TỔ CHỨC THE GIFTED BATTLEFIELD
DỰ ÁN INTELLECTUAL SHOWDOWN

TGB CONTEST #1
NĂM HỌC 2023 – 2024

Ngày thi: **17/09/2023**
Thời gian làm bài: **180 phút** (không kể thời gian phát đề)
Hướng dẫn giải gồm 11 trang, 06 bài

Tổng quan đề thi

	Tên bài	Hạn chế thời gian	Hạn chế bộ nhớ
Bài 1	Đếm bit	2 giây	256 MB
Bài 2	Sắp xếp sách	1 giây	256 MB
Bài 3	Tỉa cây	2 giây	256 MB
Bài 4	CSC	3 giây	512 MB
Bài 5	Jelly Bear	2 giây	256 MB
Bài 6	Decrease XOR	15 giây	256 MB

Nhập xuất theo kiểu chuẩn - standardIO, không sử dụng nhập xuất file.

I. Hướng dẫn chung

1. Bài thi của thí sinh được chấm thi bằng phần mềm chấm thi trực tuyến (online judge) trên website `https://oj.giftedbat.edu.vn/`, sử dụng bộ test của Tổ chức The Gifted Battlefield – Ban Tin học, đúng với đáp án và biểu điểm được nêu trong hướng dẫn chấm thi.
2. Điểm bài thi được xuất từ phần mềm chấm thi.

Bài 1. Đếm bit (100 điểm)

Tổng quan

- Đây là một bài toán đề nói gì làm nấy. .
- Với mỗi bit $i (0 \leq i \leq 29)$, ta sẽ dùng mảng đếm để lưu lại số lần bit i được bật ở các phần tử trong mảng A. Sau đó, ta chỉ cần đi qua từng vị trí bit (từ 0 đến 29) để đếm số vị trí bit có tần suất xuất hiện là lẻ.
- **Độ phức tạp :** $O(n * \log n)$**.**

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

int n;
int cnt[30];

int main() {
  cin >> n;
  for (int i = 1; i <= n; i++) {
    int a;
    cin >> a;
    for (int j = 0; j < 30; j++) {
      if ((a >> j) & 1) cnt[j]++; //nếu bit thứ j của số a[i] được bật
    }
  }

```

```
  int res = 0;
  for (int i = 0; i < 30; i++) res += (cnt[i] % 2);
  cout << res;
  return 0;
}
```

- **Thử thách : Bạn có thể tối ưu code xuống $O(n)$ không?**

Bài 2. Sắp xếp sách (200 điểm)

Ý tưởng

Gọi cnt_x là tổng số cuốn sách của môn x nằm trên kệ và trên bàn. Sort các giá trị cnt_x theo thứ tự từ lớn đến bé, lần lượt lấy các bộ môn có số sách nhiều nhất cho đến khi số sách đang có đạt ít nhất N.

Giải thích

- Ta nhận xét rằng điều kiện "không được đặt lên kệ những cuốn sách đã lấy xuống trước đó" không ảnh hưởng đến kết quả bài toán. Vì thế, các cuốn sách trên kệ và trên bàn luôn có thể hoán đổi vị trí cho nhau. Khi đó, cách sắp xếp sách tối ưu sẽ là tham lam lấy nhiều cuốn sách cùng 1 bộ môn nhất có thể để giảm thiểu số bộ môn xuất hiện trên kệ sách.
- Để đếm số sách mỗi bộ môn, ta có thể sử dụng cấu trúc dữ liệu map<int, int>, hoặc normalize các giá trị về khoảng từ 1 đến $N + M$ (vì chỉ có tối đa $N + M$ giá trị phân biệt).

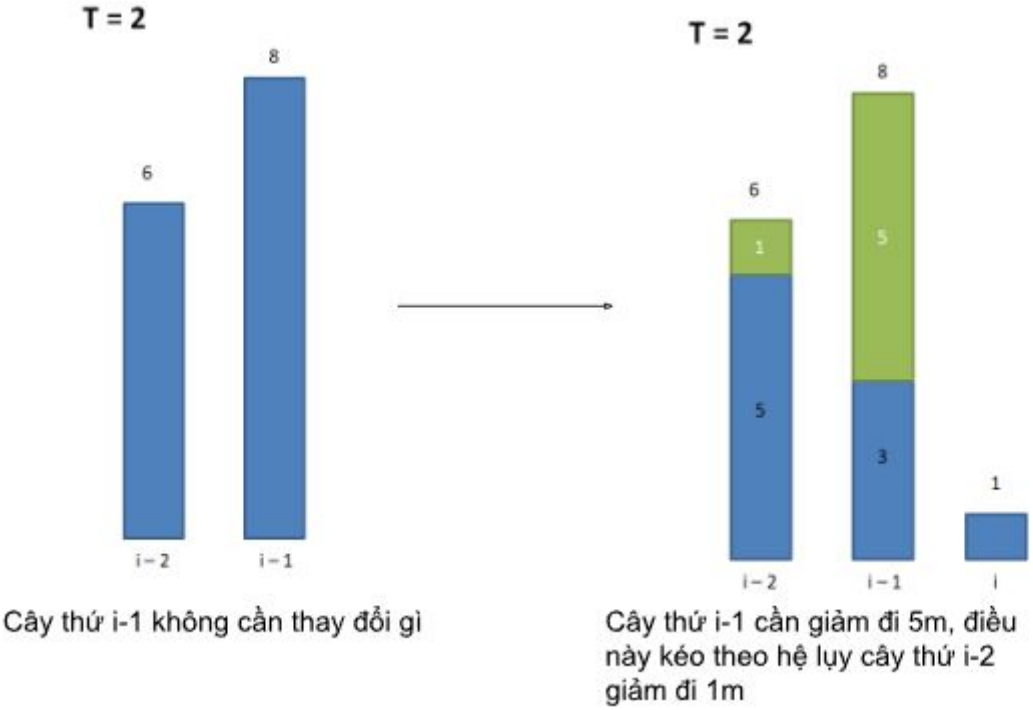
```
#include <bits/stdc++.h>

using namespace std;

const int N = 2e5;

int n, m;
int a[N+5], b[N+5];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n >> m;

    map<int, int> cnt;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        ++cnt[a[i]];
    }

    for (int i = 1; i <= m; i++) {
        cin >> b[i];
        ++cnt[b[i]];
    }

    vector<int> val;

    for (auto [u, v] : cnt) val.push_back(v);

    sort(val.begin(), val.end());

    int res = 0, sum = 0;
    for (int i = val.size() - 1; i >= 0; i--) {
        res++;
        sum += val[i];
        if (sum >= n) break;
    }

    cout << res;
```

```
41      return 0;
42  }
```

Bài 3. Tỉa cây (300 điểm)

Ý tưởng

- Ta sẽ dùng thuật toán **Tham Lam** để giải bài toán này.
- Với một **độ xấu** T cho trước, ta sẽ tìm chi phí thấp nhất để điều chỉnh chiều cao của hàng cây sao cho chênh lệch độ cao giữa 2 cây liên tiếp không quá T ($|a_i - a_{i-1}| \leq T,\ \forall i : 2 \leq i \leq n$):
  - **Bước 1:** Duyệt qua tất cả các cây 1 lần để điều chỉnh lại chiều cao các cây có chiều cao lớn hơn T so với cây đứng trước nó. Nói cách khác, với mỗi i sao cho $2 \leq i \leq n$, nếu $a_i - a_{i-1} > T$ thì ta sẽ giảm a_i xuống thành $a_{i-1} + T$ và tính chi phí cần để thực hiện thao tác trên. Tuy nhiên, lúc này, khi xét tới cây thứ i, sẽ có trường hợp mà ta cần phải giảm cây thứ $i-1$ (đồng nghĩa phải điều chỉnh chiều cao của các cây trước đó 1 lần nữa), như hình dưới:



  - **Bước 2:** Để giải quyết vấn đề trên, ta duyệt thêm 1 lần tất cả các cây, theo chiều từ $n \to 1$. Với mỗi i ($n - 1 \geq i \geq 1$), nếu $a_i - a_{i+1} > T$, ta sẽ điều chỉnh a_i và tính chi phí cho thao tác này, tương tự như bước 1.
  - Để chứng minh chỉ cần thực hiện 2 bước trên, ta sẽ đảm bảo chênh lệch chiều cao của các cây luôn bé hơn hoặc bằng T cho trước, ta có các nhận xét sau:
    + Sau lần duyệt đầu, ta dễ dàng thấy rằng với 2 cây bất kỳ liên tiếp nhau, nếu cây thứ i cao hơn cây thứ $i-1$ thì độ chênh lệch giữa 2 cây luôn không quá T.
    + Ở lần thứ 2 sau khi duyệt ngược về, ta đảm bảo được nếu cây thứ i cao hơn cây thứ $i+1$ thì chênh lệch chiều cao giữa 2 cây luôn không quá T; đồng thời, các cặp đã thỏa ở lần duyệt đầu tiên cũng sẽ không bị ảnh hưởng.
- Ngoài ra, để tối ưu hóa bài toán, ta sẽ sử dụng thuật toán Tìm kiếm nhị phân (Binary search) để chặt nhị phân độ xấu của cây. Với mỗi lần chặt nhị phân, ta kiểm tra với độ xấu T, có thể dùng 1 số tiền $\leq Q$ để chặt các cây không.

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N = 2e5;

int n, p, q, c;
int a[N + 5], h[N + 5];

bool check(long long val) {
    //Copy các giá trị từ đề ra mảng riêng vì trong hàm có thay đổi mảng
    for (int i = 1; i <= n; i++) h[i] = a[i];

    int cost = 0;
    //Duyệt bước 1: từ 2 -> n
    for (int i = 2; i <= n; i++) {
        if (h[i] - h[i - 1] > val) {
            cost += h[i] - (h[i - 1] + val);
            h[i] = h[i - 1] + val;
        }

        if (cost > c) return false;
    }

    //Duyệt bước 2: từ n - 1 -> 1
    for (int i = n - 1; i >= 1; i--) {
        if (h[i] - h[i + 1] > val) {
            cost += h[i] - (h[i + 1] + val);
            h[i] = h[i + 1] + val;
        }

        if (cost > c) return false;
    }

    return true;
}

long long binSearch() {
    long long l = 0, r = 1e18, res = -1;
    while (l <= r) {
        long long mid = (l + r) >> 1;
        if (check(mid)) {
            r = mid - 1;
            res = mid;
        } else {
            l = mid + 1;
        }
    }

    return res;
}

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

    c = q/p;
    cout << binSearch();

    return 0;
}
```

Bài 4. CSC (400 điểm)

Note

- Bài này có một edge case khi $l = r$, lúc này đáp án luôn là YES.

Subtask 1: $n \leq 1000, q \leq 1000$

- Ở subtask này, với mỗi query, ta chỉ cần thực hiện đúng như yêu cầu của đề bài. Độ phức tạp: $O(q*n \log n)$, **lưu ý: ở đây giả sử bạn đọc sử dụng thuật toán sort với độ phức tạp là** $O(n \log n)$.

Subtask 2: $x \leq 1$ **với mọi truy vấn**

- Ở subtask này, ta nhận xét rằng chúng ta không thể nào thực hiện brute force như ở trên. Nhận thấy x chỉ có 2 giá trị 0 và 1. Ta hãy thử chia thành từng trường hợp để kiểm tra.
- Với $x = 0$: Lúc này, ta cần kiểm tra $u_i = u_{i-1} + 0 = u_{i-1}$. Nói cách khác, ta cần kiểm tra xem các phần tử từ l đến r có bằng nhau hay không. Để kiểm tra điều kiện này, ta có nhiều hướng tiếp cận. Tác giả lựa chọn cách tiếp cận kiểm tra: $max(a_l, a_{l+1}, \ldots, a_r) = min(a_l, a_{l+1}, \ldots, a_r)$. Để lấy max và min, ta có thể sử dụng các cấu trúc dữ liệu

  RMQ để truy vấn. Độ phức tạp sẽ là: $O(q)/O(q \log n)$ tuỳ vào cấu trúc dữ liệu sử dụng.
- Với $x = 1$: Lúc này, ta cần kiểm tra $u_i = u_{i-1} + 1$. Ta có nhận xét sau:
  - Gọi $max(a_l, a_{l+1}, \ldots, a_r) = Max, min(a_l, a_{l+1}, \ldots, a_r) = Min$.
  - Nếu mảng ta cần kiểm tra có độ dài n, ta luôn có biểu thức $Max - Min = n - 1$. Ta cũng sẽ sử dụng RMQ để kiểm tra biểu thức này.
  - Tuy nhiên, ta cũng phải kiểm tra xem các **phần tử phải khác nhau đôi một**. Ta có thể giải bài toán này offline.
  - Kết hợp hai điều kiện, ta sẽ giải được bài toán với $x = 1$.
  - Độ phức tạp: $O(q \log n)$.

Subtask 3: Không có ràng buộc gì thêm

- Lúc này, ta cần tổng quát hoá bài toán ở subtask thứ 2. Ta có thể chứng minh được, với mỗi mảng $u[]$ **được sắp xếp không giảm** có độ dài n, ta luôn có biểu thức sau: $u_n = u_1 + (n-1) * t$ với $t = u_2 - u_1 = u_3 - u_2 = \ldots$ Dễ thấy, điều kiện này có thể sử dụng RMQ để kiểm tra.
- Ngoài ra, gọi $g_i = gcd(u_i, u_{i-1}), 1 < i \leq n$. Ta cũng sẽ kiểm tra $gcd(g_{l+1}, g_{l+2}, \ldots, g_r)$ xem có bằng x hay không. Lý do: ta nhận xét được với các cặp phần tử trong mảng u, ta luôn có hiệu của chúng chia hết cho x.
- Các kết hợp các điều kiện vừa được mở rộng ở subtask 3 và những cái đã chứng minh ở subtask 2. Ta sẽ giải được bài toán này với độ phức tạp là $O(q \log n)$.

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

const int N = 4e5 + 5;

int mx[N][21], mn[N][21], g[N][21];
int n, q, a[N], b[N], bit[N];
vector<array<int, 3>> query[N];
bool ans[N];

int get_max(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return max(mx[l][k], mx[r - (1 << k) + 1][k]);
}

```

```cpp
int get_min(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return min(mn[l][k], mn[r - (1 << k) + 1][k]);
}

int get_gcd(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return __gcd(g[l][k], g[r - (1 << k) + 1][k]);
}

int get(int x) {
    int s = 0;
    for (; x; x -= x & -x) {
        s += bit[x];
    }
    return s;
}

void upd(int p, int x) {
    for (; p <= n; p += p & -p) {
        bit[p] += x;
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n >> q;

    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        query[i].clear();
        bit[i] = 0;
    }

    for (int i = 1; i < n; ++i) {
        b[i] = abs(a[i + 1] - a[i]);
    }

    for (int i = 1; i <= n; ++i) {
        mx[i][0] = a[i];
        mn[i][0] = a[i];
    }

    for (int j = 1; (1 << j) <= n; ++j) {
        for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
            mx[i][j] = max(mx[i][j - 1], mx[i + (1 << (j - 1))][j - 1]);
            mn[i][j] = min(mn[i][j - 1], mn[i + (1 << (j - 1))][j - 1]);
        }
    }

    for (int i = 1; i < n; ++i) {
        g[i][0] = b[i];
    }

    for (int j = 1; (1 << j) < n; ++j) {
        for (int i = 1; i + (1 << j) - 1 < n; ++i) {
            g[i][j] = __gcd(g[i][j - 1], g[i + (1 << (j - 1))][j - 1]);
        }
    }

    for (int i = 1; i <= q; ++i) {
        int l, r, x;
        cin >> l >> r >> x;
        query[l].push_back({r, x, i});
    }

```

```
    map<int, int> last;
    for (int i = n; i >= 1; --i) {
        if (last.count(a[i])) {
            upd(last[a[i]], -1);
        }
        last[a[i]] = i;
        upd(i, 1);
        for (array<int, 3> it : query[i]) {
            int r = it[0], x = it[1], id = it[2];
            int distinct = r - i + 1;
            if (i == r) {
                ans[id] = 1;
                continue;
            }
            int Max = get_max(i, r), Min = get_min(i, r);
            if (x == 0) {
                ans[id] = Max == Min;
            } else {
                int cnt = get(r), len = r - i;
                if ((Max - Min) % len || cnt != distinct) {
                    ans[id] = false;
                } else {
                    int tmp = (Max - Min) / len;
                    int gcd = get_gcd(i, r - 1);
                    ans[id] = (tmp == x && gcd == x);
                }
            }
        }
    }

    for (int i = 1; i <= q; ++i) {
        cout << (ans[i] ? "YES" : "NO") << '\n';
    }
}
```

Bài 5. Jelly Bear (500 điểm)

Note

- Ta sẽ giải bài toán một cách độc lập với mỗi thành phần liên thông $G = (V, E)$.
- Nếu $|V|$ lẻ, bài toán không có nghiệm (vì tổng bậc của mọi đồ thị là chẵn).
- Nếu $|V|$ chẵn, G luôn tồn tại đồ thị con mà mọi đỉnh đều có bậc lẻ.
  - **Claim**: Mọi đồ thị cây có $2k(k \geq 0)$ đỉnh luôn tồn tại một đồ thị con sao cho mọi đỉnh có bậc lẻ.
  - **Proof**:
    1. Claim hiển nhiên đúng với $k = 0$.
    2. Với $x > 0$, giả sử claim đúng với mọi $k < x$:
       - Với mọi cây T có $2x$ đỉnh:
         - Nếu cây T không tồn tại đỉnh bậc chẵn, T tồn tại đồ thị con mà mọi đỉnh đều bậc lẻ (chính T).
         - Nếu cây T tồn tại đỉnh u bậc chẵn, vì n chẵn nên sẽ tồn tại đỉnh v kề u mà có thể cắt cạnh (u, v) để chia T thành 2 cây con T_1, T_2 sao cho số đỉnh trong T_1 và số đỉnh trong T_2 đều chẵn và bé hơn $2x$. Quy nạp lại, T tồn tại đồ thị con mà mọi đỉnh đều bậc lẻ.
       - Vậy claim cũng đúng với $k = x$.
  - Vì G liên thông nên G tồn tại đồ thị con mà mọi đỉnh đều bậc lẻ (đồ thị con thỏa mãn của *spanning tree* của G).
  - Thuật toán: Từ G lấy *spanning tree* của nó ra. Xóa mọi cạnh trong cây này chia cây thành 2 phần đều chẵn đỉnh. Đồ thị thu được sẽ có mỗi đỉnh đều bậc lẻ.

Complexity: $O(n)$

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

void dfs(vector<vector<int>> &adj, vector<bool> &vis, vector<int> &siz, vector<vector<int>> &favor, int u) {
    vis[u] = true;
    siz[u] = 1;
    for (auto v : adj[u]) {
        if (!vis[v]) {
            dfs(adj, vis, siz, favor, v);
            favor[u].push_back(v);
            siz[u] += siz[v];
        }
    }
}

void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> adj(n);

    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        u--;
        v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    string answer = "YES";
    vector<bool> vis(n, false);
    vector<int> siz(n);
    vector<vector<int>> st_adj(n); // spanning forest's adjacency list

    for (int root = 0; root < n; root++) {
        if (vis[root]) {
            continue;
        }
        dfs(adj, vis, siz, st_adj, root);
        if (siz[root] % 2 == 1) {
            answer = "NO";
            break;
        }
    }

    cout << answer << '\n';
    if (answer == "YES") {
        vector<array<int, 2>> sub_edges;
        for (int u = 0; u < n; u++) {
            for (auto v : st_adj[u]) {
                if (siz[v] % 2 == 1) {
                    sub_edges.push_back({ u, v });
                }
            }
        }

        cout << sub_edges.size() << '\n';
        for (auto e : sub_edges) {
            cout << e[0] + 1 << ' ' << e[1] + 1 << '\n';
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
```

```
    cin.tie(nullptr);

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


```

Bài 6. Decrease XOR (600 điểm)

Subtask 2

Từ giờ, ta sẽ dùng $\oplus$ để đại diện cho toán tử ***XOR***.

Gọi $f(a)$ = tổng $\oplus$ của mảng a.

Ta sẽ lưu $dp(i, j, sum)$: Số cách thực hiện j thao tác trên i phần tử đầu tiên của mảng a để $f(a) = sum$.

Cụ thể hơn, giả sử ta được thực hiện $j - i$ thao tác trong $i - 1$ phần tử đầu tiên, tiếp theo, ta thao tác trên a_i, t lần ta có:

- $f(a) = f(a) \oplus a_i \oplus (a_i - t)$
- Có $\binom{j}{t}$ cách chọn vị trí thao tác.

Vậy ta có công thức truy hồi là:

$$dp(i, j, sum) = \sum \binom{j}{t} \times dp(i - 1, j - t, sum \oplus a_i \oplus (a_i - t)$$

Độ phức tạp: $O(n2^c k^2)$

Subtask 3

Thuật toán

Gọi $S = f(a)$ với mảng a ban đầu.

Ta định nghĩa lại mảng dp một chút: dp_i, j, sum là số cách thực hiện j thao tác trên i phần tử đầu tiên của mảng a để $S \oplus f(a) = sum$.

Với định nghĩa mới này, ta chỉ cần quan tâm tới tổng XOR của các $a_i \oplus (a_i - t)$. Mặc dù độ phức tạp vẫn không thay đổi, nhưng lúc này ta sẽ có một số nhận xét mới để cải thiện thuật toán.

Gọi $g(x) = \lfloor \log_2(max(x, x \oplus 1, x \oplus 2, ..., x \oplus k)) \rfloor$

Rõ ràng, $g(a_i)$ cũng chính là số bit (tối đa) mà thao tác trên a_i có thể ảnh hưởng đến sum.

Ta sẽ thực hiện (thuật toán) sau: Sort a theo $g(a_i)$ tăng dần. Với mỗi a_i, ta chỉ duyệt qua $sum < 2^{g(a_i)}$

Độ phức tạp của thuật toán trên đã giảm một cách đáng kể, **nhưng đáng kể là bao nhiêu?**

Chứng minh

Vẫn nhắc lại, mảng a chỉ gồm các phần tử **phân biệt**, ta giả sử trường hợp tệ nhất khi $n = 2^c - k$, hay, mọi phần tử $\geq k$ đều xuất hiện trong mảng a.

Trước hết, dễ thấy $g(a_i) \geq \log_2 k$. Vậy ta chỉ xét các $g(a_i) = v$ với $\log_2 k \leq k \leq c$.

Nhận xét quan trọng: Số giá trị p sao cho $g(p) \geq v$ là $O(2^{c-v+\lfloor \log_2 k \rfloor})$. (Chứng minh khá dễ, nhường lại cho bạn đọc hoặc bạn đọc có thể tự code vét để kiểm chứng).

$\rightarrow$ Số giá trị p sao cho $g(p) = v$ là $O(2^{c-v+\lfloor \log_2 k \rfloor - 1})$.

Vậy, với độ phức tạp khi quy hoạch động qua tất cả các giá trị a_i có $g(a_i) = v$ sẽ là:

$O(2^{c-v+\lfloor \log_2 k \rfloor - 1} x 2^v x k^2) = O(2^{c+\log_2 k} k^2) = O(2^c k^3)$.

Vậy **độ phức tạp** tổng thể của thuật toán là: $O(c2^c k^3)$

```cpp
#include <bits/stdc++.h>
#define endl '\n'

using namespace std;

const int MOD = 998244353;

const int N = 2e5 + 5;
const long long INF = 1e18 + 7;
const int MAXA = 1e9;
const int B = sqrt(N) + 5;

vector <int> v[20];
int dp[2][20][1 << 16], a[N], fact[N];
int C[20][20];

int Comb(int n, int k){
    if (n < k) return 0;
    return Div(fact[n], prod(fact[k], fact[n - k]));
}

void solve()
{
    int n, k, c, sumxor = 0; cin >> n >> k >> c;
    for(int i = 1; i <= n; ++i){
        cin >> a[i];
        sumxor ^= a[i];

        int ma = 0;
        for(int j = 0; j <= k; ++j){
            int eff = a[i] ^ (a[i] - j);
            ma = max(ma, eff);
        }

        int ct = 0, tmp = 1;
        while (tmp <= ma) tmp <<= 1, ++ct;
        v[ct].push_back(a[i]);
    }

    fact[0] = 1;
    for(int i = 1; i < N; ++i) fact[i] = prod(fact[i - 1], i);

    for(int i = 0; i <= k; ++i){
        for(int j = 0; j <= i; ++j) C[i][j] = Comb(i, j);
    }

    int tmp = 0;
    dp[0][0][0] = 1;
```

```
    for(int p = 1; p <= c; ++p){
        for(int val : v[p]){
            tmp ^= 1;
            for(int step = 1; step <= k; ++step){
                int eff = val ^ (val - step);

                for(int i = 0; i + step <= k; ++i){
                    for(int j = 0; j < (1 << p); ++j){
                        add(dp[tmp][i + step][j ^ eff], prod(dp[tmp ^ 1][i][j], C[i + step
    ][step]));
                    }
                }
            }

            for(int i = 0; i <= k; ++i){
                for(int j = 0; j < (1 << p); ++j){
                    add(dp[tmp][i][j], dp[tmp ^ 1][i][j]);
                    dp[tmp ^ 1][i][j] = 0;
                }
            }
        }
    }

    int sum = 0;
    for(int j = 0; j < (1 << c); ++j) add(sum, dp[tmp][k][j]);
    for(int j = 0; j < (1 << c); ++j){
        cout << Div(dp[tmp][k][j ^ sumxor], sum) << " ";
    }
}

signed main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    // freopen("codeforces.inp","r",stdin);
    // freopen("codeforces.out","w",stdout);

    int t = 1; // cin >> t;
    while (t--)
    {
        solve();
    }
}
```

— HẾT —