# 1-4 完美，近乎完美

## problem

给定一个大小为 $n$ $(2 \leq n \leq 2 \times 10^5)$ 的可重集合，询问删去集合中的哪些元素后，集合中存在一个元素等于其余所有元素的和。集合中元素有值域 $[1, 10^6]$。

## solution

集合中存在一个元素等于其他元素的和，当且仅当该元素的大小等于集合中所有元素的和的 $\frac{1}{2}$。于是可以处理出

1. 集合中所有元素的和 $s$；
2. 值域中每个数字在集合中出现的次数和位置。

枚举每个元素 $x$，判断删去该元素后是否仍存在某个元素等于 $s - x$ 即可。

时间复杂度 $O(n)$。

## code

注意需要 C++17。

`s: int64` 即前述的 $s$，`v[i]: std::vector<int>` 记录 $i$ 出现的所有位置。

```
int z[200003];
std::vector<int> ans, v[1000003];

int main() {
  int N;
```

```
  int64_t s = 0;

  std::cin >> N;
  for (int i = 1; i ⩽ N; ++i) {
    std::cin >> z[i];
    s += z[i];
    // 这里判一下 if (v[z[i]].size > 2) 则不 push 也是可以的，但没必要
    v[z[i]].emplace_back(i);
  }

  for (int i = 1; i ⩽ N; ++i)
    if (int64_t x = s - z[i]; x ⩽ 2000000 && x % 2 == 0)
      if (v[x / 2].size() > 1 || (v[x / 2].size() == 1 && v[x / 2][0] ≠ i))
        ans.emplace_back(i);

  std::cout << ans.size() << '\n';
  if (ans.size())
    for (int x : ans)
      std::cout << x << ' ';
}
```

---

## 1-5 回文，排版错误

### problem

给定 $n\,(1 \le n \le 100)$ 个长度均为 $m\,(1 \le m \le 50)$ 的字符串 $s_i$。判断这些字符串拼接而成的最长的回文串 $P$ 的长度。字符集小写拉丁字母。字符串不能重复使用。

### solution

所有字符串长度都相等实在是很强的约束条件，不然这题就太毒了。

考虑一个字符串 $s$ 能够参与拼接进 $P$，当且仅当 $s$

1. 本身是回文串，或
2. 给定了另一个字符串 $t$ 使得 $s+t$ 是回文串（此处 $+$ 表示字符串的拼接操作，显然这时 $t+s$ 也是回文串）

满足性质 1 的串 $s$ 可以置于 $P$ 的中心，而满足性质 2 的串 $s,t$ 可以对称地置于 $P$ 中心的两侧。为了最大化 $|P|$，当一个串 $s$ 同时满足两个性质时，优先按照性质 2 参与拼接。并且可以发现，至多只能有一个只满足性质 1 的串参与构造 $P$。

于是可以预处理出每个字符串是否满足以上两个性质，并据此构造 $P$，实现细节见代码。总复杂度 $O(n^2m)$。

### code

注意需要 C++11。

`v1[i]: bool` 即字符串 $s_i$ 满足性质 1 与否；`v2[i]` 中的所有元素 $x$ 表示 $s_i, s_x$ 满足性质 2。

```
inline bool isPalin(const std::string &s) {
  /*
   * Returns if $s is a palindrome
   */
  std::string t(s);
  std::reverse(t.begin(), t.end());
  return t == s;
```

```
}

std::string s[103];
std::vector<int> v2[103];
bool v1[103], used[103];

int main() {
  int N, M;

  std::cin >> N >> M;
  for (int i = 1; i <= N; ++i)
    std::cin >> s[i];

  for (int i = 1; i <= N; ++i) {
    v1[i] = isPalin(s[i]);
    for (int j = i + 1; j <= N; ++j)
      if (isPalin(s[i] + s[j])) {
        v2[i].emplace_back(j);
        v2[j].emplace_back(i);
      }
  }

  std::string ans1, ans2;
  // 性质 2
  for (int i = 1; i <= N; ++i)
    if (v2[i].size())
      for (int j : v2[i])
        if (!used[j]) {
          ans1 += s[i];
          ans2 = s[j] + ans2;
          used[i] = used[j] = 1;
          break;
        }
  // 性质 1
  for (int i = 1; i <= N; ++i)
    if (v1[i] && !used[i]) {
      ans1 += s[i];
      break;
    }

  std::string ans = ans1 + ans2;
  std::cout << ans.size() << '\n';
  if (ans.size())
    std::cout << ans << '\n';
}
```

---

## 1-9 睁眼，便是终局 & 2-7 睁眼，便是终局

### problem

给定长为 $n$ 的数列 $\{a_i\}$ $(1 \le a_i \le 10^9)$，在其中选中 $x$ 个数字，且连续 $k$ 个数字中至少有一个被选中，最大化选中的所有数字的和。

Constraint of Easy Version(1-9): $1 \le k, x, n \le 200$;

Constraint of Hard Version(2-7): $1 \le k, x, n \le 5000$;

## solution for Easy Version

经典 dp（

设 $f_{i,j}$ 表示前 $i$ 个数字中选中了 $j$ 个数字，且第 $i$ 个数字被选中的答案。有显然的转移：

$$f_{i,j} = \max_{k'=i-k+1}^{i-1} f_{k',j-1} + a_i$$

边界条件 $f_{i,j} = -\infty, f_{0,0} = 0$。

时间复杂度 $O(nxk)$；空间复杂度 $O(nx)$，可以压一维到 $O(n)$，不过没必要，毕竟时间复杂度摆在那。

## code for Easy Version

`f[i][j]` 即前述的 $f_{i,j}$。这里用了 $-1$ 标记 $-\infty$，这样恰好把无解输出 `-1` 处理掉了，不用单独再判。注意一下写法问题别越界。

```
int z[203];
int64_t f[203][203];

int main() {
  int N, K, X;

  std::cin >> N >> K >> X;
  for (int i = 1; i ⩽ N; ++i)
    std::cin >> z[i];

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

  for (int i = 1; i ⩽ N; ++i)
    for (int j = 1; j ⩽ i; ++j) // 这里 j ⩽ min(i, X) 就可以
      for (int k = i - 1; k ⩾ std::max(0, i - K); --k) // 这里正着或反着枚举是无所谓的
        if (f[k][j - 1] ≠ -1)
          f[i][j] = std::max(f[i][j], f[k][j - 1] + z[i]);

  int64_t ans = -1;
  for (int i = N - K + 1; i ⩽ N; ++i)
    ans = std::max(ans, f[i][X]);
  std::cout << ans << '\n';
}
```

## solution for Hard Version

我这个状态数只有 $O(nx)$ 个啊，干啥要 $O(nxk)$ 的复杂度呢？

如果固定一个 $j$，枚举每个 $i$，就会发现枚举转移的时候内层的两维（对于每个 $i$ 枚举 $k'$ 求 $\max f_{k',j-1}$）和 $i$ 无关，非常的滑动窗口，于是单调队列优化转移，时间复杂度降到了 $O(nx)$。

空间也可以压到 $O(n)$，但是同上没有必要。

参考题目：洛谷 P1886 滑动窗口 /【模板】单调队列

### code for Hard Version

`f[i][j]` 即前述的 $f_{j,i}$。注意两维在思考优化的过程中反过来了，于是代码里顺手也反过来写了。

这里用了 $-1$ 标记 $-\infty$，这样恰好把无解输出 `-1` 处理掉了，不用单独再判。

```
int z[5003];
int64_t f[5003][5003];

int main() {
  int N, K, X;

  std::cin >> N >> K >> X;
  for (int i = 1; i ⩽ N; ++i)
    std::cin >> z[i];

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

  for (int j = 1; j ⩽ X; ++j) {
    std::deque<std::pair<int, int64_t>> q;
    for (int i = 1, now = std::max(0, i - K); i ⩽ N; ++i) {
      // k \in [max(0, i - K), i - 1]
      while (q.size() && q.front().first < i - K)
        q.pop_front();

      for (; now < i; ++now)
        if (f[j - 1][now] ≠ -1) {
          while (q.size() && q.back().second ⩽ f[j - 1][now])
            q.pop_back();
          q.push_back({now, f[j - 1][now]});
        }

      if (q.size())
        f[j][i] = std::max(f[j][i], q.front().second + z[i]);
    }
  }

  std::cout << *std::max_element(f[X] + N - K + 1, f[X] + N + 1) << '\n';
}
```

---

## 1-10 尾声，抑或开始 & 2-8 残页，封存往事

### problem

给定一个长为 $n\ (1 \le n \le 2 \times 10^5)$ 的序列 $\{a_i\}\ (1 \le a_i \le 2 \times 10^5)$ 和一个值 $m\ (1 \le m \le 2 \times 10^5)$。询问序列中有多少子序列（连续）的**中位数**是 $m$。

此处**中位数**的定义和数学上略有不同。区别在于对于长度为偶数 $2k$ 的子序列，定义中位数为第 $k$ 小的数。例如序列 $[1, 1, 4, 5, 1, 4]$ 的中位数为第 $3$ 小的 $1$，而非数学上的 $\frac{1+4}{2} = \frac{5}{2}$。

Extra Constraint for Easy Version: 保证序列 $\{a_i\}$ 为整数 $1 \sim n$ 的一个排列。

## solution for Easy Version

感性理解一下就会发现，我们其实只需要每个数之间的偏序关系，而不需要它们具体的值。

Easy Version 保证了序列是一个排列，也就是说，所有的值都是两两不同的。这时我们可以套路性地将原排列中小于 $m$ 的数、$m$、大于 $m$ 的数分别重新映射为 $-1, 0, 1$。然后从原排列的 $m$ 的位置开始，向左做后缀和，并记录每个后缀和 $s$ 出现的次数 $c_s$。然后再向右做前缀和，这样，当枚举到下标 $i$ 时，前缀和为 $s$ 时，$c_{-s}$ 和 $c_{1-s}$ 即为以位置 $i$ 为右端点的符合条件的区间的个数，且两者分别为区间长度为奇数、偶数时的个数。

这个做法非常容易想，但是只看文字描述是很难以理解的。这里以样例 1 的 reverse（maybe it should be called reversal?）作为**例子**：

- 我们有 $1 \sim 5$ 的一个排列 $[1, 3, 5, 4, 2]$。
- 从 $m = 4$ 所在的下标 $4$ 向左做后缀和，得到 $[-1, 0, 1, 0, \mathrm{NaN}]$。
- 于是我们得到 $c_{-1} = 1, c_0 = 2, c_1 = 1$，对于其余的 $s$ 值，$c_s$ 均为 $0$。
- 接着我们从下标 $4$ 向右做前缀和。在下标 $4$ 处 $s = 0$，此时 $c_{-s} = c_0 = 2, c_{1-s} = c_1 = 1$。在下标 $5$ 处 $s = -1$，此时 $c_{-s} = c_1 = 1, c_{1-s} = c_2 = 0$。
- 把上述提到的值全部相加得到答案 $4$。（感性理解一下样例 1 的 reverse 和样例 1 的答案肯定是相同的）

时空复杂度均为 $O(n)$。

## code for Easy Version

变量名和上述除大小写外完全相同。

这里用了很骚的写法，给下标传了负数。注意实际上 `c` 是 `int *` 而不是 `int[]`，`c[-s]` 相当于 `*(c + (-s))`，只要没有发生越界就没有问题。**向数组名的 `operator[]` 中传递负下标是未定义行为。**

```cpp
int z[200003], c_[400007], *c = c_ + 200003;

int main() {
  int N, M, p = 0;

  std::cin >> N >> M;
  for (int i = 1, x; i <= N; ++i) {
    std::cin >> x;
    if (x == M) p = i;
    else if (x < M) z[i] = -1;
    else z[i] = 1;
  }

  for (int i = p, s = 0; i; --i) {
    s += z[i];
    ++c[s];
  }

  int64_t ans = 0;
  for (int i = p, s = 0; i <= N; ++i) {
    s += z[i];
    ans += c[-s];
    ans += c[1 - s];
  }
  std::cout << ans << '\n';
}
```

## solution for Hard Version

很不幸，如果按照上面的思考方法，那么这题的 Hard Version 将会变得非常难想。

首先提一下 Easy Version 的上述做法在这里是错的：其一，Hard version 可能存在不止一个 $m$，因此上述做法变得难以扩展；其二，即使强行找到方法 $O(n)$ 地扩展，其赖以正确的约束条件「$\{a_n\}$ 中的每两个数都存在严格的偏序关系」不成立，它的正确性会出问题。这里不作更多说明。

Hard Version 用到了另一个思想：把求值转化为求前缀和的差分。这句话看起来非常的蠢，但是我们尝试把它应用到这道题上。

求中位数等于 $m$ 的区间个数，即求**中位数大于等于 $m$ 的区间个数**，减去**中位数大于 $m$ 的区间个数**。由于 $\{a_n\}$ 及其所有子区间的中位数只有整数，后者等价于中位数大于等于 $m+1$ 的区间个数。

现在问题转化为了如何求**中位数大于等于 $m$ 的区间个数**。对于一个区间，我们设其中小于 $m$ 的数、$m$、大于 $m$ 的数分别有 $x,y,z$ 个，则该区间的中位数大于等于 $m$ 等价于 $x<y+z$。这个可以通过大力分类讨论 $x,y,z$ 的值关系得到，此处略去不表。

我们同样把小于 $m$ 的数、大于等于 $m$ 的数分别记作 $-1,1$，同时统计此意义下前 $i$ 个数的前缀和 $s$、每个前缀和 $s_0$ 出现的次数 $c_{s_0}$ 以及 $c_{[0,s)}$ 的前缀和 $ss$。注意 $ss$ 的意义随着 $s$ 值的改变而改变。

对于枚举过程中每一个 $s$ 的取值，可以将其看作前一段提到的 $y+z$ 的值，此时 $ss$ 的值即为满足 $x<y+z$ 的 $x$ 的个数。类似于 Easy Version 里对每个 $c_{-s}+c_{1-s}$ 求和的想法，我们对所有的 $ss$ 求和，即可得到答案。

统计 $ss$ 既可以像下述代码里一样单次 $O(1)$ 地动态统计，也可以写个数据结构单次 $O(\log k)$ 地统计。总复杂度 $O(n)$ 或 $O(n\log k)$。其中 $k$ 为 $a_i$ 的值域。

## code for Hard Version

变量名和上述除大小写外完全相同。

```cpp
int z[200003], c_[400007], *c = c_ + 200003;

int64_t solve(int N, int M) {
  /*
   * Returns the number of subsequences(continuous) with a median of ge M
   */
  static bool g = 0;
  if (g) memset(c_, 0, sizeof c_);
  g = 1;

  // ss: sum of c[0..s)
  int64_t ss = 0, ans = 0;
  ++c[0];
  for (int i = 1, s = 0; i <= N; ++i) {
    // 下面这两行太有感觉了，简直青回（
    if (z[i] < M) ss -= c[--s];
    else ss += c[s++];
    ans += ss;
    ++c[s];
  }
  return ans;
}

int main() {
  int N, M;

  std::cin >> N >> M;
```

```
  for (int i = 1; i ⩽ N; ++i)
    std::cin >> z[i];

  std::cout << solve(N, M) - solve(N, M + 1) << '\n';
}
```

## 2-9 猎手，抑或猎物

### problem

给定一个 $n \times m$ $(3 \leq n \times m \leq 10^6)$ 的迷宫，有些位置可以通过，有些不能。你可以将一些可以通过的位置改变为不能通过。求最少改变多少个位置的性质，使得不存在从 $(1,1)$ 通向 $(n,m)$ 的路径。移动只能向右或向下。

### solution

嗷这题是非常的重量级，一眼看到这题以为要建 $O(nm)$ 个点和 $O(nm)$ 条边跑网络流，一看数据范围傻眼了😭

发现答案只可能有 $0,1,2$ 三种取值，因为至多把起点旁边的俩格子堵上就完事了。现在的任务就是分出这三种情况来。

首先答案是 $0$ 的情况非常的好判，dfs 一遍到不了终点答案就是 $0$。

那么答案是 $1$ 和 $2$ 怎么区分呢，这个也很简单，dfs 第一遍的时候把经过的路都堵上，判能不能 dfs 到终点第二遍就行了。dfs 的时候先沿同一个方向走，直到发现死路再转向，正确性不会证，感性理解一下（

这个做法和一般图上找两条路径网络流第一遍不退流有异曲同工之妙，所以这题还是没能和网络流脱开干系（确信）

复杂度显然是 $O(nm)$。

### code

```
constexpr int dx[] = {0, 1}, dy[] = {1, 0};

bool dfs(int x, int y) {
  if (x == N && y == M) return 1;
  z[x][y] = '#';
  for (int k = 0, nx, ny; k < 2; ++k) {
    nx = x + dx[k], ny = y + dy[k];
    if (nx ⩽ N && ny ⩽ M && z[nx][ny] == '.')
      if (dfs(nx, ny))
        return 1;
  }
  return 0;
}

int main() {
  int N, M;

  std::cin >> N >> M;
  std::vector<std::string> z(N + 1);
  for (int i = 1; i ⩽ N; ++i) {
    std::cin >> z[i];
    z[i] = " " + z[i];
```

```
  }

  int ans1 = dfs(1, 1);
  if (!ans1) {
    std::cout << "0\n";
    return 0;
  }

  z[1][1] = z[N][M] = '.';
  int ans2 = dfs(1, 1);
  std::cout << 1 + ans2 << '\n';
}
```

---

Fin.