TEAMBUILDING

Đề bài: Chia dãy thành các nhóm sao cho chênh lệch giữa 0 và 1 không quá k.
Solution:
Subtask 1: n <= 20.
Với mỗi vị trí i ta có 2 cách chọn: hoặc cắt đoạn kết thúc tại i, hoặc ghép với đoạn trước đó.
Duyệt 2^n trường hợp, mỗi trường hợp kiểm tra điều kiện đề bài.
Subtask 2, 3:
Gọi f(i) = số cách chia dãy [1…i].
f(0) = 1
f(i) = tổng các f(j) nếu đoạn [j+1…i] có chênh lệch giữa 0 và 1 không quá k.
Với subtask 2, để kiểm tra đoạn [j+1…i], ta có thể duyệt từ j+1 -> i và đếm trâu.
Để qua subtask 3, ta sử dụng prefix sum.
Độ phức tạp: O(n^2)

FRUITMARKET

Hôm qua do mình nhầm chỗ assume luôn một lượt đi là tổng cả dãy, còn thuật toán vẫn giữ nguyên.
Trong khi m > 0:

- Đầu tiên mình đi một lượt dãy để tính xem sẽ pick được bao nhiêu với tổng = m, tạm gọi tổng các phần tử thu được là s và số lượng các phần tử thu được là c.
- Số lượt sẽ cộng thêm là: c * (m / s) (chia lấy nguyên).
- m lúc này trở thành m % s

CARDGAME

Đề bài: Chọn đoạn [l, r] sao cho sum[l, r] - max[l, r] lớn nhất.
Solution:
Subtask 2: Duyệt mọi đoạn [l, r] có thể.
Để không cần cài RMQ, chúng ta sẽ cập nhật max và sum cùng lúc với việc for j.

```
for (int i = 1; i <= n; i++) {
      int64_t sum = 0;
      int max = 0;
      for (int j = i; j <= n; j++) {
            sum += a[j];
            max = max(max, a[j]);
            cost = max(cost, sum - max);
      }
}
```

Subtask 4:
Với mỗi phần tử a[i], ta tính được L[i], R[i] là phần tử bên trái và bên phải i nhất sao cho a[i] làm max trong đoạn [Li, Ri].
Để tính mảng L, R trong O(n), ta sử dụng stack.
Gọi pre[i] = a_1 + … + a_i. Như vậy tổng một đoạn [j, k] là pre[k] - pre[j - 1]

Sau khi tính được mảng L, R, ta duyệt lại mảng a. Giả sử đang duyệt phần tử a[i], vì a[i] làm max trong đoạn [Li, Ri], nên nếu ta chọn một đoạn [j,k] sao cho $L_i <= j <= i <= k <= R_i$ thì đoạn [j,k] sẽ có a_i làm max. Dễ thấy max đã cố định nên ta cần tìm đoạn có tổng lớn nhất → chọn max pre[k] và min pre[j - 1]. Sử dụng segment tree/sparse table để lấy min nhanh.

```
————————|—————|—————|———————|—————|———————————————
       L_i    j    a_i      k    R_i
```

DISTANCE

Đề bài: Cho 1 cây, với mỗi đỉnh u, tìm đỉnh được tô màu gần u nhất (khác u)
Solution:
Giả sử đang tìm kết quả cho đỉnh r. Ta sẽ chọn đỉnh r làm gốc của cây, sau đó dfs từ r để tính 2 mảng:

- f[u] = đỉnh được tô màu gần nhất trong cây gốc u. nếu u được tô: f[u] = 0
- g[u] = min(f[v] + w(u, v)) với v là con u, w(u, v) là độ dài đường đi trực tiếp từ u đến v.

Kết quả cho r sẽ là g[r]. Tuy nhiên nếu mỗi lần ta chọn gốc thì sẽ mất đpt: $O(n^2)$. Vì vậy ta cải tiến, sử dụng kĩ thuật reroot dp để tính kết quả cho tất cả các gốc u với độ phức tạp: O(n)
Mình bị TLE do dùng vector quá nhiều, nhưng mà do không biết tối ưu thế nào nên thôi dừng lại tại đây :/ Cảm ơn các bạn đã theo dõi

THREE

Ta sẽ sử dụng tham lam.

- Vì các số loại #2 chỉ có thể ghép với #1 để ra 3, nên ta sẽ ưu tiên loại 2 trước. Trường hợp này sẽ có min(b, a) tập
- Các số loại #1 có thể ghép với chính nó để được 3, nên sẽ có thêm a/3 tập.
- Cộng với c tập 3 có sẵn.

TRANSFORM

Thay vì đi từ A -> B, ta sẽ đi ngược từ B về A.
Trong khi B > 0:

- Nếu B = A → có thể biến đổi được → YES
- Nếu B lẻ → chắc chắn trước đó đã thực hiện thao tác * 10 + 1 → B = (B - 1) / 10
- Nếu B chẵn → chắc chẵn đã *2 → B = B / 2
- Lưu ý nếu bước nào không thực hiện được thì in NO.

Độ phức tạp: O(Q * log(B)), vì mỗi bước B bị giảm ít nhất 2 lần.

SWORD

Sort các con boss theo điểm sức mạnh, và đánh từ nhỏ đến lớn.

COLORBOX

Subtask 1, 2: Với mỗi i, ta duyệt j tăng dần, duy trì mảng đếm để kiểm tra xem dãy có phần tử nào xuất hiện >= 1 lần ko.
Độ phức tạp: O(n^2)

```
for (int i = 1; i <= n; i++) {
      // cnt[x] = số lượng số = x trong mảng hiện tại, khi đã xoá đoạn
[i,j]
      vector<int> cnt(n + 1, 0);
      // số lượng số có cnt[x] > 1 trong mảng hiện tại
      int violate = 0;
      for (int j = 1; j <= n; j++) {
            ++cnt[a[j]];
            violate += cnt[a[j]] == 2;
      }
      for (int j = i; j <= n; j++) {
            violate -= cnt[a[j]] == 2;
            --cnt[a[j]];
            if (violate == 0) {
                  ans = min(ans, j - i + 1); break;
            }
      }
}
```

Subtask 3: Với mỗi i, nếu ta gọi j là vị trí gần i nhất sao cho khi xoá đoạn [i,j] thì thoả mãn bài toán, thì khi sang đến vị trí i' > i, thì j' >= j, vì nếu j' < j thì ta chọn xoá đoạn [i, j'] sẽ tối ưu hơn → vô lý vì đang giả sử j là vị trí gần nhất.
Như vậy thay vì duyệt hết mọi j, ta sẽ sử dụng 2 con trỏ.
Độ phức tạp: O(n)