2026 年 6 月 SACCC 认证六级 题解

A

首先，我们会发现后下发的文件不会影响先下发的文件的选择。于是，我们可以将文件按 t_i 排序，记录对应文件编号，并从小到大依次选择。维护每个打印机上一次完成打印的时间（初始为 0）。那么，我们将所有打印机遍历一遍，求出等待时间的最小值与等待时间相同情况下编号最小的打印机，并选择这个打印机打印即可，注意更新完成打印的时间。总复杂度 $O(nm)$，期望得分 45 分。

考虑如何维护哪个打印机等待时间最小。维护两个小根堆 p, q，p 中存储所有已经完成打印的打印机编号，q 中存储所有未完成打印的打印机编号与对应完成时间。于是，我们在 p 中有打印机的情况下一定会选择 p 中的打印机，由于我们要取编号最小的，可以直接取出 p 的堆顶。否则，我们会选择 q 中完成时间最小的，完成时间相同情况下编号最小的，也可以直接取出 q 的堆顶。注意，当 q 堆顶完成时间比目前考虑的 t_i 小，那么要从 q 中弹出并塞入 p。总复杂度 $O((n+m)logm)$，期望得分 100 分。

AC 代码

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define ll long long
#define MP make_pair
mt19937 rnd(time(0));
const int MAXN=2e5+5;
int n,m,s[MAXN],t[MAXN],p[MAXN];
vector<int> vec[MAXN];
signed main(){
    ios::sync_with_stdio(false);
    freopen("A.in","r",stdin);
    freopen("A.out","w",stdout);
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>s[i]>>t[i];
        p[i]=i;
    }
    sort(p+1,p+n+1,[&](int x,int y){return t[x]<t[y];});
    priority_queue<int,vector<int>,greater<int> > q1;
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int> > > q2;
    for(int i=1;i<=m;i++) q1.push(i);
    for(int i=1;i<=n;i++){
        int id=p[i];
        while(!q2.empty()&&q2.top().first<=t[id]){
            q1.push(q2.top().second);
            q2.pop();
```

```
        }
        if(!q1.empty()){
            int it=q1.top();q1.pop();
            vec[it].push_back(id);
            q2.push(MP(s[id]+t[id],it));
        }else{
            auto it=q2.top();q2.pop();
            vec[it.second].push_back(id);
            it.first+=s[id];
            q2.push(it);
        }
    }
    for(int i=1;i<=m;i++){
        sort(vec[i].begin(),vec[i].end());
        int k=vec[i].size();
        cout<<k<<" \n"[k==0];
        for(int j=0;j<k;j++) cout<<vec[i][j]<<" \n"[j+1==k];
    }
    return 0;
}
```

B

题意提炼

给出若干正整数硬币，每枚最多使用一次。要求找出最小的正整数，使它不能由这些硬币的某个子集和表示。

思路

排序后从小到大处理硬币。维护变量 a，表示当前还不能保证构造出的最小正整数；等价地，已经能构造所有 $[1, a-1]$ 中的数。

- 初始 $a = 1$，表示还没有硬币时无法构造 1。
- 若下一枚硬币为 b，且 $b > a$，那么无法构造 a，直接结束。
- 若 $b \le a$，那么用这枚硬币可以把可构造范围从 $[1, a-1]$ 扩展到 $[1, a+b-1]$，于是令 $a+ = b$。

正确性关键

假设当前已能构造 $[1, a-1]$。若下一枚硬币 $b \le a$，那么：$b+[0, a-1] = [b, a+b-1]$

其中 0 表示不选旧硬币。旧范围 $[1, a-1]$ 与新范围 $[b, a+b-1]$ 连在一起，没有空洞，所以能构造到 $a+b-1$。若 $b > a$，因为所有剩余硬币都至少为 b，不能用它们补出 a，答案就是 a。

AC 代码

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
typedef long long ll;
vector<int>a;
int n;
int main(){
    freopen("B.in","r",stdin);
    freopen("B.out","w",stdout);
    cin>>n;
    for(int i=1;i<=n;i++){
        int x;
        cin>>x;
        a.push_back(x);
    }
    sort(a.begin(),a.end());
    ll ans=1;
    for(ll b:a){
        if(b>ans) break;
        else ans+=b;
    }
    cout<<ans<<endl;
    return 0;
}
```

C

思路分析

为了在 n 分钟之内结束，两个厕所一刻都不能停。由于男生不能进女厕，所以一个男生必须要和一个女生一起才能上厕所。

如果把男生视为 1，女生视为 -1，那么对于任意位置，后缀和必须要小于等于 **1** 才能保证成功（当后缀和为 1 时，多出来的那个男生可以跟着他之前的女生走）。

如果出现了后缀和大于 1 怎么办？这就需要把一些男生放到前面去。如果累计的男生个数有 x 个，则至少要放 $x-1$ 个到女生前面，因此答案应该对 $x-1$ 取 $\max$。

基于此，我们可以考虑从后往前扫描字符串，用一个变量来存储后缀和，当后缀和大于 1 时，更新一下答案。

重复字符串的处理

容易想到，重复的字符串对于后缀和的影响是相同的。并且无论影响是正还是负，对答案有影响的只有可能是第一个串和所有串连在一起的情况。因此，我们可以维护一个字符串产生的后缀和变量 $delta$ 以及变化过程中的最大变化量 $maxdelta$。如

果一个字符串一共重复 t 次，且在该字符串之前的后缀和是 cnt，那么：

- 这些重复的字符串中，能达到的最大后缀和为：$\max(cnt + maxdelta, cnt + maxdelta + delta \times (t-1))$
- 整个处理完后，对总后缀和的影响为：$cnt \leftarrow cnt + delta \times t$

无解情况

最后检查无解：如果总后缀和最终大于 0（为 1 也不行，因为这意味着第一个男生无法上厕所），说明男生过多，直接输出 -1。否则，输出整个过程中最大后缀和减一（若小于 0 则输出 0）。

AC 代码

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

const int maxm = 1e5 + 5;

long long n, ans;
int m;
string s[maxm];
long long t[maxm];

int main() {
    // 优化输入输出
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    if (!(cin >> n >> m)) return 0;

    for (int i = 1; i <= m; i++) {
        cin >> s[i] >> t[i];
        // 从后往前扫，所以先将单串反转方便正向遍历
        reverse(s[i].begin(), s[i].end());
    }

    long long cnt = 0;
    for (int i = m; i >= 1; i--) {
        long long delta = 0, mxc = 0;
        for (size_t j = 0; j < s[i].size(); j++) {
            if (s[i][j] == 'F') delta--; // F 代表女生
            else delta++;                // M 代表男生
            mxc = max(mxc, delta);
        }
        ans = max(ans, mxc + cnt);
        ans = max(ans, mxc + cnt + delta * (t[i] - 1));
```

```
        cnt += delta * t[i];
    }

    if (cnt > 0) cout << -1 << "\n";
    else if (ans > 1) cout << ans - 1 << "\n";
    else cout << 0 << "\n";

    return 0;
}
```

D

明显是一个 dp

定义一个 f[i][j] 的数组来表示前 i 条木板粉刷 j 次的情况下能正确粉刷的最大格子数

定义一个 g[i][j][k] 来表示第 i 条木板上粉刷 j 次涂了前 k 个格子数的情况下能正确粉刷的最大格子数

用 sum 数组来记录蓝色格子数 某个区间的格子数减去蓝色格子数就是绿色格子数（用前缀和来记录）

要是求最大的格子数 那么就要求一个 m 使得

f[i][j]=max(f[i][j],f[i-1][j-k]+g[i][k][m])

用一个区间来想

接下来就是求 g[i][j][k] 状态转移方程了还是很好想的

前 q 个格子粉刷正确加上下一步粉刷正确的绿色格子多还是蓝色格子多

有

g[i][j][k]=max(g[i][j][k],g[i][j-1][q]+max(sum[i][k]-sum[i][q],k-q-sum[i][k]+sum[i][q]));

其中 sum[i][k]-sum[i][q] 蓝色格子

k-q-sum[i][k]+sum[i][q] 该段的绿色格子

最后看看粉刷多少次时有最大的 f[n][j] 记作 ans

```
#include<bits/stdc++.h>
using namespace std;
int f[51][2550], sum[51][2550];
int g[51][2550][51];
int n, m, t;
char s[150];

int main() {
    cin >> n >> m >> t;
```

```
    for(int i = 1; i <= n; i++){
        cin >> s;
        sum[i][0] = 0;
        for(int j = 1; j <= m; j++){
            if(s[j-1] == 'r') sum[i][j] = sum[i][j-1] + 1;
            else sum[i][j] = sum[i][j-1];
        }
    }
    for(int i = 1; i <= n; i++)
    for(int j = 1; j <= m; j++)
    for(int k = 1; k <= m; k++)
    for(int q = j - 1; q < k; q++){
        g[i][j][k] = max(g[i][j][k],
                     g[i][j-1][q] + max(sum[i][k]-sum[i][q], k-q-sum[i][k]+sum[i][q]));
    }
    for(int i = 1; i <= n; i++)
    for(int j = 1; j <= t; j++)
    for(int k = 0; k <= min(j, m); k++){
        f[i][j] = max(f[i][j], f[i-1][j-k] + g[i][k][m]); }
    int ans = 0;
    for(int i = 1; i <= t; i++) ans = max(ans, f[n][i]);
    cout << ans;
}
```