$C1Make\quad Nonzero\quad Sum(easy\quad version)$

- addressing
- chenjiuri_10221023fdf

> 给出一个数组，其中是元素是正负1；
> 将它们分成若干段：
> aL+a(l+1)这样的形式向下推导。
> 是否可以构造出为0的情况？

关键问题的解决

- 怎么判定不存在解的情况?
- 怎么构造出一个这样的等于0的东西出来?
- 管理一些什么信息？即使是答案输出的时候，有应该何?

> 找到一个点等价的式子表达：
> 最终必然可以转化为两两之间的组合。
> 所以问题就变为，细度小于等于2的若干个组合，使得它们的和为0；
>
> 这样的一种小于等于2的解只有几种可能：
> -2，0，2；1，-1；
>
> 如果最终的长度为奇数：
> 无论如何都有多余出来的一个奇数位。
> 而前面的一堆数字最终的结果2的整数倍数。
> 本质上就是对它们做一个加减的贡献转换，无论怎么样，都不会使得和为偶数。所以必然错误。
>
> 如果最终的长度为偶数，是否有一种必然的构造策略？？

随便找一个大佬代码来分析分析

```
#include <bits/stdc++.h>
using namespace std;
#define F first
#define S second
#define R cin>>
#define ln cout<<'\n'
#define ll long long
#define in(a) insert(a)
#define pb(a) push_back(a)
#define pd(a) printf("%.10f\n",a)
#define mem(a) memset(a,0,sizeof(a))
#define all(c) (c).begin(),(c).end()
#define iter(c) __typeof((c).begin())
#define rrep(i,n) for(ll i=(ll)(n)-1;i>=0;i--)
```

```
#define REP(i,m,n) for(ll i=(ll)(m);i<(ll)(n);i++)
#define rep(i,n) REP(i,0,n)
#define tr(it,c) for(iter(c) it=(c).begin();it!=(c).end();it++)
ll check(ll n,ll m,ll x,ll y){return x>=0&&x<n&&y>=0&&y<m;}void pr()
{ln;}
template<class A,class...B>void pr(const A &a,const B&...b)
{cout<<a<<(sizeof...(b)?" ":"");pr(b...);}
template<class A>void PR(A a,ll n){rep(i,n)cout<<(i?" ":"")<<a[i];ln;}
const ll MAX=1e9+7,MAXL=1LL<<61,dx[8]={-1,0,1,0,-1,-1,1,1},dy[8]=
{0,1,0,-1,-1,1,1,-1};
typedef pair<ll,ll> P;

void Main() {
  ll T;
  R T;
  while(T--) {
    ll n;
    R n;
    ll a[n];
    rep(i,n) R a[i];
    ll sum=0;
    rep(i,n) sum+=a[i];
    vector<P> ans;
    for(ll i=0; i<n; i++) {
      if(i+1<n&&sum<0) {
        if(a[i+1]==-1) {
          ans.pb(P(i+1,i+2));
          sum+=2;
          i++;
        } else ans.pb(P(i+1,i+1));
      } else if(i+1<n&&sum>0) {
        if(a[i+1]==1) {
          ans.pb(P(i+1,i+2));
          sum-=2;
          i++;
        } else ans.pb(P(i+1,i+1));
      } else ans.pb(P(i+1,i+1));
    }
    if(!sum) {
      pr(ans.size());
      rep(i,ans.size()) pr(ans[i].F,ans[i].S);
    } else pr(-1);
  }
}

int main(){ios::sync_with_stdio(0);cin.tie(0);Main();return 0;}
```

solve

- 如果是奇数；
  - sum肯定为奇数。
  - 分块不会改变奇偶性。所以和必定为奇数。
- 如果是偶数。
  - 不妨先两两分块。
  - 发现总存在几种情况
    - 两两相等，合成一块。
    - 两两不等，独立，相加也为0
  - 最终构造出来的结果就是为0；

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

int a[maxn];

void solve()
{
    int n;
    cin >> n;
    vector<pair<int, int>> ans;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    if (n & 1)
    {
        cout << -1 << '\n';
        return;
    }
    for (int i = 2; i <= n; i += 2)
    {
        if (a[i] == a[i - 1])
            ans.push_back({i - 1, i});
        else
            ans.push_back({i - 1, i - 1}), ans.push_back({i, i});
    }
    cout << ans.size() << '\n';
    for (auto i : ans)
        cout << i.first << ' ' << i.second << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int t;
    cin >> t;
```

```
    while (t--)
        solve();
}
```

C2 hard verson

[Make Nonzero Sum (hard version)]

> 与上问题不一样的是： 数组中的元素，多了0的可能：

20min

> 和上面的一样，影响奇偶性的是1，-1的个数。如果是奇数个，不可能构造出一个合法的解。
> 如果是偶数个，按照相同类型的思路：
>
> 和上面的相似性的思路。
> 将两两之间也分成同一块：
>
> 每一组非零值中间或者两边必然有若干个0；
> 然后，
> 整体上是奇数个，
> 两两相同，分开成，一个奇数一个偶数。作为尾部。
> 两两不同，就直接分开都作单独处理。
> 整体上是偶数个
> 两两相同，
> 直接合为一块。两两不同，一个一个一个奇数一个偶数。

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

int a[maxn];

void solve()
{
    int n;
    cin >> n;
    int sum = 0;
    vector<int> pos;
    vector<pair<int, int>> ans;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        if (a[i] != 0)
            sum++, pos.push_back(i);
    }
    if (sum & 1)
    {
```

```
        cout << -1 << '\n';
        return;
    }
    if (sum == 0)
    {
        cout << 1 << '\n';
        cout << 1 << ' ' << n << '\n';
        return;
    }
    if (pos[0] != 1)
        ans.push_back({1, pos[0] - 1});
    for (int i = 1, size = pos.size(); i < size; i += 2)
    {
        if (a[pos[i]] == a[pos[i - 1]]) //两者相等;
        {
            if ((pos[i] - pos[i - 1] + 1) & 1)
                ans.push_back({pos[i - 1], pos[i - 1]}),
ans.push_back({pos[i - 1] + 1, pos[i]});
            else
                ans.push_back({pos[i - 1], pos[i]});
        }
        else //两者不同
        {
            if ((pos[i] - pos[i - 1] + 1) & 1) //
            {
                ans.push_back({pos[i - 1], pos[i] - 1});
                ans.push_back({pos[i], pos[i]});
            }
            else //此时是偶数。
            {
                ans.push_back({pos[i - 1], pos[i - 1]});
                if (pos[i] - 1 != pos[i - 1])
                    ans.push_back({pos[i - 1] + 1, pos[i] - 1});
                ans.push_back({pos[i], pos[i]});
            }
        }
        if (i != size - 1 && pos[i] + 1 != pos[i + 1])
            ans.push_back({pos[i] + 1, pos[i + 1] - 1});
    }
    if (pos[pos.size() - 1] != n)
        ans.push_back({pos[pos.size() - 1] + 1, n});
    cout << ans.size() << '\n';
    for (auto i : ans)
        cout << i.first << ' ' << i.second << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
```

```
    int t;
    cin >> t;
    while (t--)
        solve();
}
```