CSP 2020 J 题解

张晴川 @ 实验舱

November 30, 2020

1 优秀的拆分

大意

给定正整数 n，将其拆分为若干不同的 2 的**正整数**次幂之和或输出无解。

数据范围

- $1 \leq n \leq 10^7$

题解

由于 2 的正整数次幂一定是偶数，如果 n 是奇数，直接输出无解。否则从大到小枚举 2 的幂，如果 n 在二进制下的的对应位是 1 就输出。具体参考代码。

复杂度

- 时间：$O(\log(n))$
- 空间：$O(1)$

代码

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

int main()
{
        int n;
        cin>>n;
        if(n%2==1){
                cout<<-1;
        }else{
                for(int i = 1<<30;i>0;i/=2){
                        if(n & i){
```

```
                                        cout<<i<<" ";
                                }
                        }
                }
        return 0;
}
```

2 直播获奖

大意

有 n 个人，第 i 个人的分数为一个整数 a_i。给定获奖比例 w。假设只考虑前 p 个人，那么获奖分数线为从高到低第 $\max(1, \lfloor p * w\% \rfloor)$ 个人的成绩。求出 $p = 1, 2, \ldots, n$ 时的分数线。

数据范围

- $1 \leq n \leq 10^5$
- $1 \leq w \leq 99$
- $0 \leq a_i \leq 600$

题解

考虑模拟题意，枚举 $p = 1, 2, \ldots, n$。首先计算使用题中的公式计算出获奖人数，注意需要全部使用整数计算。

如果每次将 p 个人的成绩重新排序然后得到分数线，那么只能得到过 6/20 个点。但如果每次使用插入排序，（应当）可以通过 $n \leq 2000$ 的测试点。

考虑到每个人的分数都是 $[0, 600]$ 内的整数，我们只需要对每个成绩开一个桶，每次相同分数的人可以一起考虑。该做法复杂度为 $O(601n)$，可以通过。

复杂度

- 时间：$O(601n)$
- 空间：$O(601)$

代码

```
#include<bits/stdc++.h>
using namespace std;
int n,w,cnt[601];
int main()
{
        cin>>n>>w;
```

```
        for(int i = 1;i<=n;i++){
                int c;cin>>c;
                cnt[c]++;
                int tot = max(1,i*w/100);
                for(int j = 600;j>=0;j--){
                        tot -= cnt[j];
                        if(tot <= 0){
                                cout<<j<<" ";
                                break;
                        }
                }
        }
        return 0;
}
```

3 表达式

大意

给定一个包含 n 个布尔变量、与运算 &、或运算 | 以及取反运算 ! 的后缀表达式 s。每个变量出现恰好一次。给定每个变量的初始值。每次询问将一个变量取反，求原始表达式的真值。每次询问独立，即并不真的修改变量的值。

数据范围

- $1 \le |s| \le 10^6$
- $1 \le q \le 10^5$
- $2 \le n \le 10^5$

题解

首先使用一个栈将后缀表达式转为一棵表达式树，并记录每个变量对应的节点。

针对 $n \le 1000$ 的测试点，我们只需要每次修改对应变量的值，并 DFS 整棵表达式树重新求值即可，复杂度为 $O(n^2)$。

为了通过所有测试点，首先我们需要用 DFS 自下而上预处理出表达树上每棵子树的值。现在我们在每棵子树的根节点上开设两个变量，$dp[0/1]$，分别表示当前子树的值是 $0/1$ 时，整棵表达式树的值。但这应如何计算?

对于根节点，显然 $dp[0/1] = 0/1$。而对于非根节点，我们可根据其父亲的操作符类型分类讨论，`cousin->val` 表示兄弟节点的初始值:

- &: 那么 `dp[v] = father->dp[v & cousin->val]`
- |: 那么 `dp[v] = father->dp[v | cousin->val]`
- !: 那么 `dp[v] = father->dp[!v]`

根据以上公式，我们可以自上而下地 $O(n)$ 计算出每个节点的 dp 值。针对每次询问，只需要 $O(1)$ 查表回答即可。

复杂度

- 时间：$O(n)$
- 空间：$O(n)$

代码

```cpp
#include<bits/stdc++.h>
using namespace std;
struct Node{
        int dp[2],val;
        string op;
        vector<Node*>son;
        Node(){
                dp[0] = dp[1] = val = 0;
                op = "";
                son = vector<Node*>();
        }
};
const int N = 100100;
Node* ptr[N];
stack<Node*>st;
// 第一次 dfs 求默认值
void dfs(Node*cur){
        if(cur->op=="")return;
        for(int i = 0;i<cur->son.size();i++){
                dfs(cur->son[i]);
        }
        if(cur->op == "!"){
                cur->val = !cur->son[0]->val;
        }else if(cur->op=="&"){
                cur->val = cur->son[0]->val & cur->son[1]->val;
        } else{
                cur->val = cur->son[0]->val | cur->son[1]->val;
        }
}
// 第二次 dfs 求 dp 值
void ddfs(Node*cur,Node*fa){
        if(fa == NULL){
                cur->dp[0] = 0;
                cur->dp[1] = 1;
        }else if(fa->op == "!"){
```

```
                for(int i = 0;i<2;i++){
                        cur->dp[i] = fa->dp[!i];
                }
        }else if(fa->op == "&"){
                Node*other;
                for(int i = 0;i<2;i++){
                        if(fa->son[i] != cur)other = fa->son[i];
                }
                for(int i = 0;i<2;i++){
                        cur->dp[i] = fa->dp[i & other->val];
                }
        }else if(fa->op == "|"){
                Node*other;
                for(int i = 0;i<2;i++){
                        if(fa->son[i] != cur)other = fa->son[i];
                }
                for(int i = 0;i<2;i++){
                        cur->dp[i] = fa->dp[i | other->val];
                }
        }
        for(int i = 0;i<cur->son.size();i++){
                ddfs(cur->son[i],cur);
        }
}
int main(){
        string s;
        while(cin>>s){
                if(s=="!"){
                        Node*a = st.top();
                        st.pop();
                        Node*c = new Node();
                        c->op = s;
                        c->son.push_back(a);
                        st.push(c);
                }else if(s=="&" or s == "|"){
                        Node*a = st.top();
                        st.pop();
                        Node*b = st.top();
                        st.pop();
                        Node*c = new Node();
                        c->op = s;
                        c->son.push_back(a);
                        c->son.push_back(b);
                        st.push(c);
                }else if(s[0]=='x'){
                        int id = 0;
```

```
                        for(int i = 1;i<s.size();i++)id = id*10 + s[i]-'0';
                        ptr[id] = new Node();
                        st.push(ptr[id]);
                }else{
                        int n = 0;
                        for(int i = 0;i<s.size();i++)n = n*10 + s[i]-'0';
                        for(int i = 1;i<=n;i++){
                                cin>>ptr[i]->val;
                        }
                        break;
                }
        }
        Node*root = st.top();
        dfs(root);
        ddfs(root,NULL);
        int q;cin>>q;
        while(q--){
                int id;cin>>id;
                cout<<ptr[id]->dp[!ptr[id]->val]<<"\n";
        }
        return 0;
}
```

4 方格取数

大意

给定二维矩阵 $a[n][m]$。现在需要从左上角走到右下角，每次可以向上、下、右走一步，但不能重复经过同一个格子，求最大化走过的格子的权值和是多少。

数据范围

- $1 \le n, m \le 10^3$
- $|a[i][j]| \le 10^4$

题解

设 $dp[i][j]$ 为从起点走到 $a[i][j]$ 的最大权值和。

第一列的值是容易求的，因为从起点出发只有一种走法。

考虑已经求完了第 j 列的 dp 值，应该如何推出第 $j+1$ 列的值呢?

假设在计算 $dp[i][j+1]$，我们可以枚举是从第 j 列的哪个格子向右走过来的，不妨设为 i'。这里有两种情况 $i' \le i$ 和 $i' \ge i$，先假设 $i' \le i$，另一种情况类似。

那么可以得到：

$$dp[i][j+1] = \min_{1 \le i' \le i} (dp[i'][j] + \sum_{x=i'}^{i} a[x][j+1])$$

根据这个式子暴力计算，可以通过 70% 的数据。

但我们可以利用式子的特殊性质进行加速，注意到：

$$dp[i+1][j+1] = \min_{1 \le i' \le i+1} (dp[i'][j] + \sum_{x=i'}^{i+1} a[x][j+1]) \tag{1}$$

$$= \min_{1 \le i' \le i+1} (dp[i'][j] + \sum_{x=i'}^{i} a[x][j+1]) + a[i+1][j+1] \tag{2}$$

$$= \min(dp[i][j+1], dp[i+1][j]) + a[i+1][j+1] \tag{3}$$

所以只需要从上面的格子 $O(1)$ 转移过来即可，对于 $i' \ge i$ 的情况也一样。于是对于每一列，只需要上下各扫一次即可完成计算，总复杂度 $O(nm)$，可以通过。

复杂度

- 时间：$O(nm)$
- 空间：$O(nm)$

代码

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1010;
ll dp[N][N],a[N][N],n,m;
int main(){
        memset(dp,0xcf,sizeof dp);
        cin>>n>>m;
        for(int i = 0;i<n;i++){
                for(int j = 0;j<m;j++){
                        scanf("%lld",&a[i][j]);
                }
        }
        dp[0][0] = a[0][0];
        for(int i = 1;i<n;i++){
                dp[i][0] = dp[i-1][0] + a[i][0];
        }
        for(int j = 1;j<m;j++){
                ll v = -1e15;
```

```
                for(int i = 0;i<n;i++){
                        v = max(v,dp[i][j-1]);
                        v += a[i][j];
                        dp[i][j] = max(v,dp[i][j]);
                }
                v = -1e15;
                for(int i = n-1;i>=0;i--){
                        v = max(v,dp[i][j-1]);
                        v += a[i][j];
                        dp[i][j] = max(v,dp[i][j]);
                }
        }
        cout<<dp[n-1][m-1]<<endl;
        return 0;
}
```