JOI 2022/2023 春季トレーニング Day 2
議会 (Council) 解説

解説担当 : tatyam

問題概要

- N 人の人がいて，M 個の案に投票を行う.
- $A_{i,j} = 1$ のとき，人 i は案 j に賛成票を投じる.
- $i = 1, 2, \ldots, N$ について，以下の問題を解け.
  - $j\ (\neq i)$ を選び，人 i と人 j を投票から除外する.
  - 過半数 ($\lfloor N/2 \rfloor$ 票以上) の賛成票を獲得する案の個数の最大値は？

$N \leq 3 \times 10^5$, $\boldsymbol{M \leq 20}$

小課題 (JOI)

番号	配点	制約	計算量の例
1.	8 点	$N \leqq 300$	$O(N^3 M)$
2.	8 点	$N \leqq 3000$	$O(N^2 M)$
3.	6 点	$M \leqq 2$	
4.	19 点	$M \leqq 10$	$O(N\,2^M)$
5.	15 点	$M \leqq 14$	$O(NM + 4^M)$
6.	22 点	$M \leqq 17$	$O(NM + 3^M M)$
7.	22 点	$M \leqq 20$	$O(NM + 2^M M^2)$

小課題 (JOIG)

番号	配点	制約	計算量の例
1.	13 点	$N \leqq 300$	$O(N^3 M)$
2.	17 点	$N \leqq 3000$	$O(N^2 M)$
3.	10 点	$M \leqq 2$	
4.	18 点	$M \leqq 10$	$O(N\,2^M)$
5.	12 点	$M \leqq 14$	$O(NM + 4^M)$
6.	22 点	$M \leqq 17$	$O(NM + 3^M M)$
7.	8 点	$M \leqq 20$	$O(NM + 2^M M^2)$

小課題 1 $(N \leqq 300)$

- $O(N^3 M)$ くらいの計算量で解けば良い.

小課題 1 $(N \leqq 300)$

- $O(N^3M)$ くらいの計算量で解けば良い.
- 議長 i を全探索 : $O(N)$
- 副議長 j を全探索 : $O(N)$
- それぞれの案に何票集まったかを集計 : $O(NM)$

$\Longrightarrow$ 全部合わせて $O(N^3M)$

小課題 2 ($N \leqq 3000$)

- 議長 i を全探索 : $O(N)$
- 副議長 j を全探索 : $O(N)$
- **それぞれの案に何票集まったかを集計 : $O(NM)$**

議長と副議長の 2 人しか変わらないのに，毎回集計し直すのは無駄！

小課題 2 ($N \leqq 3000$)

- それぞれの案に何票集まっているかを前計算 : $O(NM)$

その後,

- 議長 i を全探索 : $O(N)$
- 副議長 j を全探索 : $O(N)$
- 議長と副議長の分を取り除く : $O(M)$

$\Longrightarrow$ 全部合わせて $O(N^2M)$

小課題 2 ($N \leqq 3000$)

- それぞれの案に何票集まっているかを前計算 : $O(NM)$

その後,

- 議長 i を全探索 : $O(N)$
- 副議長 j を全探索 : $O(N)$
- 議長と副議長の分を取り除く : $O(M)$

$\Longrightarrow$ 全部合わせて $O(N^2M)$

実装テクニック：整数をビット列として扱おう

- ビット演算 `|` `&` `^` `<<` `>>` を使って，整数をビット列として扱うことができる．
- 議長と副議長を決めたときに，成立する案の個数を $O(M)$ で求めていたが，これを $O(1)$ にできる．

実装テクニック：整数をビット列として扱おう

A をビット列で入力する

```
vector<int> A(N);

for(int i = 0; i < N; i++) for(int j = 0; j < M; j++) {
    int a;
    cin >> a;
    A[i] |= a << j;
}
```

考察

各案は

- 議長・副議長に関わらず可決
- 議長・副議長のどちらかが反対なら可決
- 議長・副議長の両方が反対なら可決
- 議長・副議長に関わらず否決

の 4 通りに分けられる．そのうち中央の 2 つのみが重要である．

実装テクニック : 整数をビット列として扱おう

- 可決票数 $+ 2$ 票以上の賛成票を得ている案の集合
- 可決票数 $+ 1$ 票の賛成票を得ている案の集合
- 可決票数 $+ 0$ 票の賛成票を得ている案の集合

をそれぞれ整数で持つ

```
for(int j = 0; j < M; j++) {
    if(vote[j] >= N / 2 + 2) two |= 1 << j;
    if(vote[j] == N / 2 + 1) one |= 1 << j;
    if(vote[j] == N / 2) zero |= 1 << j;
}
```

実装テクニック：整数をビット列として扱おう

- 議長 i と副議長 j を決めたら，可決される案の集合が $O(1)$ で求められる

```
int accept = two | one & ~(A[i] & A[j]) | zero & ~(A[i] | A[j]);
```

あとは，これの立っているビットの個数を数えれば良い．

小課題 2 ($N \leqq 3000$)

popcount

- C++20

どうして…

/ リファレンス / bit / popcount

function template

std::popcount C++20

```
namespace std {
  template <class T>
  constexpr int popcount(T x) noexcept;
}
```

概要

立っているビットを数える。popcountは「population count」の略。

popcount

- `std::bitset::count` を使う

```
int popcnt(int x) { return bitset<32>(x).count(); }
```

- `__builtin_popcount` を使う

```
int popcnt(int x) { return __builtin_popcount(x); }
```

これらは通常賢いビット演算をしている `__popcountdi2` にコンパイルされるが，今回はコンパイルオプションに `-march=native` が指定されていたので，`POPCNT` 命令にコンパイルされる 🙌

popcount

- 自分で実装する

```
int popcnt(int x) {
    int ans = 0;
    while(x) {
        ans += x & 1;
        x >>= 1;
    }
    return ans;
}
```

計算量は $O(M)$

popcount

- 全部前計算しておく

```
int popcnt(int x) {
    static array<int, 1 << 20> table = []{
        // 前計算する
    }();
    return table[x];
}
```

計算量は $O(1)$

クイズ

どっちが速い？

[A]

```
int popcnt(int x) {
    int ans = 0;
    while(x) {
        ans += x & 1;
        x >>= 1;
    }
    return ans;
}
```

計算量は $O(M)$

[B]

```
int popcnt(int x) {
    static array<int, 1 << 20> table = []{
        // 前計算する
    }();
    return table[x];
}
```

計算量は $O(1)$

答え

[A]

```
int popcnt(int x) {
    int ans = 0;
    while(x) {
        ans += x & 1;
        x >>= 1;
    }
    return ans;
}
```

計算量は $O(M)$

- キャッシュの効かないメモリアクセスは, ビット演算や足し算に比べて非常に遅い！！

popcnt

- 賢いビット演算

```
int popcnt(int x) {
    x = (x & 0x55555555) + (x >> 1 & 0x55555555);
    x = (x & 0x33333333) + (x >> 2 & 0x33333333);
    x = (x & 0x0f0f0f0f) + (x >> 4 & 0x0f0f0f0f);
    x = (x & 0x00ff00ff) + (x >> 8 & 0x00ff00ff);
    return (x & 0x0000ffff) + (x >> 16);
}
```

計算量は $O(\log M)$

小課題 3 $(M \leqq 2)$

- なんと $M \leqq 2$
- $M = 2$ のとき，$(0,0), (0,1), (1,0), (1,1)$ の 4 種類のタイプの人間がいる．

こんな例を考える.

	案 1	案 2
人 1	0	1
人 2	0	1
人 3	1	0
人 4	1	0
人 5	1	1
合計	3	3

$N = 5, M = 2$

可決票数 : 2 票

人 1 が議長の場合と人 2 が議長の場合で，状況が全く同じ．

	案 1	案 2
人 2	0	1
人 3	1	0
人 4	1	0
人 5	1	1
合計	3	2

	案 1	案 2
人 1	0	1
人 3	1	0
人 4	1	0
人 5	1	1
合計	3	2

つまり，**答えも同じ**．

結局, 「議長と副議長が 4 種類のタイプのうちどのタイプの人間か？」が分かれば可決される案も分かる.

副議長が :	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
議長が : $(0,0)$	2	2	2	2
$(0,1)$	2	1	2	1
$(1,0)$	2	2	1	1
$(1,1)$	2	1	1	0

この $4 \times 4 = 16$ 種類を調べれば良い.

小課題 $4, 5$ $(M \leqq 10, M \leqq 14)$

- M が小さめ
- $2^{14} \approx 1.6 \times 10^4 \ll 3 \times 10^5$ なので，同じタイプの人間を圧縮することで N を小さくできる.
- 小課題 3 で $4 \times 4 = 16$ 通りを調べたのと同様に，$2^M \times 2^M = 4^M$ 通りを全部調べることができる.

計算量は $O(4^M)$

小課題 6 ($M \leqq 17$)

- どうにかして計算量を $O(3^M \mathrm{poly}(M))$ にしたい.

小課題 6 $(M \leqq 17)$

i, j を決めたとき，可決される案の集合は以下のように計算された.

```
accept := two | one & ~(A[i] & A[j]) | zero & ~(A[i] | A[j])
```

ここで i を固定すると,

- j によって可決されるか変わる案 (可決票数 $+ 0$ 票)
- j に関わらず可決される案 (可決票数 $+ 1$ 票以上)

の 2 つを持てば良い.

小課題 6 ($M \leqq 17$)

i を固定すると,

- j によって可決されるか変わる案 (可決票数 $+ 0$ 票)
- j に関わらず可決される案 (可決票数 $+ 1$ 票以上)

の 2 つを持てば良い.

```
ONE := two | (one & ~A[i])
ZERO := (one & A[i]) | (zero & ~A[i])
accept := ONE | (ZERO & ~A[j])
```

小課題 6 $(M \leqq 17)$

```
accept := ONE | (ZERO & ~A[j])
```

- `popcnt(accept)` を最大化したい.
- つまり, `popcnt(ZERO & ~A[j])` を最大化したい.
- 案の全体集合を `ZERO` に絞ったときの, 0 の個数が (人 i を除いて) 最も多い人を探したい.

小課題 6 ($M \leqq 17$)

すべての $x \subset \mathbf{ZERO}$ であるような $(x, \mathbf{ZERO})$ の組 (3^M 個) について，以下が求まれば良い.

- x に含まれる案に反対で，$\mathbf{ZERO} \setminus x$ に含まれる案に賛成であるような人の数 ($\mathbf{ZERO}$ に含まれない案については 賛成 / 反対 どちらでもよい)

これは，**高速ゼータ変換** (M 次元累積和) をして，$\mathbf{ZERO}$ の部分だけ戻すことを全ての $\mathbf{ZERO}$ について行えば，$O(3^M M)$ 時間になる.

小課題 7 $(M \leqq 20)$

- 高速ゼータ変換をどうにかして計算量を $O(2^M \mathrm{poly}(M))$ にしたい.

小課題 7 ($M \leqq 20$)

```
accept := ONE | (ZERO & ~A[j])
```

→ わかりやすさのため `A[j]` を反転させておく.

各 $\mathbf{ZERO}$ に対し,

$$\max_{j \neq i} \left| \mathbf{ZERO} \cap A_j \right|$$

を求める問題である.

小課題 7 ($M \leqq 20$)

$$\max_{j \neq i} \left| \mathrm{ZERO} \cap A_j \right|$$

$j \neq i$ の制約が面倒なので, 一旦忘れると

$$\max_{j} \left| \mathrm{ZERO} \cap A_j \right|$$

となる.

$$\max_j \left|\mathbf{ZERO} \cap A_j\right|$$

A_j を $\mathbf{ZERO}$ まで減らしたときの集合の大きさ (popcount) の最大値であるので，各 A_j から要素を減らしていったときに現れる集合を前計算しておく．

$$B = \{b \mid \exists j : b \subset A_j\}$$

これは下位集合への累積 OR で $O(2^M M)$ でできる．問題は

$$\max\{|b| \mid b \in B,\ b \subset \mathbf{ZERO}\}$$

となった．

$$\max\{|b| \mid b \in B,\ b \subset \mathrm{ZERO}\}$$

今度は，各 b から要素を増やしていって，ZERO まで増やしたときの元々の集合の大きさ (popcount) の最大値という問題である.
したがって，上位集合への chmax をすることで，各 ZERO に対する答えが $O(2^M M)$ で求められる.

まとめ

$j \neq i$ の制約を忘れると,

1. bool 配列 `dp1[a]` を作る
2. 各 A_j に対し, `dp1[A[j]]` に 1 を代入
3. 下位集合に累積 OR
4. int 配列 `dp2[b]` を作る
5. 各 b に対し, `dp1[b] == 1` であれば, `dp2[b]` に `popcnt(b)` を代入
6. 上位集合に累積 chmax

自分自身の分を取り除くには？

自分自身の分を取り除くには？

→ 個数を数えれば良い！

1. int 配列 `dp1[a]` を作る
2. 各 A_j に対し，`dp1[A[j]]` に 1 を加算
3. 下位集合に累積和
4. int 配列 `dp2[k][b]` を作る **(popcount の値ごとに個数を数える)**
5. 各 b に対し，`dp2[popcnt(b)][b]` に `dp1[b]` を加算
6. popcount の値ごとに，上位集合に累積和

自分自身の分を取り除くには？

1. int 配列 `dp1[a]` を作る
2. 各 A_j に対し，`dp1[A[j]]` に 1 を加算
3. 下位集合に累積和
4. int 配列 `dp2[k][b]` を作る **(popcount の値ごとに個数を数える)**
5. 各 b に対し，`dp2[popcnt(b)][b]` に `dp1[b]` を加算
6. popcount の値ごとに，上位集合に累積和

自分自身の影響は簡単に数えられるので，その分を引けば良い.

計算量は $O(2^M M^2)$
$O(2^M M)$ にもなりますが，省略します.

統計情報 (JOI)

点数	1	2	3	4	5	6	7	人数	累積人数
100	O	O	O	O	O	O	O	10	10
56	O	O	O	O	O	-	-	5	15
41	O	O	O	O	-	-	-	6	21
33	O	-	O	O	-	-	-	1	22
22	O	O	O	-	-	-	-	1	23
16	O	O	-	-	-	-	-	3	26
0	-	-	-	-	-	-	-	1	27

統計情報 (JOIG)

点数	1	2	3	4	5	6	7	人数	累積人数
58	O	O	O	O	-	-	-	3	3
40	O	O	O	-	-	-	-	2	5
30	O	O	-	-	-	-	-	5	10