Bài 1: Odd skip:

Để tính số số chẵn từ L đến R, ta có thể tính số số chẵn từ 1 đến R trừ đi số số chẵn từ 1 đến L – 1.

Số số chẵn từ 1 đến X là X/2.

Vậy kết quả bài toán là R/2 - (L–1) / 2.

Bài 2: NKS

- Ý tưởng: Sử dụng duyệt quay lui kết hợp nhánh cận.

Duyệt quay lui với 3 thông tin: Hiện tại đang xét đến miếng gỗ thứ mấy (n'), số miễng gỗ đã chọn (k') và tổng chiều rộng các miễng gỗ đã chọn (s'). Tại mỗi trạng thái như thế, ta có thể chọn hay không chọn miếng gỗ hiện tại.

- Tối ưu:

Sắp xếp lại dãy giảm dần, giúp chọn các số lớn trước các số nhỏ để giảm giá trị tổng nhanh hơn.

Đặt cận: Dễ thấy khi xét đến một miếng gỗ, do đã sắp xếp giảm dần nên miếng gỗ hiện tại là miếng rộng nhất, miếng gỗ cuối cùng là miếng hẹp nhất. Ta đặt cận là s' + a[n'] * (k – k') ≥ s và s' + a[n] * (k – k') ≤ s.

Bài 3: Palindrome

- Subtask 2: N ≤ 1000

Tính chất: Một xâu có thể sắp xếp thành palindrome khi có ít hơn 2 kí tự xuất hiện hiện lẻ lần trong xâu đó. VD: *aaabcc* có *a* xuất hiện 3 lần, *b* 1 lần tức có 2 số lẻ nên không thể sắp xếp thành palindrome.

Cách làm đơn giản: Xét tất cả các đoạn (i, j), xét từ i đến j để đếm số lần xuất hiện của các kí tự, sau đó kiểm tra có bao nhiêu kí tự xuất hiện lẻ lần. Để tối ưu, ta có thể đi từ (i, j) tính luôn (i, j + 1) mà chỉ cần cập nhật thêm phần tử thứ j+1 vào mảng đếm. Dùng thêm một biến để kiếm soát số số lẻ trong mảng đếm giúp đạt được độ phức tạp O(n^2).

- Subtask 3: $N \leq 100000$

Do chỉ cần biết tính chẵn lẻ của số lần xuất hiện các kí tự nên thay vì sử dụng mảng đếm, ta chỉ cần sử dụng một bitset gồm 26 bit để lưu trữ dữ liệu này, mỗi bit tượng trưng cho một ký tự, bit=1 có nghĩa là xuất hiện lẻ lần, bit=0 là xuất hiện chẵn lần. Gọi bitset của đoạn [i,j] là bitset[i][j].

Đoạn [i, j] thỏa mãn yêu cầu khi bitset của nó có 0 hoặc 1 bit=1. Ta tính mảng F[i] = bitset[1][i], từ đó tính nhanh được bitset[i][j] = F[i – 1] XOR F[j].

Lúc này ra đảo ngược vấn đề lại, từ j tìm các vị trí p ($p \leq j$) sao cho F[p – 1] chỉ khác F[j] tối đa 1 bit, như vậy bitset[p][j] thỏa mãn yêu cầu.

Sử dụng một unordered_map (C++ STL) để lưu số lần xuất hiện của các giá trị trong F. Tại mỗi j, ta để nguyên hoặc đảo một bit của F[j] để tìm các trạng thái bitset thỏa mãn rồi dùng unordered_map ở trên xem số lần xuất hiện và cập nhật vào kết quả.

Do không gian số 26 bit khá nhỏ, công việc đếm đơn giản nên unordered_map chạy với thời gian trung bình là O(1). Độ phức tạp O(26*n)