Answer-Quiz-1

Algorithm Design - Spring 2026

Instructor: Dr. Amin Eskandari

Head Teaching Assistants: Hamid Namjoo, Amir Hossein Hemati

Teaching Assistants: Ali Nikvan, Viyana Hasanbeigi, Nima Soltani, Reza Rezaie, Maryam Meidanshahi , Babak Janfaza , Fatemeh Dehgan , Elham Mosavi

---

سوال ۱ :

گام۰: قبل از شروع، تصویر کلی مسئله را بفهمید

مسئله از ٤ موضوع مختلف تشکیل شده:

- هافمن
- کوله‌پشتی کسری
- دیکد کردن بدون ابهام
- خرد کردن پول

کار شما این است که همه این‌ها را با هم ترکیب کنید

نه جداجدا

گام۱: فهمیدن داده‌های مسئله

جدول داده‌ها در مسئله:

کاراکتر | فراوانی | وزن | هزینه

A | 50 | 4 | 5

B | 40 | 3 | 3

C | 30 | 5 | 4

D | 20 | 2 | 2

E | 10 | 6 | 6

و محدودیت‌های مسئله:

- حداکثر وزن قابل ارسال = ۱۸۰

- سکه‌های قابل پرداخت = {۱، ۳، ٤}
- باید کد هافمن بسازید
- باید مطمئن شوید که دیکدینگ در بسته‌ها مبهم نشود
- باید انتخاب کنید هر کاراکتر چقدر ارسال شود
- باید هزینهٔ پرداخت را با سکه‌ها کمینه کنید

اشتباهات رایج در این مرحله:

ممکن است خیال کنید هدف فقط ساخت هافمن است نه!

بعضی‌ها وزن را با هزینه قاطی می‌کنند

بعضی‌ها فکر می‌کنند کوله‌پشتی از نوع ۱/۰ است اشتباه! کسری است

گام ۲: ساخت هافمن

چرا این مرحله مهم است؟

چون بعداً برای انتخاب مقدار ارسال کاراکترها، باید طول هر کد را بدانید

روش:

۱- فراوانی‌ها را مرتب کنید

۲- دو کمترین را با هم جمع کنید

۳- گره جدید را وارد کنید

۴- ادامه دهید تا یک درخت کامل بسازید

۵- از ریشه تا برگ کدها را پیدا کنید

نتیجهٔ هافمن (که از قبل محاسبه شده):

A → 0

B → 100

C → 101

D → 110

E → 111

گام۳: بررسی Packet-Safe بودن

اگر یک بسته وسط یک کد قطع شود

و ادامه آن در بسته بعدی بیاید

نباید در ترکیب این دو، یک کد معتبر دیگر ساخته شود

در این مسئله همه کدها طول حداقل ۳ دارند جز A که طولش ۱ است

اما A با هیچ کدی قابل اشتباه شدن نیست، چون: 0 = A

تمام کدهای دیگر با ۱ شروع می‌شوند

پس اگر A نصفه بیاید، به کد دیگری تبدیل نمی‌شود

اشتباهات رایج:

بعضی فکر می‌کنند باید طول همه کدها برابر باشد نه لازم نیست!

بعضی‌ها فقط Prefix-free بودن را چک می‌کنند و بخش Packet-Safe را فراموش می‌کنند

ممکن است فکر کنید اگر بسته پاره شود، کل کد خراب می‌شود؛ نه! تعریف مسئله این نیست

گام ۴: حل کوله‌پشتی کسری (Fractional Knapsack)

در این مرحله باید تصمیم بگیرید از هر کاراکتر چقدر ارسال کنیم؟

در مسئله ما معیار ارزش چیست؟

ارزش = فراوانی / طول_کد_هافمن

بعد ارزش/وزن را محاسبه می‌کنید:

A: 12.5

B: 4.44

D: 3.33

C: 2

E: 0.55

مراحل:

۱. کاراکترها را بر اساس ارزش/وزن مرتب کنید

۲. از بالاترین شروع کنید

۳. هر چه می‌توانید از آن بفرستید

٤. اگر وزن پر شد، از آن کاراکتر به مقدار کسری بردارید

٥. به کاراکتر بعدی نمی‌روید مگر اینکه وزن هنوز خالی باشد

نتیجه:

از A فقط ٤٥ عدد قابل ارسال است (٤٥×٤=۱۸۰)

کوله‌پشتی پر شد >>> بنابراین هیچ کاراکتر دیگری ارسال نمی‌شود

اشتباهات رایج در این بخش:

- اشتباه در «ارزش» بعضی‌ها مستقیم فراوانی را ارزش می‌گیرند
- اشتباه در محاسبه طول کد هافمن
- اشتباه در محاسبه ارزش/وزن
- انتخاب کاراکتر بعدی بدون پر شدن قبلی
- حل اشتباه به روش کوله‌پشتی ۱/۰
- ارسال بیش از ٥۰ عدد از A (فقط ٥۰ عدد داریم)

گام٥: حل مسئله خرد کردن پول (Coin Change)

هزینه برای ٤٥ عدد A:

٤٥×٥ = ۲۲٥

حالا با سکه‌های {۱،۳،٤} باید ۲۲٥ را بسازیم با کمترین تعداد

چون سکه ٤ ارزش بیشتری دارد (بزرگ‌تر است):

٢٢٥ ÷ ٤ = ٥٦ باقیمانده ١

پس:

٥٦ × ٤ = ٢٢٤

١ × ١ = ١

جمع ٢٢٥

حداقل تعداد سکه = ٥٧

اشتباهات رایج:

تصور اینکه چون ٣ و ٤ هستند، شاید ١ لازم نیست

اشتباه در تشخیص هدف: هدف کمینه کردن تعداد سکه است نه کمینه کردن مجموع پول که ثابت است

سوال ٢ :

این مسئله در واقع٥ مرحله وابسته به هم دارد

اگر یک مرحله را اشتباه انجام دهید، مراحل بعدی هم خراب می‌شوند

ترتیب مراحل:

١. تولید رشته با فیبوناچی

٢. ساخت درخت هافمن

٣. رمزگذاری رشته

٤. کار با ماتریس و دنباله‌ها

٥. محاسبه خروجی نهایی

مرحله ۱ ساخت رشته S با فیبوناچی

مسئله چه می‌گوید؟

باید برای i از ۱ تا ۱۲:

۱- عدد فیبوناچی F(i) را پیدا کنید

۲- باقی‌مانده آن را بر ٤ حساب کنید

۳- عدد حاصل را به یک حرف تبدیل کنید

نگاشت داده شده:

A → ۰

B → ۱

C → ۲

D → ۳

گام ۱: تولید اعداد فیبوناچی

فرمول:

$$F(1) = 1$$
$$F(2) = 1$$
$$F(n) = F(n-1) + F(n-2)$$

پس تا ۱۲:

۱

۱

۲

۳

٥

۸

۱۳

۲۱

۳٤

٥٥

۸۹

۱٤٤

بعضی ها ممکنه از

$$F(0) = 0$$

شروع می‌کنند.

در این مسئله شروع از F(1) است

گام ۲: محاسبه mod 4

باید هر عدد را بر ٤ تقسیم کنید و باقی‌مانده را بگیرید

مثال:

٥ ÷ ٤ → باقی‌مانده ۱

۸ ÷ ٤ → باقی‌مانده ۰

نتیجه:

۱

۱

۲

۳

۱

۰

۱

۱

۲

۳

۱

۰

گام ۳: تبدیل به حروف

طبق نگاشتمون:

A → ۰

B → ۱

C → ۲

D → ۳

پس رشته S ساخته می‌شود

نکته مهم: ترتیب را تغییر ندهید

مرحله ۲ ساخت درخت هافمن

حالا مسئله چه می‌خواهد؟

باید ببینید هر حرف چند بار در رشته S تکرار شده است

گام ۱: شمارش فراوانی

تعداد هر حرف را بشمارید

مثلاً:

A چند بار؟

B چند بار؟

C چند بار؟

D چند بار؟

گام ۲: ساخت درخت هافمن

قاعده اصلی هافمن:

۱- دو گره با کمترین فراوانی را انتخاب کنید

۲- آن‌ها را ترکیب کنید

۳- وزن جدید = جمع آن دو

این کار را ادامه دهید تا یک ریشه باقی بماند

گام ۳: تعیین کدها

قاعده:

چپ ← ۰

راست ← ۱

هر مسیر از ریشه تا حرف ← کد آن حرف است

مرحله ۳ رمزگذاری رشته

مسئله چه می‌خواهد؟

باید هر حرف رشته S را با کد هافمن آن جایگزین کنید

مثال فرضی:

اگر

B = 0

A = 10

C = 110

و رشته:

B A C

رمز می‌شود:

۰ ۱۰ ۱۱۰

مرحله ٤ حذف بیت‌ها و ساخت ماتریس

مسئله چه می‌گوید؟

۲۰٪ از بیت‌ها حذف شده‌اند

گام ۱: محاسبه ۲۰٪

فرمول:

۰٫۲ × طول رشته

اگر طول ۲۲ باشد:

۰٫۲ × ۲۲ = ٤٫٤

پس تقریباً ٤ بیت حذف می‌شود

گام ۲: ساخت ماتریس

گفته شده:

ماتریس k×٤

یعنی:

٤ ردیف

K ستون

دقت کنید K برابر تعداد بیت‌های باقی‌مانده است

گام ۳: پر کردن ماتریس

بیت‌ها را ستونی وارد کنید:

از بالا به پایین

بعد ستون بعدی

نکته اینجا شاید شما ردیفی پر کنید در حالی که مسئله گفته بالا به پایین

مرحله ٥ محاسبه LIS

الان مسئله چه می‌گوید؟

هر ستون یک عدد دودویی ٤ بیتی است

مثلاً:

۱۰۱۰ → عدد ۱۰

پس باید:

۱- ستون‌ها را به عدد تبدیل کنید

۲- دنباله اعداد بسازید

۳. -L را پیدا کنید

LIS چیست؟

بلندترین زیر دنباله‌ای که:

به صورت صعودی باشد

مثال:

۵ ۲ ٤ ۳

= LIS

۵ ٤ ۳

طول = ۳

مرحله ٦ محاسبه LCS

دیگه حتما میدونید مسئله چه می‌خواهد؟

بین:

ردیف اول ماتریس

ردیف آخر ماتریس

باید LCS را پیدا کنیم

LCS چیست؟

بلندترین دنباله مشترک بین دو رشته

مثال:

ABCBDAB

BDCAB

LCS = BCAB

مرحله ۷ خروجی نهایی

فرمول داده شده:

Output = LIS × LCS

فقط کافی است:

طول LIS

طول LCS

را ضرب کنید.

خب میرسیم به پاسخ نهایی مسئله

بعد از انجام همه مراحل:

LIS = 1

LCS = 18

پس:

Output = 18

سوال ۳ :

بخش الف

برای پیدا کردن بازه ای که مجموع عناصرش بیشترین مقدار باشد، می‌توان آرایه را **فقط یک بار** از ابتدا تا انتها پیمایش کرد و یک متغیر به نام **مجموع جاری (current sum)** نگه داشت.

روند کار:

- در هر قدم، مقدار عنصر فعلی را به **مجموع جاری** اضافه می‌کنیم.

- اگر **مجموع جاری** منفی شود آن را **صفر** می‌کنیم، چون ادامه دادن یک مقدار منفی باعث می‌شود هر بازه‌ای که بعداً شروع کنیم از همان ابتدا مقدار نامطلوب بگیرد.
- در هر لحظه اگر مقدار **مجموع جاری** از **مجموع بیشینه** ثبت شده بیشتر شد، مقدار **مجموع بیشینه** را به روز رسانی می‌کنیم.
- با این روش هر بار که **مجموع جاری** صفر می‌شود، در واقع بازه جدیدی را شروع می‌کنیم.

بخش ب

برای انتخاب سه بازه جدا از هم، ایده کلی این است که:

۱. در ابتدا آرایه را طوری پردازش کنیم که در هر نقطه آرایه بدانیم:
   - اگر از سمت چپ تا این نقطه بیاییم، **بهترین مجموع بیشینه** در بازه‌های قبلی چه بوده است.
   - اگر از سمت راست تا این نقطه برویم، **بهترین مجموع بیشینه** در ادامه آرایه چه خواهد بود.

۲. این دو اطلاعات (بهترین از چپ و بهترین از راست) را می‌توان با همان منطق بخش الف به دست آورد.

با این تفاوت که این بار برای **هر نقطه**، مقدار بهترین بازه ممکن در آن سمت ذخیره می‌شود نه فقط یک مقدار در کل آرایه.

۳. سپس یک بازه وسطی انتخاب می‌کنیم (باز هم با همان روند مجموع جاری و مجموع بیشینه) و برای هر انتخاب ممکن بازه وسط، مقادیر زیر را محاسبه می‌کنیم:
   - بهترین بازه قبل از آن (از نتایج سمت چپ)
   - بهترین بازه بعد از آن (از نتایج سمت راست)
   - مقدار بازه وسط

٤. مجموع این سه مقدار را بررسی می‌کنیم و بیشترین مقدار را نگه می‌داریم.

سوال ۴ :

بخش الف

۱. **چرا نمی‌توان از ایده تقسیم شیرینی‌ها استفاده کرد؟**

در مسئله کوله پشتی کسری (fractional knapsack)، فرض بر این است که می‌توانیم یک کالا را به **قسمت‌های دلخواه** تقسیم کنیم. مثلاً اگر یک کیک وزن زیادی دارد و ارزش کمی نسبت به وزنش دارد، می‌توانیم فقط یک برش از آن را برداریم. این کار به ما اجازه می‌دهد که نسبت ارزش به وزن را برای همه کالاها محاسبه کرده و **کالاهایی با بالاترین نسبت را انتخاب کنیم**، حتی اگر مجبور شویم بخشی از آخرین کالا را برداریم تا ظرفیت دقیقاً پر شود.

اما در این مسئله (و مسئله کوله پشتی۰ /۱)، هر شیرینی باید **کامل** انتخاب شود یا اصلاً انتخاب نشود. ما نمی‌توانیم مثلاً نصف شیرینی گردویی را برداریم. این محدودیت، یعنی **گسسته‌بودن انتخاب‌ها** (۰ یا ۱)، باعث می‌شود که رویکرد حریصانه (greedy) دیگر جواب بهینه ندهد. دلیلش این است که ممکن است برداشتن یک شیرینی با نسبت ارزش به وزن کمی پایین‌تر، ما را قادر سازد شیرینی‌های دیگری را برداریم که در نهایت ارزش کل بیشتری ایجاد کنند. این امکان در حالت کسری وجود ندارد چون ما همیشه بهترین نسبت را انتخاب می‌کنیم و فضا را پر می‌کنیم.

۲. **چرا رویکرد حریصانه (Greedy) در مسئلهٔ ۱/۰ جواب بهینه نمی‌دهد؟**

رویکرد حریصانه بر اساس بهترین نسبت ارزش به وزن عمل می‌کند. فرض کنید شیرینی‌ها این‌گونه باشند:

- شیرینی X: ارزش ۱۰، وزن ۳ (نسبت ≈ ۳٫۳۳)
- شیرینی Y: ارزش ۱۲، وزن ٤ (نسبت ≈ ۳)
- شیرینی Z: ارزش ۱۸، وزن ٦ (نسبت = ۳)

اگر ظرفیت کوله W=7 باشد:

- رویکرد حریصانه اول X را انتخاب می‌کند (ارزش ۱۰، وزن ۳). کوله $7 - 3 = 4$ فضا دارد.
- بعد به سراغ Y می‌رود (نسبت ۳). اگر Y را بردارد، مجموع وزن $3 + 4 = 7$ می‌شود و ارزش کل $10 + 12 = 22$.
- اما اگر به جای Y، شیرینی Z را (که نسبت کمتری دارد ولی وزنش بیشتر است) برمی‌داشتیم، شاید نمی‌توانستیم X را هم برداریم.

در مثال بالا، اگر ابتدا X را برداریم (وزن ۳، ارزش ۱۰)، ٤ واحد فضا باقی می‌ماند. سپس اگر Y را برداریم (وزن ٤، ارزش ۱۲)، وزن کل $3 + 4 = 7$ و ارزش کل $10 + 12 = 22$ می‌شود.

حالا فرض کنید شیرینی دیگری داشته باشیم:

- شیرینی W: ارزش ۱۵، وزن ۷ (نسبت ≈ ۲٫۱٤)

اگر greedy باشیم، اول X را انتخاب می‌کنیم (وزن ۳، ارزش ۱۰). ٤ واحد فضا باقی می‌ماند. حالا باید بین Y (وزن ٤، ارزش ۱۲) و W (وزن ۷، ارزش ۱۵) یکی را انتخاب کنیم. Y را انتخاب می‌کنیم. مجموع وزن ۷، ارزش ۲۲.

اما **جواب بهینه** این است که فقط شیرینی W را انتخاب کنیم (وزن ۷، ارزش ۱۵). این نشان می‌دهد که انتخاب اولیه X (با نسبت بالاتر) ما را از رسیدن به ترکیب بهینه باز داشت. در مسائل ۱/۰، انتخاب یک آیتم ممکن است مانع برداشتن ترکیب بهتری از آیتم‌های دیگر شود، حتی اگر آن آیتم اولیه نسبت بالاتری داشته باشد.

بخش ب)

۱. **تعریف حالت** $DP[i][w]$ **و رابطه بازگشتی:**

(الف) **توضیح تعریف** $DP[i][w]$**:**

این تعریف بیانگر **بیشترین ارزش ممکنی** است که می‌توانیم با در نظر گرفتن **فقط i تا شیرینی اول** (از ۱ تا i) و با داشتن **دقیقاً w واحد ظرفیت وزنی** در کوله پشتی، به دست آوریم. این تعریف دقیقاً w مهم است؛ یعنی اگر نتوانستیم با این i شیرینی، دقیقاً w واحد ظرفیت را پر کنیم، مقدار مربوط به آن حالت یا نامعتبر است یا باید با یک مقدار

خاص (مثلاً صفر یا منفی بی‌نهایت) نمایش داده شود تا در محاسبات بعدی لحاظ نشود. این حالت، کل مسئله را به زیرمسئله‌های کوچک‌تر و قابل مدیریت تقسیم می‌کند.

(ب) **دو انتخاب اصلی و رابطه بازگشتی:**

وقتی می‌خواهیم $DP[i][w]$را محاسبه کنیم (یعنی بهترین ارزش با i شیرینی و ظرفیت w)، برای **شیرینی i ام** دو انتخاب اساسی داریم:

- **انتخاب اول (شیرینی i را برنداریم):** اگر شیرینی i را در کوله نگذاریم، بیشترین ارزشی که می‌توانیم داشته باشیم، دقیقاً همان بیشترین ارزشی است که با **i-1 شیرینی اول** و با همان ظرفیت wبه دست می‌آمد. یعنی: $DP[i-1][w]$.
- **انتخاب دوم (شیرینی i را برداریم):** این انتخاب فقط زمانی ممکن است که **وزن شیرینی i (که آن را** $weight[i]$**بنامیم) کمتر یا مساوی ظرفیت باقی‌مانده** w**باشد**. اگر آن را برداریم، ارزش آن $value[i]$را به دست می‌آوریم، اما ظرفیت wبه اندازه $weight[i]$کم می‌شود. پس ارزش کلی ما برابر خواهد بود با: $value[i] + DP[i-1][w - weight[i]]$. (یعنی ارزش شیرینی i ام به اضافه بهترین ارزشی که با i-1 شیرینی قبلی و با ظرفیت باقی‌مانده $w - weight[i]$ به دست می‌آید).

بنابراین، $DP[i][w]$برابر است با **بیشترین** این دو مقدار (یا فقط مقدار اول اگر برداشتن شیرینی i ام ممکن نباشد).

$$DP[i][w] = \max\ (DP[i-1][w], \text{اگر } weight[i] \leq w: value[i] + DP[i-1][w - weight[i]], \text{در غیر این صورت فقط } DP[i-1][w])$$

۲. **ترکیب شیرینی‌ها برای** $W = 11$**و ارزش نهایی:**

بیایید جدول DP را به صورت ذهنی یا روی کاغذ بسازیم. (در اینجا به دلیل محدودیت، فقط مراحل کلیدی را برای رسیدن به جواب نهایی $W = 11$نشان می‌دهیم):

داده‌ها:

- A: ارزش ۱۰، وزن ۳

- B: ارزش ۱۲، وزن ٤
- C: ارزش ۱۸، وزن ٦
- D: ارزش ۲۲، وزن ۷
- W = 11

مراحل کلیدی محاسبه برای $W = 11$:

- **برای $DP[i][w]$ با $w < 3$:** فقط $DP[i][w] = 0$(چون هیچ شیرینی‌ای را نمی‌توان برداشت).
- **شیرینی A (وزن ۳، ارزش ۱۰):**
- $DP[A][3] = 10$
- $DP[A][4] = 10$(چون $3 + 3 < 4$نیست، فقط $DP[A][4] = DP[A][4 - 3] + 10 = 0 + 10 = 10$)
- ...
- $DP[A][11] = 10$(فقط A برداشته شده)
- **شیرینی B (وزن ٤، ارزش ۱۲):**
- $DP[B][4] = \max(DP[A][4], DP[A][0] + 12) = \max(10,0 + 12) = 12$(فقط B)
- $DP[B][7] = \max(DP[A][7], DP[A][3] + 12) = \max(10,10 + 12) = 22$(A + B، وزن ۳+٤=۷، ارزش ۱۰+۱۲=۲۲)
- ...
- $DP[B][11] = \max(DP[A][11], DP[A][7] + 12) = \max(10,10 + 12) = 22$(A + B)
- **شیرینی C (وزن ٦، ارزش ۱۸):**
- $DP[C][6] = \max(DP[B][6], DP[B][0] + 18) = \max(12,0 + 18) = 18$(فقط C)
- $DP[C][9] = \max(DP[B][9], DP[B][3] + 18) = \max(22,10 + 18) = 28$(A + C، وزن ۳+٦=۹، ارزش ۱۰+۱۸=۲۸)
- $DP[C][10] = \max(DP[B][10], DP[B][4] + 18) = \max(22,12 + 18) = 30$(B + C، وزن ٤+٦=۱۰، ارزش ۱۲+۱۸=۳۰)
- $DP[C][11] = \max(DP[B][11], DP[B][5] + 18)$ → DP[B][5] خیلی کم است، حدود ۱۲) $\max(22,12 + 18) = 30$(B + C)
- **شیرینی D (وزن ۷، ارزش ۲۲):**

- $DP[D][7] = \max(DP[C][7], DP[C][0] + 22) = \max(22,0 + 22) =$ 22(فقط D)
- $DP[D][9] = \max(DP[C][9], DP[C][2] + 22) = \max(28,0 + 22) = 28$ (A + C)
- $DP[D][10] = \max(DP[C][10], DP[C][3] + 22) = \max(30,10 + 22) =$ 32(A + D، وزن ۳+۷=۱۰، ارزش ۱۰+۲۲=۳۲)
- $DP[D][11] = \max(DP[C][11], DP[C][4] + 22) = \max(30,12 + 22) =$ 34(B + D، وزن ٤+۷=۱۱، ارزش ۱۲+۲۲=۳٤)

نتیجه:

با بررسی مراحل بالا، بیشترین ارزش قابل دستیابی برای $W = 11$ برابر **۳٤** است.

این ارزش از انتخاب **شیرینی B (وزن ٤، ارزش ۱۲)** و **شیرینی D (وزن ۷، ارزش ۲۲)** به دست می‌آید. مجموع وزن آن‌ها $7 + 4 = 11$است که دقیقاً برابر ظرفیت کوله است.

مقایسه‌های کلیدی برای رسیدن به $DP[D][11] = 34$:

- ابتدا $DP[C][11] = 30$بود (از ترکیب B+C).
- حالا برای $DP[D][11]$، دو گزینه داریم:

۱. شیرینی D را برنداریم: ارزش برابر $DP[C][11] = 30$.
۲. شیرینی D را برداریم (چون وزن ۷ <= ۱۱): ارزش برابر $value[D] + DP[C][11 - 7]$یعنی $22 + DP[C][4]$.

- برای محاسبه $DP[C][4]$:
- C را برنداریم: $DP[B][4] = 12$.
- C را برداریم (وزن ٦ > ٤): ممکن نیست.
- پس $DP[C][4] = 12$.
- بنابراین، ارزش با برداشتن D برابر است با $22 + 12 = 34$.
- چون $34 > 30$، پس $DP[D][11] = 34$.

سوال ۵:

بررسی کد هافمن :

به کمک داده‌ها، کدهای هافمن رو می‌سازیم. کد هر کاراکتر معلومه، و باید دقت کنیم این کدها پیشوند هم نیستن تا بتونیم بدون مشکل پیام‌ها رو باز کنیم.

2.انتخاب پیام‌ها با محدودیت وزن :

وزن هر پیام مشخصه. حالا باید پیام‌هایی رو انتخاب کنیم که مجموع وزنشون از ۱۸۰ بیشتر نشه و ارزش اطلاعاتی که می‌فرستیم (مثلاً نسبت تکرار به طول کد) بیشینه بشه. برای این قسمت از الگوریتم کوله‌پشتی (صفر و یک) استفاده می‌کنیم.

3.محاسبه بلندترین دنباله مشترک (LCS):

دو پیام نمونه داریم. باید بلندترین دنباله‌ای که تو هر دو هست رو پیدا کنیم تا بفهمیم چقدر پیام‌ها به هم شباهت دارن و فشرده‌سازی چقدر خوب بوده.

4.بهینه‌سازی پرداخت هزینه :

هزینه نهایی ارسال پیام‌ها رو حساب می‌کنیم. عددی که می‌رسه باید با کمترین تعداد سکه‌های ۲، ۵ و ۷ ساخته بشه. این کار رو با الگوریتم «خرد کردن پول» انجام می‌دیم تا تعداد سکه‌ها رو کمینه کنیم.

نکات مهم:

-یادتون باشه وزن و هزینه با هم فرق دارن؛ کوله‌پشتی که داریم کسریه یعنی می‌تونیم بخشی از یه پیام رو انتخاب کنیم .

-کدها باید طوری باشن که هر کدی جزئی از کد دیگه‌ای نباشه (پیشوند نباشه) .

-موقع حل مسایل خرد کردن پول هدف، کم کردن تعداد سکه‌هاست نه کم کردن مجموع هزینه که ثابت هست .

سوال ۶ :

تعیین کد هافمن :

با توجه به فراوانی داده‌ها کدها ساخته می‌شن. باید مطمئن شد طول کدها درست باشن و خاصیت پیش‌وندی رعایت شده باشه تا دیکدینگ بدون ابهام باشه.

2.انتخاب داده‌ها با محدودیت انرژی – کوله‌پشتی کسری :

باید تصمیم گرفت از هر داده چقدر بفرستیم طوری که جمع وزن‌ها از ۲۵۰ بیشتر نشه. هدف بیشینه کردن ارزش اطلاعاتی (مثلا نسبت فراوانی به طول کد) هست.

3.محاسبه طول بلندترین دنباله صعودی :(LIS)

روی توالی طول کدهای داده‌های انتخاب شده LIS محاسبه می‌شه تا ترتیب ارسال داده‌ها بهینه بمونه و خطاها کاهش پیدا کنه.

4.بهینه‌سازی پرداخت هزینه :

باید هزینه کل ارسال داده‌ها رو با سکه‌های {۱، ٤، ٦} با کمترین تعداد سکه پرداخت کنیم. با الگوریتم خرد کردن پول (Coin Change) بهترین روش این کار بدست می‌آد.

نکات کلیدی:

-بین وزن ارسال و هزینه پرداخت تفاوت قائل باشین .

-الگوریتم‌های هافمن، کوله‌پشتی، LISو خرد کردن پول باید با هم و به‌صورت همزمان در نظر گرفته بشن .

-دقت کنید خاصیت پیش‌وند بودن کدها برای دیکدینگ ضروری است .

LIS -کمک می‌کنه ترتیب ارسال داده‌ها به‌گونه‌ای باشه که پایین‌ترین نرخ خطا رو داشته باشیم.

خروجی‌های مورد انتظار

-کدهای هافمن داده‌ها
-مقدار هر داده ارسال شده تحت محدودیت انرژی
-طول دنباله LIS روی توالی داده‌های ارسال شده
-کمترین تعداد سکه لازم برای پرداخت هزینه کل

سوال ۷:

الف) چرا باید از min-priority queue استفاده شود؟ اگر از صف معمولی استفاده کنیم چه مشکلی پیش می‌آید؟

۱. ماهیت الگوریتم

در ساخت درخت هافمن، در هر مرحله باید دو گره با کمترین فرکانس انتخاب شوند.
این انتخاب باید بارها و سریع انجام شود.

۲. دلیل استفاده از min-priority queue

ساختار min-priority queue (یا min-heap):

- کمترین عنصر را با Extract-Min در زمان $O(\log\ n)$ برمی‌گرداند.
- درج یک گره جدید نیز با Insert در زمان $O(\log\ n)$ انجام می‌شود.
- پس این ساختار مناسب انتخاب مکرّر «دو مین» است.

۳. مشکل صف/آرایه ی معمولی

اگر از صف معمولی یا آرایه استفاده شود:

- پیدا کردن کمترین عنصر زمان $O(n)$دارد.
- در هر مرحله باید دو بار جستجو انجام دهیم.
- این کار باید حدوداً n بار تکرار شود.

در نتیجه الگوریتم بسیار کند می‌شود و زمان کل به $O(n^2)$می‌رسد.

نتیجه ی بخش (الف):

برای اینکه بتوانیم در هر مرحله دو گره ی کم‌وزن را سریع و درست پیدا و حذف کنیم، استفاده از min-priority queue ضروری است.

ب) پیچیدگی زمانی ساخت کُد هافمن با استفاده از priority-queue چیست؟

تحلیل:

۱. درخت هافمن تقریباً n−1 مرحله ادغام دارد.

۲. در هر مرحله:

- یک Extract-Min
- یک Extract-Min دیگر
- یک Insert

انجام می‌شود.

۳. هرکدام از این عملیات‌ها روی min-heap زمان $O(\log\ n)$دارند.

پس زمان کل:

$$O(n\log\ n)$$

نتیجه ی بخش (ب):

پیچیدگی زمانی ساخت درخت و کد هافمن با priority queue برابر است با:

O(n log n)

سوال ۸ :

(الف) – الگوریتم Kadane روی آرایه A

آرایه:

A = [۴ ,۵- ,۳ ,۱ ,۲- ,۴]

شبه کد الگوریتمKadane

$$
\begin{aligned}
&Kadane(A):\\
&\quad current = A[0]\\
&\quad best = A[0]\\
&\\
&\quad for\ i = 1\ to\ A.length - 1:\\
&\quad\quad current = max(A[i], current + A[i])\\
&\quad\quad best = max(best, current)\\
&\\
&\quad return\ best
\end{aligned}
$$

اجرای مرحله‌به‌مرحله روی آرایه

۱. مقداردهی:

$$
\begin{aligned}
current &= 4\\
best &= 4
\end{aligned}
$$

۲.

$$
\begin{aligned}
i = 1 \rightarrow A[1] &= -2\\
current &= max(-2, 4 + (-2) = 2) = 2\\
best &= max(4, 2) = 4
\end{aligned}
$$

۳.

$$
\begin{aligned}
i = 2 \rightarrow A[2] &= 1\\
current &= max(1, 2 + 1 = 3) = 3\\
best &= max(4, 3) = 4
\end{aligned}
$$

۴.

$$i = 3 \rightarrow A[3] = 3$$
$$current = max(3, 3 + 3 = 6) = 6$$
$$best = max(4, 6) = 6$$

۵.

$$i = 4 \rightarrow A[4] = -5$$
$$current = max(-5, 6 - 5 = 1) = 1$$
$$best = max(6, 1) = 6$$

۶.

$$i = 5 \rightarrow A[5] = 4$$
$$current = max(4, 1 + 4 = 5) = 5$$
$$best = max(6, 5) = 6$$

نتیجهٔ بخش (الف)

بیشترین مجموع زیرآرایهٔ متوالی: 6

که مربوط است به زیرآرایه:

[4, -2, 1, 3]

(در واقع بیشینه از 4 + (-2) + 1+ 3 = 6)

بخش (ب)

هدف:

نوشتن شبه‌کدی که بررسی کند آیا در مجموعهٔ کدها، یک کد پیشوند دیگری است یا نه.

ورودی:

لیستی از رشته‌های باینری مثل ["0","10","110","111"]

خروجی:

True → اگر پیشوندی وجود داشته باشد

False → اگر هیچ کدی پیشوند دیگری نباشد

شبه‌کد تشخیص پیشوند

$CheckPrefixFree(Codes)$:
  $sort\ Codes\ by\ length\ ascending$

  $for\ i\ =\ 0\ to\ Codes.length\ -\ 1$:
    $for\ j\ =\ i+1\ to\ Codes.length\ -\ 1$:
      $if\ Codes[j].startsWith(Codes[i])$:
        $return\ True\ \ //\ prefix\ violation\ found$

  $return\ False$

- ابتدا کدهای کوتاه‌تر را جلو می‌گذاریم.
- اگر هر کدی پیشوند کد دیگری باشد، ساختار هافمن نقض شده است.

نتیجهٔ بخش (ب)

شبه‌کد بالا دقیقاً تشخیص می‌دهد آیا یک کلمهٔ کد پیشوند کلمهٔ دیگر است یا نه.

(در کد هافمن واقعی، نتیجه باید *همیشه False* باشد؛ چون هافمن prefix-free است.)