鸡尾酒QAQ
编辑于 10-07 07:49 上海

+ 关注

2022OI集训营普及组第二场题解 精

T1 隔离

要么只隔离一次，要么一直来回 AB 两地，不被隔离。如果去一趟 B 地把所有事情办完也是回来隔离，办一部分回来也是隔离，那肯定选择一次办完所有事再回来隔离。

分类讨论这两种情况哪一种耗时多就可以了。

```
#include <iostream>
using namespace std;
int main(){     int n, a;     cin >> n;     int sum = 0, now = 0, ans = 0;     for
}
```

T2 和积

可以直接暴力枚举判断 $[M, N]$ 中的数，一个一个求数字和然后判断。

这样复杂度是 $O(N \log N)$，可以得到 70 分。

注意到如果设 $\mathrm{Sum}(x)$ 为 x 在十进制下的数字和，那么有 $\mathrm{Sum}(x) = \mathrm{Sum}(\lfloor x/10 \rfloor) + (x \bmod 10)$。

同理也可以这样计算积。

因此可以 $O(N)$ 先预处理每个数的数字和积再枚举，时间复杂度 $O(N)$，期望得分 100。

T3 电梯

假设某人要乘坐电梯，从楼层 a 到楼层 b，若电梯停靠位置 x 在 $a \sim b$ 之间，则电梯移动距离为 $|x - a| + |a - b| + |b - x| = 2 * |a - b|$。

若停靠位置在 $a - 1$，则电梯要多移动 2 距离。若停靠位置在 $a - 2$，则电梯要多移动 4 距离。我们可以发现，停靠位置从 $a - 1$ 到 1，电梯多移动的距离呈公差为 2 的等差数列。

我们假设 $f[x]$ 表示电梯停靠在 x 的总移动距离，那么给定 a, b 的时候，相当于将整个数组全部加 $2 * |a - b|$，并且将 $a - 1$ 到 1 的位置额外加一个公差为 2 的等差数列；同理，将 $b + 1$ 到 n 的位置加一个公差为 2 的等差数列。

考虑用差分来维护区间加法，等差数列加法，即可解决此题。

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n, m, a, b;
    long long ans = 1e18, pos, sum = 0;
```

```
    cin >> n >> m;
    vector<long long>pre(n+2), suf(n+2), pre1(n+2), suf1(n+2);
    while(m--){
        cin >> a >> b;
        pre[a-1] += 2;
        suf[b+1] += 2;
        sum += (b - a) * 2;
    }
    for(int i = n; i >= 1; i--){
        pre[i] += pre[i+1];
        pre1[i] = pre1[i+1] + pre[i];
    }
    for(int i = 1; i <= n; i++){
        suf[i] += suf[i-1];
        suf1[i] += suf1[i-1] + suf[i];
        if(suf1[i] + pre1[i] < ans){
            pos = i;
            ans = suf1[i] + pre1[i];
        }
    }
    cout << pos << " " << ans + sum << endl;
}
```

T4 分组选数

10pt

考虑 $m = n, b = 1$ 的情况，即可以在一个组之内任意选数。

建立数组 $dp[i][j]$，表示看到第 i 个数，能否异或出 j（能为1，不能为0）则不难得到代码为：

```
for(int i = 1;i <= n;i++){
    for(int j = 1;j <= 2047;j++){
        if(dp[i-1][j]){
            dp[i][j] = 1;
            dp[i][j^a[i]] = 1;
        }
    }
}
```

20pt

然后考虑加入 m 的限制。

在前一种情况的基础上，dp 数组里存的值仅是 0 或 1，没有得到充分的利用，因此可以将 dp 存储的内容变为看到第 i 个数，异或出 j 至少所需要的数字个数。但此时转移时需要求min，无法异或的情况不能用0来表示，可以用1e9来表示，于是得到代码：

```
for(int i = 1;i <= n;i++){
    for(int j = 1;j <= 2047;j++){
        if(dp[i-1][j] != 1e9){
            dp[i][j] = min(dp[i][j],dp[i-1][j]);
            dp[i][j^a[i]] = min(dp[i-1][j],dp[i][j^a[i]]);
        }
    }
}
```

最后只需要遍历一遍的同时判断个数小于等于 m 即可

50~100 pt

此时不难想到对于每一个集合进行一次如上操作，但若这样开成三维数组会超内存，故可以之用二维数组，但每次使用后清空。这样就需要再开一个数组 $num[i][j]$ 表示第 i 组选 j 个数最大的异或值。每一组结束 dp 预处理操作后存入 num 中，最后使用一个分组背包即可得到最大异或值。但是要注意，在每一

组 dp 预处理时， i 应循环到组中元素的个数而不是 n 否则会超时。 由于 n 的总值仍是 2000，所以操作时组和i的循环次数总和仍为 2000，时间复杂度不变。

备注：最后使用分组背包的话就是 $O(n^2)$，如果循环条件没写好，那就是 $O(n^3)$

```cpp
#include <bits/stdc++.h>
using namespace std;
int n,m,dp[2005][2050],num[2005][2005] = {},dpp[2050] = {},zz[2005] = {};  //dp[i]
vector<int> ve[2005];
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i++){
        int x,y;
        cin >> x >> y;
        ve[y].push_back(x);
        zz[y]++;
    }
    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= 2047;j++){
            dp[i][j] = 1e9;
        }
    }
    for(int zu = 1;zu <= 2000;zu++){
        if(zz[zu] != 0)   dp[1][ve[zu][0]] = 1;  //不加判断的话会访问越界导致运行崩
        for(int i = 2;i <= zz[zu];i++){
            dp[i][ve[zu][i-1]] = 1;  //这一步很重要，不要忘记
            for(int j = 1;j <= 2047;j++){
                if(dp[i-1][j] != 1e9){
                    dp[i][j] = min(dp[i][j],dp[i-1][j]);
                    dp[i][j^ve[zu][i-1]] = min(dp[i-1][j]+1,dp[i][j^ve[zu][i-1]]);
                }
            }
        }
        for(int j = 1;j <= 2047;j++){
            if(dp[zz[zu]][j] != 1e9)  num[zu][dp[zz[zu]][j]] = max(num[zu][dp[zz[z
        }
        for(int i = 1;i <= zz[zu];i++){
            for(int j = 1;j <= 2047;j++){
                dp[i][j] = 1e9;
            }
        }
    }
    for(int i = 1;i <= 2000;i++){
        for(int j = m;j >= 1;j--){
            for(int k = 1;k <= zz[i];k++){ // 最后的枚举，只枚举到本组的个数
                if(j >= k)  dpp[j] = max(dpp[j],dpp[j-k] + num[i][k]);
            }
        }
    }
    cout << dpp[m];
}
```