D.

Nhận xét: Việc di chuyển theo 4 hướng bài ra không cho phép quân mã đi thành chu trình, bởi sau mỗi bước đi, tổng x+y của tọa độ con mã luôn tăng. Nếu đi theo hướng (2, -1) hoặc (-1, 2) thì x+y tăng 1. Nếu đi theo hướng (1, 2) hoặc (2, 1) thì x+y tăng 3.

Do việc di chuyển không thể tạo thành chu trình, gọi f(x, y) là số cách để đi từ ô (0, 0) đến ô (x, y). Theo hướng thông thường, có hai cách để biểu diễn công thức QHĐ:
Từ f(x, y) ta cập nhật cho các trạng thái f(x - 1, y + 2); f(x + 1, y + 2); f(x + 2, y + 1); f(x + 2, y - 1).
Cách 2: f(x, y) = f(x + 1, y - 2) + f(x - 1, y - 2) + f(x - 2, y - 1) + f(x - 2, y + 1)

Để trả lời nhiều truy vấn, ta thấy rằng số cách để đi từ ô (x1, y1) đến ô (x2, y2) chỉ phụ thuộc vào x2 - x1 và y2 - y1. Do đó, ta chuẩn bị trước mảng f trong đó f(x, y) là số cách đi từ ô (0, 0) đến ô (x, y). Khi đó, đáp số của truy vấn trên là f(x2 - x1, y2 - y1). Trả lời mỗi truy vấn trong O(1).

Nhận xét: Trong tất cả các truy vấn, quân mã có nhu cầu đi từ ô (0, 0) đến ô (x, y) với -2000 <= x, y <= 2000. Biên di chuyển của quân mã có tọa độ từ -2000 đến 4000. Do x, y <= 2000 nên x + y <= 4000. Như đã phân tích ở trên, sau mỗi bước, tổng x+y của vị trí con mã tăng ít nhất 1, nên con mã không di chuyển quá 4000 bước. Giả sử con mã muốn đến vị trí có hoành độ x = -2000, có cần đi 2000 bước theo hướng (-1, 2). Khi đó, từ (0, 0) con mã đi tới vị trí (-2000, 4000). Khi đó tung độ y = 4000 là quá cao so với vị trí kết thúc. Do đó, con mã cần giảm tung độ y xuống bằng cách đi thêm 2000 bước nữa theo hướng (2, -1). Do đó, con mã đã đi đủ 4000 bước và không thể đi đâu thêm. Như vậy, ta thấy với giới hạn 4000 bước đi, con mã không bao giờ đi được tới hoành độ (hay tương tự, tung độ) < -2000.

Về cận trên tọa độ của vùng con mã có thể đi, giả sư con mã đi tới vị trí có hoành độ x = 4000. Khi đó, con mã mất 2000 bước đi theo một trong hai hước là (2, -1) hoặc (2, 1). Tuy nhiên, hoành độ x = 4000 là quá lớn so với vị trí kết thúc, do đó nó cần giảm hoành độ xuống bằng 2000 bước khác đi theo hướng (-1, 2). Do đó với giới hạn 4000 bước, con mã không thể đi tới vị trí có tọa độ lớn hơn 4000.

```
const int MIN_CORD = -2020;
const int MAX_CORD = 4040;
const int MOD = 998244353;

void add(int &x, int y) {
      x += y;
      if (x >= MOD) x -= MOD;
}

const int dx[] = {-1, 1, 2, 2};
const int dy[] = {2, 2, -1, 1};

int f[MAX_CORD - MIN_CORD + 1][MAX_CORD - MIN_CORD + 1];
#define F(x, y) f[(x) - MIN_CORD][(y) - MIN_CORD]
// do x >= MIN_CORD và y >= MIN_CORD nên x - MIN_CORD và y - MIN_CORD đảm bảo >= 0

F(0, 0) = 1; // từ ô (0, 0) đến ô (0, 0) có 1 cách
for (int sum = 2 * MIN_CORD; sum <= 2 * MAX_CORD; sum++)
      for (int x = MIN_CORD; x <= MAX_CORD; x++) {
            int y = sum - x;
```

```
            if (y < MIN_CORD || y > MAX_CORD) continue;
            if (F(x, y) == 0) continue;

            for (int i = 0; i < 4; i++) {
                  int nx = x + dx[i];
                  int ny = y + dy[i];
                  if (MIN_CORD <= nx && nx <= MAX_CORD && MIN_CORD <= ny && ny <= MAX_CORD)
                        add(F(nx, ny), F(x, y));
            }
      }

query(x1, y1, x2, y2): return F(x2 - x1, y2 - y1)
```

E.

Bản chất bài toán yêu cầu đếm số bộ số nguyên (a, b, c, d, e,...) thỏa mãn các ràng buộc. Với mỗi biến có hai ràng buộc về cận trên và cận dưới: Trong test ví dụ, các ràng buộc là:
1 <= a <= 2
a <= b <= 5
3 <= c <= b
a <= d <= 7

Từ hai dữ kiện: Mỗi biến chỉ bị chặn bởi tối đa một biến khác, và nếu bị chặn bởi biến khác, biến chặn cũng xuất hiện trước biến bị chặn. Do đó, các biến lập thành mô hình gồm nhiều cây.
Nếu một biến có cả hai cận đều là số, nó là gốc cây. Nếu một biến bị chặn bởi một biến khác, nó sẽ nhận biến này là cha.

Gọi F(i, j) là số cách gán các biến trong cây con gốc i sao cho biến i nhận giá trị j.
Giả sử biến i có hai con là u và v. Các ràng buộc tương ứng lần lượt là u <= i và v >= i. Với hai ràng buộc này, ta có công thức quy hoạch động là F(i, j) = [F(u, j) + F(u, j-1) + ... + F(u, 1)] * [F(v, j) + F(v, j+1) + ... + F(v, 300k)].

Nếu biến i bị chặn bởi cận dưới Li là số, ta gán F(i, 1) = F(i, 2) = ... = F(i, Li - 1) = 0. Tương tự, nếu biến i bị chặn bởi cận trên Ri là số, ta gán F(i, Ri + 1) = F(i, Ri + 2) = ... = F(i, 300k) = 0.

Code: https://ideone.com/q9j8HE

F.

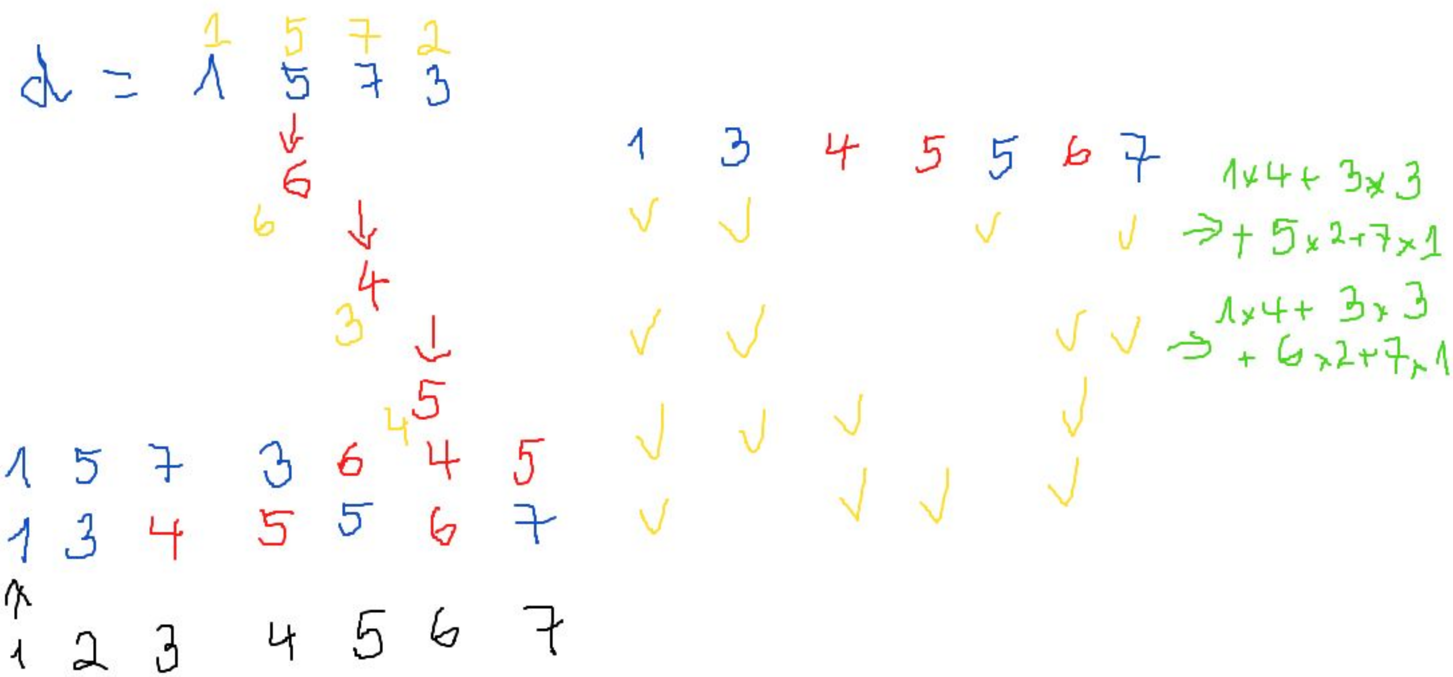
Giả sử n khách hàng muốn nhận vào các thời điểm t1 t2 ... tn. Thực tế, ta hoàn thành xong các đơn hàng vào các thời điểm s1 s2 ... sn. Khi đó, người thứ nhất cho t1 - s1 đồng, người thứ 2 cho t2 - s2 đồng,... người thứ n cho tn - sn đồng. Suy ra tổng số tiền nhận được là t1 - s1 + t2 - s2 + ... + tn - sn = (t1 + t2 + ... + tn) - (s1 + s2 + ... + sn). Do t1 + t2 + ... + tn là cố định nên ta cần cực tiểu hóa tổng s1 + s2 + ... + sn.

Giả sử các đơn hàng được xử lý theo thứ tự p1 p2 ... pn (p1 p2 ... pn là một hoán vị của 1 2 ... n). Khi đó, đến thời điểm d(p1) xử lý xong đơn hàng p1, thời điểm d(p1) + d(p2) xử lý xong đơn hàng p1 và p2, thời điểm d(p1) + d(p2) + d(p3) xử lý xong đơn hàng p3, ... thời gian xử lý xong hết toàn bộ n đơn hàng là d(p1) + d(p2) + ... + d(pn).

Tổng thời điểm xử lý xong n đơn hàng (tổng s1 + s2 + ... + sn ở trên) chính là d(p1) + [d(p1) + d(p2)] + [d(p1) + d(p2) + d(p3)] + ... + [d(p1) + d(p2) + ... + d(pn)] = n * d(p1) + (n-1) * d(p2) + (n-2) * d(p3) + ... + d(pn)

Do đó, trong tổng này món hàng thực hiện trước có hệ số cao hơn món hàng thực hiện sau. Như vậy, để tổng nhỏ nhất, ta phải có các đơn hàng được xử lý theo thứ tự thời gian tăng dần. Như vậy, bài toán của chúng ta là sắp xếp các số d1 d2 ... dn theo thứ tự tăng dần và tính giá trị biểu thức n * d(p1) + (n-1) * d(p2) + (n-2) * d(p3) + ... + d(pn). Với d(pi) là dãy các món hàng đã được sắp xếp: d(p1) <= d(p2) <= d(p3) <= ... <= d(pn).

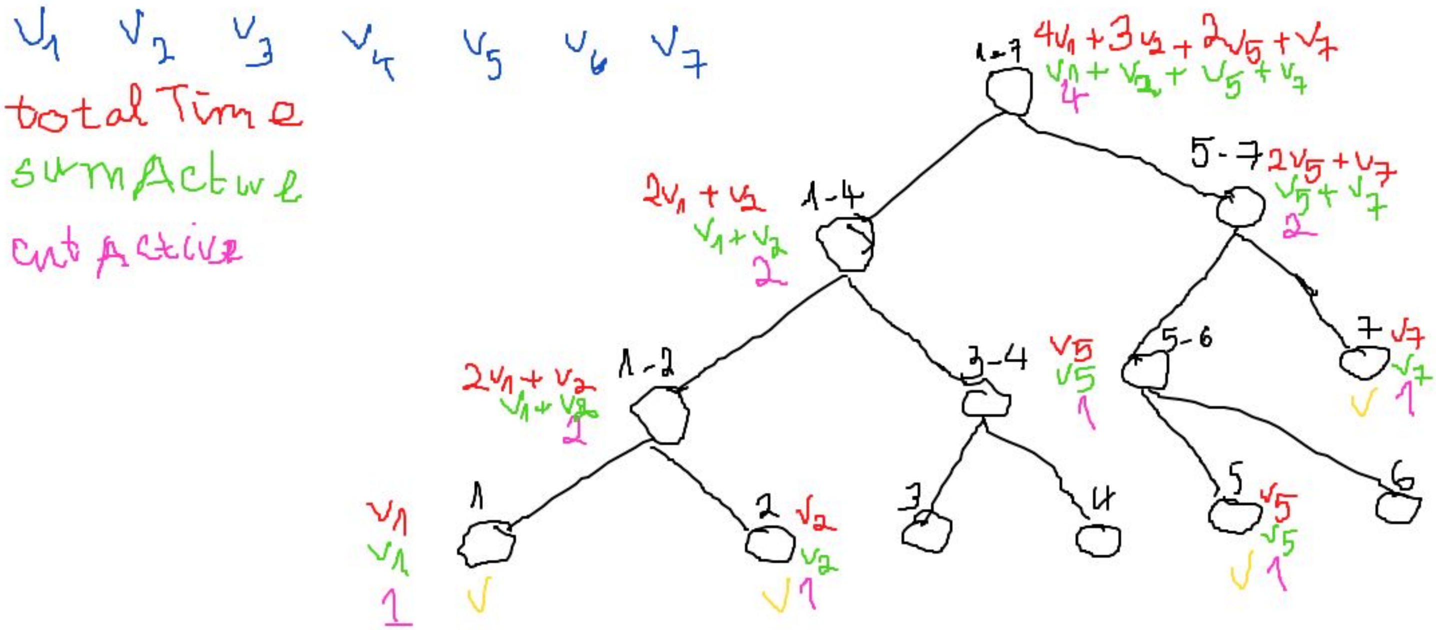
Giả sử dãy d ban đầu là d = (3, 6, 7, 2). Kết quả: 4 * 2 + 3 * 3 + 2 * 6 + 1 * 7
Sau khi thực hiện truy vấn thay số thứ 3, từ 7 thành 1, lúc này kết quả lại là 4 * 1 + 3 * 2 + 2 * 3 + 1 * 6



Mỗi một nút trên cây IT lưu ba giá trị:
+ TotalTime (màu đỏ trong hình dưới): Đáp số của bài toán nếu ngoài nút đang xét không có ô nào được bật
+ SumActive (màu xanh lục trong hình dưới): Tổng giá trị của các ô đang bật
+ CntActive (màu tím mộng mơ): Số ô đang bật.

Do bài này không có lazy update, ta chỉ quan tâm tới thao tác gộp hai nút (gộp thông tin ở hai nút con vào nút cha, được thực hiện ở cuối hàm update). Bài này thậm chí không có hàm get, vì bài toán chỉ yêu cầu tính tổng cả dãy, là một giá trị được tính sẵn ở nút gốc.

```
// gộp thông tin ở hai nút con (nút 2 * i và nút 2 * i + 1) vào thông tin ở nút cha (nút i)
cntActive[i] = cntActive[2 * i] + cntActive[2 * i + 1];
sumActive[i] = sumActive[2 * i] + sumActive[2 * i + 1];
totalTime[i] = totalTime[2 * i] + totalTime[2 * i + 1] + cntActive[2*i+1] * sumActive[2*i];
```



Xét tính toán ở nút gốc trong bài trên, con trái có total Time là 2*v1 + v2, con phải có total Time là 2 * v5 + v7. Sau khi cộng hai đại lượng này, ta được 2v1 + v2 + 2v5 + v7. Tuy nhiên, giá trị đúng phải là 4v1 + 3v2 + 2v5 + v7, do đó ta phải cộng thêm 2 vào hệ số của v1 và v2 nữa. Sở dĩ vì sao ta cộng vào hệ số của v1 và v2 thêm 2, vì ở nút bên phải có 2 nút được bật, chắc chắn hai nút bên phải được xếp sau hai nút bên trái. Vì thế, khi có thêm hai nút bên phải để vào sau, các nút bên trái có hệ số tăng 2, và do đó ta cộng thêm 2 * (v1 + v2). v1 + v2 chính là tổng giá trị của các ô đang bật.