برای محاسبه اینکه بازیکن ۱ چه سکه‌هایی را در هر مرحله بردارد تا در نهایت بیشترین جایزه را برنده شود می‌توانیم از راهبرد برنامه‌ریزی پویا ($dynamic\ programming$) استفاده‌کنیم. ابتدا باید تابع بازگشتی‌ای را تعریف کنیم که با گرفتن اندیس ابتدایی و انتهایی سکه‌ها به ما حداکثر جایزه‌ای را که بازیکن ۱ می‌تواند در یافت کند را برگرداند. پیش از ارائه این تابع، باید به این نکته توجه کنیم که بازیکن دوم نیز در تلاش برای بردن این بازی است :

$$f(m,n) = s - min\{f(m+1,n), f(m,n-1)\}$$

در اینجا s مجموع ارزش همه‌ی سکه‌هایی‌است که شماره آنها از m تا n است. تابع بدین شکل تعریف شده‌است که اگر بازیکن دوم کم‌ارزش ترین مجموع سکه‌ها را بردار آنگاه باقیمانده‌ی سکه‌ها با ارزش‌ترین مجموع را دارند که آنها را بازیکن اول برمیدارد. پیاده‌سازی این تابع بازگشتی به شکل زیر خواهد بود:

```
def max_value(list, start, end):
    if start == end:
        return list[start]
    coin_values = sum(list[start : end + 1])
    return coin_values - min(
        max_value(list, start + 1, end),
        max_value(list, start, end - 1),
    )
```

این تابع برای مثال قسمت ب مقدار 36 را برمی‌گرداند که درست است. می‌توانیم یا استفاده‌از راهبرد برنامه‌ریزی پویا و کاهش زمان اجرا در ازای مصرف حافظه علاوه بر سریع‌تر پیدا کردن مقدار جواب، خود جواب را نیز پیدا کنیم.

از آنجایی که در تابع به مقدار سطر بعد احتیاج داریم، ساخت جدول را از سطر آخر شروع می‌کنیم و دو شرط برای ساختن این جدول تعیین می‌کنیم تا به درستی ساخته شود: شرط اول اینست که اگر اندیس ابتدا از اندیس انتها بزرگتر باشد، مقدار این خانه از جدول صفر خواهد بود و شرط دوم اینست که اگر درایه، یک درایه قطری باشد یعنی تنها یک سکه برای برداشتن وجود دارد و مقدار بهینه همان ارزش این سکه خواهد بود و اگر شرطها برقرار نبودند، آنگاه مقادیر را با استفاده از شرط تابع پیدا می‌کنیم:

```
def coin_values_table(A: list):
    D = [[0] * len(A) for i in range(len(A))]
    for i in range(len(A) - 1, -1, -1):
        for j in range(len(A)):
            if i > j:
                D[i][j] = 0
            elif i == j:
                D[i][j] = A[i]
            else:
                s = sum(A[i : j + 1])
                D[i][j] = s - min(D[i + 1][j], D[i][j - 1])
    return D
```

حال کافیست با استفاده از این جدول حرکت‌های هر دو بازیکن را برای گرفتن بیشترین جایزه پیدا کنیم. بدین صورت که اگر با برداشتن سکه اول، حریف جایزه کمتری می گیرد آنرا برمی‌داریم در غیر اینصور سکه آخر را برمی‌داریم و دامنه سکه‌ها را بروز کرده و دوباره این مقایسه را انجام می‌دهیم حرکت‌هایی با اندیس زوج (اگر از صفر شروع کنیم)، حرکت‌های بازیکن اول و با اندیس فرد حرکت‌های بازیکن دوم خواهند بود:

```
def get_moves(A, D):
    moves = []
    domain = (0, len(A) - 1)
    for _ in range(len(A)):
        first_coin = D[domain[0] + 1][domain[1]]
        last_coin = D[domain[0]][domain[1] - 1]
        if first_coin < last_coin:
            moves.append(A[domain[0]])
            domain = (domain[0] + 1, domain[1])
        else:
            moves.append(A[domain[1]])
            domain = (domain[0], domain[1] - 1)
    first_player = [moves[i] for i in range(0, len(moves), 2)]
    second_player = [moves[i] for i in range(1, len(moves), 2)]
    return sum(first_player), first_player, second_player
```