۱- در این سوال باید سعی کنیم ورودی مسئله‌ها را به هم تبدیل کنیم. به مسئله‌ی بزرگ‌ترین زیر دنباله‌ی صعودی در یک دنباله‌ی اعداد LIS گویند، فرض کنید دنباله‌ی ورودی اولیه ما A باشد. حال باید این آرایه را به دو آرایه تبدیل کنیم که اگر مسئله‌ی LCS را روی آنها اجرا کنیم خروجی(LIS(A شود، برای این کار ما دو آرایه‌ی A و sorted(A) را انتخاب می‌کنیم. اگر مسئله‌ی LCS را روی این دو آرایه انجام دهیم به یک خروجی می‌رسیم که باید دو مورد زیر را ثابت کنیم:

1- به ازای هر زیر دنباله‌ی صعودی در A یک زیر دنباله‌ی مشترک در LCS وجود دارد: اثبات این موضوع سخت نیست زیرا هر دنباله‌ی صعودی در A هم در A و هم در مرتب‌شده‌ی A به صورت یک زیر دنباله وجود دارد.

۲-به ازای هر زیر دنباله‌ی مشترک در LCS یک زیر دنباله‌ی صعودی در A وجود دارد. با توجه به اینکه این زیر دنباله در هر دو وجود دارد و مشترک است،پس در مرتب‌شده‌ی A وجود دارد پس بدیهتا این زیردنباله صعودی خواهد بود و چون در خود A نیز وجود دارد پس به صورت زیردنباله‌ی صعودی در A است.

با اثبات موارد فوق به این نتیجه می‌رسیم که LCS گفته شده در واقع همان LIS است که هدف مسئله است. چون اردر LCS نیز ار اردر خواسته شده است پس رامحل ما درست است.

۲- تعریف برنامه‌نویسی پویای خود را به صورت زیر می‌نویسیم:

P[i] = column size of i

Dp[i][j] = min(dp[i][k] + dp[k+1][j] + p[i-1] * p[k] * p[j]) for all k between i and j-1

Dp[i][i] = 0

Dp[1][2] = 0 + 0 + 20 * 2 * 30 = 1200
Dp[2][3] = 0 + 0 + 2 * 30 * 12 = 720
Dp[3][4] = 0 + 0 + 30 * 12 * 8 = 2880

DP[1][3] = min(dp[1][1] + dp[2][3] + 20 * 2 * 12 , dp[1][2] + dp[3][3] + 20 * 30 * 12)
= min(0 + 720 + 480 , 1200 + 0 + 7200) = 1200

DP[2][4] = min(dp[2][2] + dp[3][4] + 2*30 *۶ 8, dp[2][3] + dp[4][4] + 2*12 * 8)

= min(0 + 2880 + 480, 720 + 0 192)

= 912

DP[1][4] = min(dp[1][1] + dp[2][4] + 20 * 2 * 8, dp[1][2] + dp[3][4] + 20 * 30 * 8, dp[1][3] + dp[4][4] + 20 * 12 * 8)

= min(1232, 8880 , 3120) = 1232

به ترتیب پر شدن خانه‌های DP دقت کنید این ترتیب با توجه به تعریف DP شما باید باشد. با توجه به اینکه ما برای پر کردن خانه‌ی(i,j) به تمام خانه‌هایی از DP که در این بین هستند یعنی شروع و پایانشان در این بین است نیاز داریم ابتدا DP هایی را پر می‌کنیم که اختلاف اندیس شروع و پایانشان یک است، سپس آنهایی که اختلاف آنها دو است و به همین ترتیب.

۳- الف) الگوریتم حریصانه‌ی ما بر اساس تقسیم ارزش بر وزن به دست می‌آید، یعنی:

Calc_value(1) = 50 / 5 = 10
Calc_Value(2) = 60 / 10 = 6
Calc_value(3) = 140 / 20 = 7

با توجه به مقدار گفته شده ما شی اول و سوم را به ترتیب برمی‌داریم که در مجموع سود ما برابر است با ۵۰ + ۱۴۰ = ۱۹۰.

توجه کنید که ما می‌توانستیم شی دوم و سوم را نیز برداریم که با توجه به اینکه به صورت حریصانه‌ی گفته شده عمل کردیم این کار امکان‌پذیر نبود.

ب) اگر با همان روش سوال قبل ارزش هر جنس را به دست بیاوریم به صورت بهینه خواهد بود زیرا ما برای هر واحد به دست آورده‌ایم که چقدر ارزش خواهد داشت.

پس الگوریتم ما به این صورت عمل می‌کند. ابتدا ۵ واحد از جنس اول برمی‌دارد، سپس ۲۰ واحد از جنس شماره‌ی ۳ برمیدارد و درنهایت ۵ واحد از جنس شماره ۲.

که مجموع سود به صورت زیر محاسبه می‌شود:

Sum_val = 10 * 5 + 7*20 + 6 * 5 = 220

ج) ابتدا نحوه‌ی محاسبه‌ی برنامه نویسی پویا را خواهیم نوشت:

Dp[w][i] = max(dp[w][i-1] , dp[w-w[i]][i-1] + v[i])

با توجه به اینکه اندازه‌ی جدول ساخته شده بزرگ خواهد بود تنها مقدار نهایی را می‌نویسیم.

Dp[30][3] = 140 + 60 = 200

۴- این روش به صورت حریصانه خواهد بود زیرا ما در هر مرحله با انتخاب آخرین کار تمام شده سعی می‌کنیم که تعداد کارهای انتخاب شده را بیشینه کنیم، در واقع ما برای بیشینه کردن کارهای انتخاب شده سعی می‌کنیم آخرین کاری که تمام می‌کنیم دیرترین حالت ممکن شروع شده باشد. به همین دلیل ما با یک انتخاب حریصانه سعی در بیشینه کردن یک مقدار داریم.
درستی الگوریتم: مانند بسیاری دیگر از الگوریتم‌های حریصانه روش اثبات ما به صورت زیر است:

فرض می‌کنیم جواب بهینه‌ای وجود داشته باشد که با جواب ما متفاوت باشد یعنی دنباله‌ی از task ها باشد که تعدادشان از task های انتخاب شده توسط ما بیشتر بوده و با یکدیگر اشتراک زمانی نداشته باشند.

آخرین task از لحاظ شروع زمانی که با یکدیگر تفاوت داریم را در نظر می‌گیریم در واقع task با شماره‌ی A توسط ما انتخاب شده است و task با شماره‌ی B توسط الگوریتم بهینه.

بدیهتا نقطه‌ی شروع A دیرتر از B خواهد بود(با توجه به انتخاب حریصانه‌ی ما) حال اگر در راه‌حل بهینه به جای B کار شماره‌ی A را انتخاب کنیم بهتر خواهد بود زیرا A دیرتر شروع شده است و احتمال اشتراک با task های قبلی کمتر خواهد بود.(برای درک بهتر روی کاغذ رسم کنید.)

۵- علت اصلی این موضوع ارتفاع درخت است، در مرحله‌ی اول برای انتخاب دو عنصری که کمترین تکرار را دارند جمع تکرارشان از تکرار هیچ عنصری کمتر نخواهد شد. در مرحله بعدی نیز همین اتفاق خواهد، یعنی دو عنصری که کمترین تکرار را دارند جمع تکرارشان از هیچ عنصری کمتر نخواهد بود. در واقع ۲۵۶ کاراکتر ما برگ‌های لایه‌ی آخر ما خواهند بود که ۱۲۸ پدر دارند و آنها نیز ۶۴ پدر و به همین ترتیب. در واقع درخت دودویی ساخته شده پر خواهد بود پس ارتفاع آن برابر است با(log(256 که برابر ۸ است، یعنی طول رشته‌های اختصاص داده به هر کاراکتر ۸ است که مزیتی نسبت به روش عادی اختصاص دادن رشته ندارد.

با توجه به مشکل پیش‌آمده نتوانستیم این جلسه کلاس را برگزار کنیم، اگر حس می‌کنید جایی از راه‌حل‌ها را هنوز متوجه نشدین می‌تونید از طریق‌های زیر سوالاتون رو بپرسید.

Email:mr.mim1377@gmail.com
Skype-id: live:.cid.3491ff6fa509156d

قسمتی از راه‌حل سوالات برای درک بهتر راه‌حل نوشته شده است و نمره‌ای ندارد.