مسئله: امین وارد یک آب‌میوه فروشی می‌شود. معده امین V لیتر ظرفیت آب‌میوه دارد!

در این آب‌میوه فروشی n نوع آب‌میوه وجود دارد. انواع آب‌میوه را با اعداد 1 تا n شماره‌گذاری می‌کنیم. از آب‌میوه‌ی نوع i ام ($1 \leq i \leq n$) به اندازه‌ی v_i لیتر در مخزن آن وجود دارد.

امین می‌داند که اگر همه‌ی ظرف آب‌میوه‌ی موجود در مخزن i ام را بنوشد، به اندازه‌ی h_i خوشحال می‌شود. همچنین اگر هر کسری از این ظرف را بخورد، به همان نسبت خوشحالی بدست می‌آورد. (برای مثال، اگر نیمی از v_i را بنوشد، به اندازه نصف h_i خوشحالی به دست می‌آورد.)

حال امین می‌خواهد از هر نوع آب‌میوه مقداری بنوشد به طوری که مجموع خوشحالی او بیشینه باشد. مقدار آب‌میوه می‌تواند هر عدد حقیقی نامنفی باشد!

الف) شبه‌کدهایی بنویسید که این مقدار بیشینه را محاسبه و در خروجی قرار دهد. از شما می‌خواهیم یک شبه‌کد *با رویکرد حریصانه* و یک شبه‌کد *با رویکرد پویا* ارائه نمایید.

در سطر اول ورودی، عدد صحیح و مثبت n و V داده می‌شود. در n سطر بعدی، در هر سطر، دو عدد h_i و v_i داده شده است که با یک فاصله از هم جدا شده‌اند.

ب) پیچیدگی زمانی شبه‌کدهای خود را به دست آورده و مقایسه نمایید.

پ) کدامیک از رویکردهای حریصانه یا پویا در مسئله کوله‌پشتی کسری بهتر عمل می‌کند؟ در مسئله کوله‌پشتی 0/1 چطور؟ با جستجو و ارائه دلیل، پاسخ خود را بیان نمایید.

الف) شبه کد

حریصانه: سؤال مانند مسئله کوله‌پشتی کسری است. با این تفاوت که در اینجا بر اساس نسبت خوشحالی به حجم انتخاب‌ها را انجام می‌دهیم. در کوله‌پشتی کسری، رویکرد حریصانه همیشه به جواب بهینه می‌رسد.

```
funct greedy_juices(n, V, juices):
    for i from 1 to n:
        weight[i] = hᵢ / vᵢ

    sort juices by weights in descending order

    happiness = 0
    capacity = V
```

```
    for each (h, v) in sorted juices:
        If v <= capacity:
            happiness += h
            capacity -= v
        else:
            happiness += (h / v) * capacity
            break

    return happiness
```

برنامه‌نویسی پویا: در حالت کسری، استفاده از برنامه‌نویسی پویا بهینه نیست، چون فضای حالت بی‌نهایت داریم. ایده این است که فضای حالت را به واحدهای کوچک تقسیم کنیم.

```
function dp_juices(n, V, juices):
    delta = 0.01
    size = int(V / delta)

    dp[0...size] = zeros
    for i from 1 to n:
        (h, v) = juices[i]
        max_units = int(v / delta)

        for j from size down to 0:
            for units from 0 to min(max_units, j):
                x = units * delta
                happiness = (x / v) * h
                dp[j] = max(dp[j], dp[j - units] + happiness)

    Return dp[size]
```

ب) پیچیدگی زمانی

حریصانه: مرتب‌سازی + حلقه انتخاب: $O(n) + O(n \log n) = O(n\log n)$

برنامه‌نویسی پویا: وابسته به دقت delta است. تعداد حالت‌های حلقه‌ها:

$O(n * (V/delta) * (v_{max}/delta))$ تقریباً: $O(n * V^2 / delta^2)$

رویکرد حریصانه بسیار بهتر عمل می‌کند.

پ) مقایسه رویکردها

در کوله‌پشتی کسری رویکرد حریصانه همیشه بهینه است. چون سود هر واحد وزن را داریم و می‌توانیم هر کسری از آیتم را انتخاب کنیم و با استقرا و برهان خلف می‌توانیم بهینه بودن جوابمان را اثبات کنیم. در کوله‌پشتی ۰/۱ رویکرد پویا بهتر عمل می‌کند. چون نمی‌توان کسری از آیتم برداشت، پس انتخاب‌ها ۰ یا ۱ هستند. در این حالت الگوریتم حریصانه ممکن است جواب بهینه را ندهد. زیرا بر اساس سود به وزن انتخاب می‌کند ولی آیتم‌های پر سود ممکن است حجم زیادی داشته باشند. پس با کمک رویکرد پویا همه حالت‌های انتخاب را در نظر می‌گیریم و ذخیره می‌کنیم و جواب مساله را بر اساس زیرمساله های کوچکتر تعریف می‌کنیم.

4023613068 - 4023613060