它小，所以再前半部分的集合$m-c$中二分出大于$\frac{sum}{2} - s$的最小值和小于$\frac{sum}{2} - s$的最大值，比较一下这两种方法算出的答案哪个较小，即为这个后半部分的方案的答案。

最后求出所有后半部分的方案的答案中最小的答案是哪个，将其输出即可。

代码如下：

```
#include<bits/stdc++.h>
using namespace std;
using LL = long long;
const int N = 40;
LL a[N];
int n;
vector<LL> v[N];
int main(){
        freopen("select.in","r",stdin);
        freopen("select.out","w",stdout);
        ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
        cin >> n;
        LL tot = 0;
        for(int i = 1; i <= n; i++){
                cin >> a[i];
                tot += a[i];
        }
        int m = n / 2;

        int U = 1 << m;
        for(int i = 1; i < U; i++){
                int t = 0;
                LL sum = 0;
                for(int j = 0; j < m; j++){
                        if((i >> j) & 1){
                                t++;
                                sum += a[j + 1];
                        }
                }
                v[t].push_back(sum);
        }
        for(int i = 1; i <= m; i++){
                sort(v[i].begin(), v[i].end());
        }

        LL ans = tot;
        U = 1 << (n - m);
        for(int i = 1; i < U; i++){
                int t = 0;
```

```
                LL sum = 0;
                for(int j = 0; j < n - m; j++){
                        if((i >> j) & 1){
                                t++;
                                sum += a[j + m + 1];
                        }
                }
                if(t > m) continue;
                if(t == m){
                        ans = min(ans, abs(tot - sum - sum));
                        continue;
                }
                int k = lower_bound(v[m - t].begin(), v[m - t].end(), tot / 2 - sum) - v[m - t].be
                if(k < v[m - t].size()) ans = min(ans, abs(tot - (sum + v[m - t][k]) * 2));
                if(k > 0) ans = min(ans, abs(tot - (sum + v[m - t][k - 1]) * 2));
        }

        cout << ans << endl;
        return 0;
}
```

T4 握手(hand)

题目大意：两排人员找对方的一个好友握手，但是所有的握手路线不能交叉。求最多可以有多少人同时握手。

sol1 70pts

考查知识点：排序，最长上升子序列（线性DP）。

因为握手与输入时的下标没有任何关系。

所以可以将每一对朋友(A_i, B_i)按照A_i作为关键字升序排序。

如果两对朋友(i, j)握手没有冲突，那么就满足$A_i < A_j$、$B_i < B_j$或者$A_i > A_j$、$B_i > B_j$。

而我们已经排好了序，所以如果$i < j$，当且仅当$A_i \neq A_j$且$B_i < B_j$，握手才没有冲突。

如果保证A中任意两个元素互不相同，如果我们选出的若干对朋友的（已经排好序的）下标分别为$x_1, x_2, ..., x_k (x_1 < x_2 < ... < x_k)$ 当且仅当$B_{x_1} < B_{x_2} < ... < B_{x_k}$，那么此时已经可以从中发现其实这就是一个最长上升子序列。

但是题目没保证A中任意两个元素互不相同，但是如果有多组朋友的A相同，只能选一组。

在多组朋友的A相同时，如何排序才能保证最长上升子序列在这若干个B_j中最多选一个呢?

只要在对每对朋友按照A排序，当A相同时，按B降序排序即可。

这样如果多组朋友的A相同，也不存在问题了。因为最长上升子序列不可能同时经过前大后小的两个数字。

sol2 100pts

考查知识点：最长上升子序列的二分优化。

代码如下：

```
#include<bits/stdc++.h>
using namespace std;
using PII = pair<int, int>;
const int N = 2e5 + 5;
PII a[N];
int n, f[N];
bool cmp(PII &a, PII &b){
        if(a.first != b.first) return a.first < b.first;
        return a.second > b.second;
}
int main(){
        freopen("hand.in","r",stdin);
        freopen("hand.out","w",stdout);
        ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
        cin >> n;
        for(int i = 1; i <= n; i++){
                int x, y; cin >> x >> y;
                a[i] = {x, y};
        }
        sort(a + 1, a + n + 1, cmp);

        for(int i = 1; i <= n; i++){
                f[i] = INT_MAX;
        }
        int ans = 0;
        for(int i = 1; i <= n; i++){
                int p = lower_bound(f + 1, f + n + 1, a[i].second) - f;
                f[p] = a[i].second;
                ans = max(ans, p);
        }

        cout << ans << endl;
        return 0;
}
```