题解

1.drink

我们可以枚举 1 出现的开始和结尾下标，那么这个区间的长度就是最少拿起瓶子数量，时间复杂度 $O(n)$

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5+10;
int a[N];
int main(){
    int n;
    cin >> n;
    int sum = 0;
    for (int i = 1; i <= n; i ++){
        cin >> a[i];
        if(a[i]) sum ++;
    }
    int fi = -1, last = -1;//fi 表示开始的下标，last 表示最后的下标
    for (int i = 1; i <= n; i ++){
        if(a[i] == 1 && fi == -1){
            fi = i;
            last = i;
        } else if(a[i] == 1 && fi != -1){
            last = i;
        }
    }
    cout << last - fi + 1;

    return 0;
}
```

2.Creep

"使它的 0 和 1 的个数相差最多的前缀，0 和 1 的个数相差最少"。意思是：对于这个字符串，考虑所有前缀，找到那个前缀中 $|count0 - count1|$ 最大的那个值，然后这个最大值应该尽可能小。也就是说，最小化所有前缀中 $|count0 - count1|$ 的最大值。

- 此处的 $count0$ 指的是前缀中 0 的数量，$count1$同理
- 前缀指的是从序列或数组的 起始位置 到 某个特定位置 这一段连续的元素
  如：`ABCDE`
  则`A`，`AB`，`ABC`，`ABCD`，`ABCDE`都为前缀，只是结束位置不同

为了前缀最小，即 前面的 1 和 0 的数量最好要相等，那么我们可以先匹配 1 和 0 组成一对最多有多少对，1和 0 组成配对能使得前缀$|count0 - count1| = 0$，剩下的 无法配对的 1 或 0 直接输出在后面即可

```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
    int t;
    cin >> t;
    while(t --){
        int a, b;
        cin >> a >> b;
        if(a < b){
            for (int i = 1; i <= a; i ++){
                cout << "10";
            }
            for (int i = 1; i <= b-a; i ++) cout << "1";
        }else{
            for (int i = 1; i <= b; i ++){
                cout << "10";
            }
            for (int i = 1; i <= a-b; i ++) cout << "0";
        }
        cout << endl;
    }
    return 0;
}
```

3.离开中山路

BFS 模板题，时间复杂度 $O(n^2)$

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1010;
int  x1, yl,x2,y2;
string a[N];
struct node{
    int x, y;
    long long d;
};
int n;
int dx[4] = {1,0,-1,0}, dy[4] = {0,1,0,-1};
bool vis[N][N];
bool in(int x, int y){
    return x >=0 && x<= n && y >= 0 && y <= n;
}
long long bfs(int bx, int by){
    queue<node> q;
    q.push({bx,by,0});
    vis[bx][by] = true;
    while(!q.empty()){
        node t = q.front();
```

```
        q.pop();
        if(t.x == x2 && t.y == y2){
            return t.d;
        }
        for (int i = 0; i < 4; i ++){
            int tx = t.x + dx[i], ty = t.y + dy[i];
            if(in(tx,ty) && !vis[tx][ty]){
                vis[tx][ty] = true;
                q.push({tx,ty,t.d+1});
            }
        }
    }
    return -1;
}
int main(){
    cin >> n;
    for (int i = 0; i < n; i ++){
        cin >> a[i];
    }
    cin >> x1 >> yl >> x2 >> y2;
    x1--,yl--,x2--,y2--;
    cout << bfs(x1,yl);
    return 0;
}
```

4.膜拜

观察题目，最值问题有很多种方法，这道题明显适合使用动态规划。

为了方便统计我们两边的人数相差不超过 m，我们设置值为 2 的同学为 -1 ，使用前缀和保证 i ~ j 中的学生中，$|\ sum[i]-sum[j-1])\ | \leq m$ 或者 $|\ sum[i]-sum[j-1]\ | = i-j+1$ 即可，

那么如何去定义我们的状态，题目中说每次都去一段放到不同的机房，那么这道题相到于将长度为 n 的数列分解成不同的一段，每一段分到 两个不同的集合当中。同时当时有多少个人 i 个人是影响我们最终的结果的，同时我们是需要从前面一段的情况转移过来。

所以我们设定 $dp[i]$ 为当前以第 i 个人作为结尾所需的机房个数，$dp[j](1 \leq j \leq i)$ 为 上一个以第 j 位同学作为结尾的子段。

为了方便统计我们两边的人数相差不超过 m，我们设置值为 2 的同学为 -1 ，使用前缀和保证 i ~ j 中的学生中，$|\ sum[i]-sum[j-1])\ | \leq m$ 或者 $|\ sum[i]-sum[j-1]\ | = i-j+1$ 即可，

我们的状态转移方程为：

```
for (int j = 1~ i-1)
  if(abs(sum[i] - sum[j] <= m))
dp[i] = min(dp[j]) + 1 ( 1 <= j <= i-1)
```

同时初始化，一个人都没有的时候，最多需要 0 个机房 $dp[0] = 0$

```
memset(dp,0x3f,sizeof dp);
dp[0] = 0;
```

时间复杂度为：$O(n^2)$

代码：

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2510;
int A[N], sum[N], dp[N];
int main(){
    int n, m;
    cin >> n >> m;
    memset(dp,0x3f,sizeof dp);
    for (int i = 1; i <= n; i ++){
        cin >> A[i];
        if(A[i] == 2) A[i] = -1;
        sum[i] = sum[i-1] + A[i];
    }
    dp[0] = 0;
    for (int i = 1; i <= n; i ++){
        for (int j = 1; j <= i; j ++){
            if(abs(sum[i] - sum[j-1]) <= m || abs(sum[i] - sum[j-1]) == i - j + 1){
                dp[i] = min(dp[i],dp[j-1]+1);
            }
        }
    }
    cout << dp[n];
    return 0;
}
```

5.调味平衡

要求食物的两种味道之和一样，我们可以分为三种情况。酸度比甜度高，酸度和甜度相等，酸度比甜度低。如果酸度比甜度高的值之和和酸度比甜度高之和相等，那就可以变成一种方案。

所以我们可以用动态规划来解决。

1. **状态定义**：`dp[i][diff]` 表示考虑前i种食材，酸度与甜度差值为diff时的最大总和
2. **状态转移**：
    - 不选第i种食材：`dp[i][diff] = dp[i-1][diff]`
    - 选第i种食材：`dp[i][diff + a[i] - b[i]] = dp[i-1][diff] + a[i] + b[i]`
3. **答案**：`dp[n][0]` 即为答案

时间复杂度：$O(n \times S)$

代码：

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

const int MAXN = 105;
const int MAXS = 100005;
const int OFFSET = 50000;

int n;
int a[MAXN], b[MAXN];
int dp[MAXN][MAXS];

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i] >> b[i];
    }

    memset(dp, -1, sizeof(dp));
    dp[0][OFFSET] = 0;

    for (int i = 1; i <= n; i++) {
        for (int diff = 0; diff < MAXS; diff++) {
            if (dp[i-1][diff] == -1) continue;

            dp[i][diff] = max(dp[i][diff], dp[i-1][diff]);

            int new_diff = diff + a[i] - b[i];
            if (new_diff >= 0 && new_diff < MAXS) {
                dp[i][new_diff] = max(dp[i][new_diff], dp[i-1][diff] + a[i] + b[i]);
            }
        }
    }

    cout << dp[n][OFFSET] << endl;
    return 0;
}
```