SOLUTION 19/12-C

BÀI 1: Chọn bóng

Qhđ: F(i) = S lớn nhất khi chỉ xét i bóng đầu tiên, và bóng i bắt buộc phải chọn.

Kq bài toán là max(F(1), F(2), …, F(N))

Tính F(i):

bóng i được chọn. Giả sử bóng được chọn ngay trước bóng i là bóng j (j < i). Nghĩa là trong số các bóng j + 1, j + 2, …, i - 2, i - 1 không có bóng nào được chọn.

TH1: c(i) = c(j). màu bóng i, j giống nhau.

F(i) = F(j) + a * v(i)

TH2: c(i) khác c(j). màu bóng i, j khác nhau

F(i) = F(j) + b * v(i)

Ta muốn F(i) lớn nhất có thể, do đó:

- Trong tất cả các bóng có màu c(i), cần tìm bóng có F() lớn nhất (*)
- Trong tất cả các bóng có màu khác c(i), cần tìm bóng có F() lớn nhất (**)

Ta sẽ tính F(1), rồi F(2), F(3), …, F(N). Đặt G(c) = max(F(k)) với mọi bóng k có màu c. Trong quá trình tính F(), ta lưu 2 màu khác nhau x và y thỏa mãn:

- màu x là màu có G(x) lớn nhất
- màu y là màu có G(y) lớn thứ nhì

=> dễ dàng tính được TH (*) và (**) trong O(1), vì màu cần tìm sẽ là màu x hoặc y mà thôi.

Sau khi tính F(i) xong, ta cập nhật lại G() và x, y.

Đpt: O(N) cho mỗi truy vấn

BÀI 2: Những hộp kẹo

Subtask 1 (60% số điểm): Vét cạn

Với $n \leq 10$, ta duyệt qua hết tất cả các hoán vị vị trí của n hộp kẹo.

Với mỗi hoán vị $x_1, x_2, \dots, x_n$, ta xét việc đổ dồn kẹo lần lượt từ hộp thứ x_i sang hộp x_{i+1} $(1 \leq i \leq n-k)$, cứ đổ như thế $n-k$ lần để còn lại đúng k hộp. Với mỗi hoán vị ta tính chi phí cần để thực hiện, sau đó chọn chi phí nhỏ nhất trong số đó.

Độ phức tạp O(n!).

Subtask 2 (100% số điểm): Quy hoạch động trạng thái

Giả sử $x_1, x_2, \dots, x_n$ là các bit của số nguyên X thể hiện trạng thái của các hộp, trong đó $x_i = 1$ nếu hộp i chưa bị đổ dồn sang hộp khác và $x_i = 0$ trong trường hợp ngược lại.

Mỗi bước dồn hai hộp ta làm như sau:

Chọn một vị trí i bất kì trong X có bit là1, dồn kẹo từ hộp ở vị trí i sang hộp ở vị trí $j (j \neq i)$ trong X, mà bit thứ j cũng có giá trị là 1, sau đó chọn kết quả nhỏ nhất.

Gọi $f(x)$ là chi phí nhỏ nhất để dồn các hộp về trạng thái x, ta có công thức:

$f(x) = \min(f(x), f(y) + c_{i+1,j+1})$ với $0 \le i, j \le n - 1$

Độ phức tạp O(2^n*n^2)

BÀI 3: Điện toán đám mây

Lõi của máy tính i cho khách hàng j thuê được nếu f_i >= F_j => nghĩ tới việc sort các máy tính và đơn đặt hàng theo f_i, F_j tăng dần.

Sau khi sort xong, duyệt các máy tính/đơn đặt hàng theo thứ tự đã sort:

TH1: Đang xét đơn đặt hàng (C_k, F_k, V_k).

Vì ta đã sort theo f_i, F_j tăng dần, **mọi** máy tính đã xét **trước đó** không thể cho khách hàng đang xét thuê được, vì các máy tính đã xét trước đó sẽ có tốc độ f < F_k. Do đó, nếu quyết định chấp nhận đơn đặt hàng đang xét, ta coi như đang "nợ" khách hàng C_k lõi.

TH2: Đang xét máy tính (c_k, f_k, v_k):

Vì ta đã sort theo f_i, F_j tăng dần, ta có thể dùng lõi của máy tính đang xét để "trả nợ" cho **mọi** đơn đặt hàng đã chấp nhận **trước đó**, vì F của các đơn đặt hàng trước đó < f_k.

Vậy ta sẽ qhđ như sau:

dp(i, j) = lợi nhuận tối đa khi đã xét i "vật" (máy tính hoặc đơn đặt hàng) đầu tiên (theo thứ tự đã sort f, F) và ta đang "nợ" khách hàng đúng j lõi.

TH1: "Vật" thứ i là đơn đặt hàng (C_k, F_k, V_k)

Nếu chấp nhận đơn đặt hàng này: ta sẽ "nợ" thêm C_k lõi, thu thêm V_k lợi nhuận: dp(i, j) = dp(i - 1, j - C_k) + V_k

Nếu không chấp nhận đơn đặt hàng này: dp(i, j) = dp(i - 1, j)

TH2: "Vật" thứ i là khách hàng (c_k, f_k, v_k)

Nếu mua máy tính này: ta sẽ trả nợ được c_k lõi đã nợ trước đó, mất thêm v_k tiền

=> dp(i, j) = dp(i - 1, j + c_k) - v_k

Nếu không mua máy tính này: dp(i, j) = dp(i - 1, j)

Kết quả bài toán là dp(N + M, 0) (có N+M "vật", và ta không được phép nợ khách hàng lõi nào)

Đpt: O((N+M) * 50*M)