طراحی الگوریتم

جلسه حل تمرین دوم – برنامه‌ریزی پویا

۱ . سکه‌ها **۳۰ نمره**

الگوریتمی برای هر یک از سوالات زیر ارائه دهید و پیچیدگی زمانی آنها را نیز بررسی کنید.

الف) N سکه با قیمت متفاوت (مرتب شده) داریم. روشی ارائه دهید که دریابیم، آیا می‌توان با این سکه‌ها قیمت x را ارائه داد؟

ب) اگر از هر نوع سکه بی‌نهایت تا داشته باشیم، چه می‌توان کرد؟

ج) اگر از هر نوع سکه t_i تا داشته باشیم، چه راهکاری می‌توان ارائه داد؟

پاسخ :

الف)

تعریف DP: $DP[i][j]$ را از جنس $boolean$ تعریف می‌کنیم. مقدار $DP[i][j]$ معادل $True$ یا همان 1 است، اگر و تنها اگر با j سکه اول، بتوان قیمت i را ارائه داد و در غیر اینصورت برابر $False$ یا همان 0 است.

پایه DP: از آنجایی که برای ساخت قیمت صفر به هیچ سکه‌ای نیاز نداریم، پس در هر حالتی می‌توانیم این قیمت را بسازیم.

$$DP[0][j] = 1 \qquad for\ j\ from\ 0\ to\ n$$

رابطه DP: قیمت i را به دو روش می‌توان ساخت؛ یا آن را با $j-1$ سکه اول می‌سازیم، پس حتما با j سکه اول نیز می‌توان، یا برای ساخت آن حتما از سکه jام استفاده می‌شود، پس باقی قیمت ($i-p[j]$ که $p[j]$ ارزش سکه j ام است) را با $j-1$ سکه پیشین می‌سازیم.

$$DP[i][j] = (DP[i][j-1]\ \ or\ \ DP[i-p[j]][j-1])$$

پاسخ DP: طبق تعریف گفته شده، پاسخ مسئله (امکان ساخت قیمت x با n سکه) همان $DP[x][n]$ است.

پیچیدگی الگوریتم: محاسبه DP را طبق رابطه گفته شده، برای تمام سکه‌ها و قیمت‌های کوچکتر مساوی x انجام می‌دهیم؛ در نتیجه پیچیدگی این الگوریتم $O(nx)$ است.

```
int P[N], x
bool dp[X][N]
for (int j = 0; j <= n; j++)
    dp[0][j] = 1
for (int i = 1; i <= x; i++)
    for (int j = 1; j <= n; j++)
        dp[i][j] = (dp[i][j-1] || dp[i-p[j]][j-1])
```

ب)

تعریف DP: $DP[i]$ را از جنس $boolean$ تعریف می‌کنیم. مقدار $DP[i]$ معادل $True$ یا همان 1 است، اگر و تنها اگر با استفاده از این سکه‌ها، بتوان قیمت i را ارائه داد و در غیر اینصورت برابر $False$ یا همان 0 است.

پایه DP: از آنجایی که برای ساخت قیمت صفر به هیچ سکه‌ای نیاز نداریم، پس حتما می‌توان آن را ساخت.

$$DP[0] = true$$

رابطه DP: قیمت i را به دو روش می‌توان ساخت؛ یا آن را تنها با یک سکه (مثل سکه jام به ارزش $P[j]$) می‌سازیم، یا برای ساخت آن از چند سکه استفاده می‌کنیم که سکه jام به ارزش $P[j]$ یکی از آنهاست.

$$DP[i] = DP[i - P[j]] \qquad for\ any\ P[j] \leq i$$

پاسخ DP: طبق تعریف گفته شده، پاسخ مسئله (امکان ساخت قیمت x) همان $DP[x]$ است.

پیچیدگی الگوریتم: محاسبه DP را طبق رابطه گفته شده، برای تمام انواع سکه‌ها و قیمت‌های کوچکتر مساوی x انجام می‌دهیم؛ در نتیجه پیچیدگی این الگوریتم $O(nx)$ است.

```
int P[N], x
bool dp[N]
dp[0] = 1
for (int i = 1; i <= x; i++)
    for (int j = 1; j <= n; j++)
        if (P[j] > i)
            break
        if (dp[i-P[j]]){
            dp[i] = 1
            break;
        }
```

ج)

تعریف DP: $DP[i][j]$ را به صورت یک جفت $boolean$ و int تعریف می‌کنیم، که مقدار اول آن مشابه قسمت الف تعریف می‌شود (معادل $True$ یا همان 1 است، اگر و تنها اگر با i سکه اول، بتوان قیمت j را ارائه داد و در غیر اینصورت برابر $False$ یا همان 0 است) و قسمت دوم آن تعداد سکه‌هایی است که از iامین نوع سکه برداشته‌ایم.

پایه DP: از آنجایی که برای ساخت قیمت صفر به هیچ سکه‌ای نیاز نداریم، پس در هر حالتی می‌توانیم این

قیمت را بسازیم.

$$DP[i][0] = pair(true, 0) \qquad for\ i\ from\ 0\ to\ n$$

رابطه DP: قیمت j را به دو روش می‌توان ساخت؛ یا آن را با $i-1$ سکه اول می‌سازیم، پس حتما با i سکه اول نیز می‌توان، یا برای ساخت آن حتما از سکه iام استفاده می‌شود، پس باقی قیمت ($j - p[i]$ که $p[i]$ ارزش سکه i ام است) را با $i-1$ سکه پیشین و سایر سکه‌هایی که از نوع iام همچنان داریم، می‌سازیم.

$$DP[i][j] = \begin{cases} pair(true, 0) & if\ DP[i-1][j].first = ture \\ pair(true, DP[i][j-P[i]].second + 1 & if\ j \geq P[i]\ \&\ DP[i][j-P[i]].first \\ & \&\ DP[i][j-P[i]].second < t[i] \end{cases}$$

پاسخ DP: طبق تعریف گفته شده، پاسخ مسئله (امکان ساخت قیمت x با n سکه) همان $DP[n][x]$ است.

پیچیدگی الگوریتم: محاسبه DP را طبق رابطه گفته شده، برای تمام سکه‌ها و قیمت‌های کوچکتر مساوی x انجام می‌دهیم؛ در نتیجه پیچیدگی این الگوریتم $O(nx)$ است.

```
int P[N], t[N], x
pair<bool, int> dp[N][X]
for (int i = 1; i <= n; i++)
    dp[i][0] = pair(true, 0)
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= x; j++)
        if (dp[i-1][j].first)
            dp[i][j] = pair(true, 0)
        else if (j >= P[i] && dp[i][j-P[i]].first && dp[i][j-P[i]].second < t[i])
            dp[i][j] = pair(true, dp[i][j-P[i]].second + 1)
```

۲. مهارت حل مسئله ۲۰ نمره

درس طراحی الگوریتم شامل n دانشجو است. این دانشجویان به ترتیب در کلاس نشسته‌اند و مهارت دانشجو iام در حل مسائل برنامه‌ریزی پویا s_i است.

از آنجایی که مهارت این افراد ممکن است متفاوت باشد، برنا تصمیم گرفت دانشجویان را به چندین تیم با افراد مجاور تقسیم کند، تا به صورت گروهی به حل مسائل بپردازند. هر گروه می‌تواند شامل حداکثر k نفر باشد و هیچ‌کس نمی‌تواند عضو بیش از یک تیم باشد.

از آنجایی که دانشجویان از یکدیگر می‌آموزند، پس از مدت کمی سطح تمام اعضای هر تیم، به سطح مهارت خفن‌ترین عضو گروه می‌رسد.

روشی ارائه دهید تا بیشترین مجموع مهارت دانشجویان کلاس را در میان تمام حالات تیم‌بندی بیابیم.

پاسخ :

تعریف DP: $DP[i]$ را معادل بیشترین مجموع مهارت یک کلاس، شامل i دانشجو اول لیست، تعریف می‌کنیم.

پایه DP: اگر کلاس تنها شامل اولین دانشجو لیست باشد، در اینصورت بیشترین مجموع مهارت برابر مهارت همان دانشجو خواهد بود.

$$DP[1] = S[1]$$

رابطه DP: اگر بدانیم تعداد اعضای آخرین تیم برابر m است، در اینصورت بیشترین مجموع مهارت کلاسی شامل i دانشجو اول لیست برابر است با:

$$DP[i] = DP[i-m] + \max_{0 \le j < m} s_{i-j}$$

و باید در نظر داشت مقدار m می‌تواند هر عددی از 1 تا k باشد.

پاسخ DP: ما به دنبال یافتن بیشترین مجموع مهارت برای کلاسی شامل تمام n دانشجو لیست هستیم. درنتیجه پاسخ مسئله طبق تعریف DP برابر $DP[n]$ است.

پیچیدگی الگوریتم: طبق آنچه برای رابطه DP گفته شد، برای تمام iهایی که $1 \le i \le n$ باید بر روی تعداد اعضای گروه آخر حالت‌بندی کنیم و برای هر یک عنصر max را نیز به دست آوریم، که باعث می‌شود پیچیدگی زمانی الگوریتم $O(nk^2)$ باشد.

اما می‌توانیم با نگهداری max سرعت این الگوریتم را بیشتر کنیم. به‌روزرسانی مقدار max زمانی $O(1)$ می‌گیرد و در نتیجه پیچیدگی زمانی کل الگوریتم $O(nk)$ می‌شود.

```
int S[N], N, K
dp[1] = S[1]
for(int i = 2; i <= N; i++){
    int mx = S[i]
    for(int j = i; j >= 1 && i+1-j <= K; j--){
        mx = max(mx, S[j])
        if(j == 1)
            dp[i] = max(dp[i], mx*(i+1-j))
        else
            dp[i] = max(dp[i], dp[j-1] + mx*(i+1-j))
    }
}
```

۳. بازی هادی و پادی ۱۵ نمره

تعدادی سکه داریم که در یک ردیف پشت سر هم چیده شده‌اند و سکه‌ی i ام ارزش c_i دارد. هادی و پادی در هر نوبت سکه‌ها را برمی‌دارند به طوری که هادی به عنوان نفر اول یکی از سکه‌ها را برمی‌دارد و در ادامه پادی باید سکه قبلی و سکه بعدیِ هادی را (در صورت وجود) بردارد. بازی به همین شکل ادامه پیدا می‌کند. نفر اول یعنی هادی یک محدودیت دارد و آن این است که مجموع ارزش سکه‌هایی که برمی‌دارد نباید بیشتر از k مقدار شود. بیشترین مقداری که سکه‌های هادی می‌تواند داشته‌باشد چقدر است؟

پاسخ :

جدول DP را تعریف می‌کنیم به طوری که $DP[i][j]$ ماکزیمم مقداری است که نفر اول با i سکه اول بدست

می‌آورد، در حالی که حداکثر بتواند j مقدار داشته‌باشد. در ادامه جدول را به این صورت پر می‌کنیم:
حالت پایه:

$$DP[i][0] = DP[0][j] = 0$$

و برای j های بعد از $coin[0]$ داریم:

$$DP[1][j] = coin[0]$$

و از آن به بعد اگر $coin[i-1]$ از j کوچکتر یا مساوی باشد، داریم:

$$DP[i][j] = \max(DP[i-1][j], coin[i-1] + DP[i-2][j - coin[i-1]])$$

در غیر اینصورت یعنی اگر $coin[i-1]$ از j بزرگتر باشد:

$$DP[i][j] = DP[i-1][j]$$