even number addics

简介

```
给定一一个数字序列。
两个人，玩一个关于该序列的游戏。
每人在上面选择一个数字；
alice先选。这样交替进行。当没有数字可以选择的时候，游戏结束。
若alice选的数字和是偶数，alice赢，否则是bob赢。
```

$20min$

```
尝试用 dp解决问题：
d[i]//表示在前i个数中，是否必赢。
当接上下一个数时。//
所谓必赢，就是说，某一个状态之下，无论如何，后续发展，都可以必赢。
```

题解（寻找到一个具体的确定的方案。使得必胜或者必输）

方法一： 博弈：寻找出一种方案，使得两方必然有一方获胜。

关注 a--(偶数的数量) b--奇数的数量。
b%4=0; 说明，奇数的个数是偶数个，且除二为奇数。alice追求平分。bob选择奇数，紧跟着选择奇数。如果选择偶数，紧跟着选择偶数。这样一直进行，如果还有偶数剩余，也可以保证，前面两者奇数的个数是一样的。当没偶数选了。就开始平分后面的奇数。肯定可以平分。最终alice必然可以赢。

b%4=1; 说明，奇数的个数是奇数个。谁先选，如果alice先选，等效于第一个情况，bob可以采取上述策略，使得alice拿到的和是奇数。如果bob先选，也是上面的方案。必然是bob赢。所以只需要计算谁先拿到即可。这样两者都保持选择偶数。如果 a%2=0,alice先拿，a%2=1,bob先拿。

b%4=2; 说明，奇数的个数是偶数，其中除二是奇数个。只要alice先选，bob就紧接着选。这样的策略，可以保证奇数平分。此时是bob赢

b%4=3; 说明，说明是奇数的数目奇数个，除二后向下取整，是奇数个。只要alice先手拿一个奇数，转化上一个问题。
控制奇数平分就可以赢。

code

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e5 + 10;

void solve()
{
    int n;
    cin >> n;
    int a[2]{};
    for (int i = 1, b; i <= n; i++)
    {
        cin >> b;
```

```
        a[b & 1]++;
    }
    if (a[1] % 4 == 0 || a[1] % 4 == 3 || (a[1] % 4 == 1 && a[0] %
2 == 1))
    {
        cout << "Alice" << '\n';
        return;
    }
    else
        cout << "Bob" << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int t;
    cin >> t;
    while (t--)
        solve();
}
```

动态规划

```
记忆化搜索：
dfs(int u,int now, int s0,int s1)从当前状态开始选：达到当前状态，u开始选。u是否必赢。
f[u][now][s0][s1];相关状态。
u=0
-》f[1-u][now][s0-1][s1](条件满足得情形之下。)
-》f[1-u][1-now][s0][s1-1](条件满足情况之下。)
u=1
-> f[1-u][now][s0-1][s1];
-> f[1-u][now][s0][s1-1];
//////////////////
由于两者都是采取的最优策略，所以向下一步的迁移中，对手只要有一个必输，那么当前的状态下，先手就是赢的。
```

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e2 + 10;
int f[2][2][maxn][maxn]; //分别表示现在是谁的回合  ，alice拿的是奇数还是偶数。厂商剩余的奇数个数，偶数个数。
int dfs(int u, int now, int c0, int c1){
    if (f[u][now][c0][c1] != -1)
        return f[u][now][c0][c1];
    if (c0 + c1 == 0) //说明现在结束。
    {
        if (u == 0 && now == 0)
            return f[u][now][c0][c1] = 1;
        if (u == 0 && now == 1)
            return f[u][now][c0][c1] = 0;
        if (u == 1 && now == 0)
            return f[u][now][c0][c1] = 0;
        if (u == 1 && now == 1)
            return f[u][now][c0][c1] = 1;
    }
    if (c0) //说明偶数现在还有的选；
```

```
        if (dfs(u ^ 1, now, c0 - 1, c1) == 0)
            return f[u][now][c0][c1] = 1;
    if (c1) //说明现在奇数还有的选。
    {
        if (u == 0)
        {
            if (dfs(u ^ 1, now ^ 1, c0, c1 - 1) == 0)
                return f[u][now][c0][c1] = 1;
        }
        else if (dfs(u ^ 1, now, c0, c1 - 1) == 0)
            return f[u][now][c0][c1] = 1;
    }
    return f[u][now][c0][c1] = 0;
}
void solve(){
    int n;
    cin >> n;
    int c[2]{};

    for (int i = 1; i <= n; i++){
        int a;
        cin >> a;
        c[a & 1]++;
    }
    cout << (dfs(0, 0, c[0], c[1]) ? "Alice" : "Bob") << '\n';
}
int main(){

    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int t;
    cin >> t;

    for (int i = 0; i <= 110; i++)
        for (int j = 0; j <= 110; j++)
            f[1][1][i][j] = f[1][0][i][j] = f[0][0][i][j] = f[0][1]
[i][j] = -1;
```

```
57      while (t--)
58          solve();
59  }
```

抽象出二维数组的写法：