به نام خدا

---

۱. $dp[n][k]$ برابر تعداد حالاتی است که می‌توان n را به صورت جمع اعدادی نوشت که حداکثر مقدار آن‌ها k باشد. جواب نهایی مسأله $dp[n][n]$است.

$$dp[n][k] = dp[n][k-1] + dp[n-k][k]$$

رابطه‌ی بالا برقرار است چرا که یا همه‌ی اعداد از $k-1$کمتر می‌باشند که در این صورت $dp[n][k-1]$آن‌ها را پوشش می‌دهد یا دست‌کم یک عدد با مقدار k داریم که در این صورت $dp[n-k][k]$این حالت را پوشش می‌دهد.آرایه بالا را می‌توان با مرتبه زمانی n^2 پر کرد.

۲. ارزش سکه‌ها را $v_1, v_2, \dots, v_n$در نظر می‌گیریم. $V(i,j)$ برابر است با بیشترین سود ممکن که از سکه‌های $v_i, \dots, v_j$بدست می‌آید.

$$V(i,j) = max(min(V(i+1,j-1), V(i+2,j)) + V_i, min(V(i,j-2), V(i+1,j-1) + V_j)$$

جواب نهایی نیز $V(1,n)$می‌باشد. آرایه بالا را می‌توان با مرتبه زمانی n^2پر کرد. برای راه‌حل برنامه‌نویسی پویا باید ابتدا V(i, i + 1)ها پر شوند. سپس V(i, i + 3)و ... و در نهایت V(1, n):

```
for (j = 1; j <= n; j += 2)
        for (i = 0; i <= n – j; ++i)
                fill V(i, i + j)
```

برای راه‌حل بازگشتی حافظه‌دار باید به چند نکته توجه کنیم اول اینکه همه‌ی عناصر آرایه را با مقداری که دور از واقعیت اند مقدار دهی می‌کنیم(مثلا بی‌نهایت یا یک مقدار منفی) سپس باید به حالت انتهایی بازگشتی توجه کنیم (در اینجا وقتی j ≤ i)

```
solve(i, j) {
        If (V(i, j) != ∞)
                Return
        If (j ≤ i)  // حالت پایانی
                V(i, j) = 0
                Return
        solve(i + 1, j - 1) & solve(i + 2, j) & solve(i, j - 2) & solve(i + 1, j - 1)
        fill V(i, j)
}
```

در این دو شبه‌کد تابع fill از رابطه بالا استفاده میکند.

۳. واضح است که رُندی یک عدد کمینه‌ی توان ۲ و ۵ در عدد است. فرض کنید pw_i2 و pw_i5 به ترتیب توان ۲ و ۵ در عدد i ام آرایه باشند. $dp[i][j][l]$برابر با حداکثر توان ۲ ایست که می‌توانیم با انتخاب j عدد از i عدد نخست که توان ۵ حاصل‌ضربشان l است جمع کنیم.(یعنی حاصل‌ضربشان با شرایط فوق بیشینه توان ۲ را بین تمام حالات داشته باشد) حال برای محاسبه‌ی آن دو راه وجود دارد : ۱-عدد i ام انتخاب نمی‌شود که در این صورت جواب مسأله $dp[i-1][j][l]$خواهد بود. ۲-عدد i ام انتخاب می‌شود که در این صورت جواب مسأله $dp[i-1][j-1][l-pw_i5]+1$خواهد بود. پس در کل جواب مسأله بیشینه‌ی این دو حالت خواهد بود. برای پر کردن آرایه چون i و j تا n تغییر می‌کنند و l نیز تا $a\log(a)$ حداکثر مقدار اعداد است.) تغییر می‌کند پس مرتبه زمانی کلی الگوریتم $O(n^2 log(a))$خواهد بود. در ضمن هیچ‌جای الگوریتم مقدار محاسبه‌ای بیشتر از $n \times a$نشده است.

۴. حروف غیر از 'O' و 'K' تفاوتی با یک‌دیگر ندارند پس آن‌ها را با حرف 'X' نشان می‌دهیم. $dp[o][k][x]$تعداد حرکات لازم برای قرار دادن o تا 'O' اول، k تا 'K' اول و x تا 'X' اول به ابتدای رشته می‌باشد.(این حروف باید o + k + x حرف اول رشته را تشکیل دهند.) هم‌چنین باید حرف آخر را نیز نگه داریم (چرا که اگر حرف آخر 'O' بود حرف بعدی نتواند 'K' بشود.) پس تعریف بالا را به $dp[o][k][x][IsTheLastLetter'O']$تغییر می‌دهیم. حال برای پر کردن آرایه باید 'K' بعدی (k + ۱ امین 'K' در رشته اصلی) یا 'O' بعدی(o + ۱ امین 'O' در رشته اصلی) یا 'X' بعدی(x + ۱ امین 'X' در رشته اصلی) را به رشته ساخته شده تا کنون اضافه کنیم. در ضمن باید توجه کنید که اگر حرف آخر تا کنون 'O' هست، نمی‌توان 'K' بعدی را اضافه کرد.

حال آخرین گام این است که هنگام اضافه کردن یک حرف امتیاز چگونه اضافه می‌شود. با کمی دقت می‌توان متوجه شد که اختلاف بین اندیس های آن‌ها(موقعیت اصلی در رشته و جایی که قرار می‌گیرد.) درست نیست.(سعی کنید مثال نقض آن را بزنید.) در عوض می‌دانیم چه حروفی به اول رشته منتقل شده اند، پس رشته دقیق فعلی را داریم و می‌توان امتیاز درست را محاسبه کرد به عنوان مثال فرض کنید رشته مورد نظر "OOKXXOKOO" است و می‌خواهیم از حالت $o = 4, k = 1, x = 1$ 'K' بعدی را اضافه کنیم. کافی است 'K' با حروف سمت چپش که تا کنون استفاده نشده‌اند جا به جا شود (در اینجا دومین 'X'). پس برای محاسبه افزایش امتیاز به زمان $O(n)$نیاز است و برای پر کردن آرایه $O(n^3)$، پس زمان کلی اجرای الگوریتم $O(n^4)$می‌باشد.

۵. راه‌حل این مسأله از دو مسأله برنامه‌نویسی پویا تشکیل شده است.

زیررشته‌ی متشکل از حروف i تا j را S_{ij}می‌نامیم. هم‌چنین C_iرا حرف i ام رشته در نظر می‌گیریم.

S_{ij} یک palindrome است در صورتی که حداقل یکی از شرایط زیر برقرار باشد:

- $i = j$
- $C_i = C_j, i + 1 = j$
- $C_i = C_j$ باشد و $S_{(i+1)(j-1)}$یک palindrome

بنابرین X_{ij}را تعریف می‌کنیم آیا رشته‌ی S_{ij}یک palindrome است یا خیر. برای پر کردن آرایه به زمان $O(n^2)$نیاز است.

حال D_jرا تعریف می‌کنیم کمترین تعداد palindrome تشکیل دهنده‌ی رشته‌ی S_{0j}. رابطی زیر برقرار است:

$$D_j = min\{D_i : i < j \ and \ X_{(i+1)j}\} + 1$$

که جواب مسأله D_{n-1}می‌باشد. برای پر کردن آرایه هم به زمان $O(n^2)$نیاز است پس در کل پیچیدگی زمانی الگوریتم $O(n^2)$می‌باشد.

۶. $dp[i][j]$ (برابر با کمترین تعداد palindrome حذف شده برای خالی کردن زیررشته‌ی i تا j است. چپ‌ترین حرف را در نظر بگیرید، یا به صورت مستقل حذف خواهد شد که در این صورت مسأله به $dp[i+1][j]$کاهش می‌یابد یا با یک حرف مشابه دیگر در سمت راست خود حذف خواهد شد. در این حالت فرض کنید k اندیس آن حرف باشد در این صورت مسأله را می‌توانیم برای $dp[i][k-1]$و $dp[k+1][j]$حل کنیم و در آخرین گام $dp[i][k-1]$، این دو حرف را نیز حذف کنیم. باید دقت کرد که حالتی که دو حرف اول یکسان باشند را باید جداگانه در نظر گرفت چرا که در حالات بالا پوشش داده نمی‌شود. برای همه‌ی k ها باید این تقسیم‌بندی انجام شود پس مرتبه زمانی الگوریتم $O(n^3)$می‌شود.