طراحی الگوریتم ها

نیم‌سال دوم ۹۹-۹۸

دانشکده‌ی مهندسی کامپیوتر

گردآورندگان: امیرحسین قبادی، حمیدرضا کامکاری، مهدی عرفانیان

پاسخ‌نامه تمرین تئوری سری اول

سوالات تحویلی

مسئله‌ی ۱. قباد در سلف!

قباد پس از چند سال تحصیل در رشته‌ی کامپیوتر به این نتیجه رسید که به درد این رشته نمی‌خورد! پس تصمیم گرفت شروع به کار کند. به دلیل خستگی بیش از حد، قباد برای یافتن کار به نزدیک‌ترین محل ممکن مراجعه کرد: سلف دانشگاه.

او پس از مدتی فهمید روزانه n نفر برای صرف غذا به سلف مراجعه می‌کنند و قباد $m > n$ غذا برای پذیرایی از آن‌ها دارد (هر شخص حداکثر می‌تواند یک غذا از سلف بگیرد و هر غذا به یک شخص داده می‌شود). هر مراجعه کننده به سلف یک فاکتور گرسنگی به نام $g(\alpha)$ که $1 \leqslant \alpha \leqslant n$ دارد که حداقل مقدار غذایی است که او با آن سیر می‌شود. همچنین هر غذا یک فاکتور اندازه به نام $s(\beta)$ با شرط $1 \leqslant \beta \leqslant m$ دارد.

هدف قباد، سیر کردن بیشترین تعداد فرد مراجعه کننده است. سیر شدن به معنی این است غذای داده شده به فرد از فرمول $g(\alpha) \leqslant s(\beta)$ پیروی کند. از انجا که قباد در دروس کامپیوتر ضعیف است با دادن یک الگوریتم حریصانه به او کمک کنید به هدف خود برسد. (راه حل خود را ارائه دهید، بهینه بودن الگوریتم خود را اثبات کرده، شبه‌کد و پیچیدگی زمانی آن را بنویسید).

حل. مراجعه کنندگان را به ترتیب گرسنگی سورت می‌کنیم و به مراجعه کننده با بیشترین مقدار گرسنگی نگاه می‌کنیم. اگر بزرگترین غذا او را سیر می‌کرد، بزرگترین غذا را به او می‌دهیم در غیر این صورت کوچک‌ترین غذا را به او می‌دهیم.

برای اثبات بیهنه بودن مسئله را به دو بخش تقسیم می‌کنیم:

- حالت اول:گرسنه‌ترین مراجعه‌کننده با بزرگترین غذا سیر می‌شود.
- حالت دوم: گرسنه‌ترین مراجعه‌کننده با بزرگترین غذا سیر نمی‌شود.

لم۱: فرض کنید $g(1)$ بزرگترین فاکتور گرسنگی برای هر شخص است و $s(1)$ بزرگترین غذا حداقل برابر با $g(1)$ است. در این صورت تخصیص غذا به فرد اپتیمال معادل تخصیص غذای ۱ به مراجعه کننده‌ی ۱ است.

اثبات: فرض کنید $A*$ هر تخصیص اپتیمال است اگر در $A*$ غذای ۱ به مراجعه کننده‌ی ۱ رسیده باشد. اگر در $A*$ هر غذا با شرط $1 \neq \beta$ را به شخص یک بدهیم به این معنی است که غذای ۱ را

به هیچ کس ندادیم یا به شخصی با مشخصه‌ی $\alpha > ۱$ داده ایم. در حالت اول تخصیص جدید به اسم A تعریف می‌کنیم و در آن غذای ۱ را به مراجعه کننده‌ی ۱ می‌دهیم و بقیه‌ی غذاها را مشابه با تخصیص $A*$ می گذاریم . از آنجا که فرد ۱ با غذای ۱ سیر می‌شود و دیگر افراد هم غذایی دارند که قبلا گرفته اند بنابراین تخصیص A هم اپتیمال است.

در حالت دوم تخصیص جدید به اسم A تعریف می‌کنیم و در آن غذای ۱ را به مراجعه کننده‌ی ۱ می‌دهیم و غذای β را به مراجعه کننده‌ی α می‌دهیم و دیگر مراجعه کنندگان نیز غذا هایی را دریافت می‌کنند که در تخصیص قبل گرفته اند. می‌دانیم در تخصیص فعلی مراجعه کننده‌ی ۱ از α گرسنه‌تر است و با غذای یک سیر می‌شود اگر مراجعه‌کننده‌ی α با غذای β سیر شود که تخصیص اپتیمم است و اگر سیر نشود می‌دانیم در صورت هرگونه تغییر در تخصیص مراجعه کننده‌ی ۱ گرسنه می‌ماند بنابراین باز هم تخصیص اپتیمم است.

لم۲: فرض کنید مراجعه کنندگان و غذاهارا به ترتیب نزولی گرسنگی و اندازه سورت کردیم. حال فرض کنید $s(۱) < g(۱)$. فرض کنید در $A*$ غذای m به مراجعه کننده‌ی ۱ داده نشده است پس تخصیصی موجود است که در آن به مراجعه کننده ۱ غذای m داده شده است پس تعداد مراجعه کنندگان α که به آنها غذای β داده شده است و $s(\beta) > g(\alpha)$ در تخصیص موجود حداقل به اندازه‌ی $A*$ است.

اثبات: فرض کنید در $A*$ به مراجعه کننده‌ی ۱ غذای $\beta > m$ داده شده است. در لم ۲ فرض کنیم کوچک ترین غذا (m) به هیچ مراجعه‌کننده‌ای تخصیص داده نشده است و بعد از آن کوچکترین غذا را به مراجعه کننده‌ی شماره ۱ بدهیم می‌بینیم به جز مراجعه کننده‌ی شماره ی یک همه ی تخصیص های $A*$ و A شبیه هم هستند. از انجا ک $s(\beta) \geqslant s(۱) > g(۱)$ مراجعه کننده ی ۱ با هیچ کدام از دو تخصیص راضی نمی‌شود بنابراین $A*$ و A تعداد مراجعه کننده ی سیر برابر دارد. در حالت دیگر غذای m به مراجعه کننده ای با مشخصه ی $\alpha > ۱$ توسط تخصیص $A*$ داده شده است. حال در نظر بگیرید توسط تخصیص A به تمامی مراجعه کنندگان به جز ۱ و α غذا داده است و سپس به مراجعه کننده ی ۱ غذای m و به مراجعه کننده ی α غذای β را داده است. در این حالت می بینیم تعداد مراجعه کننده ی سیر $A*$ و A با هم برابر است و مراجعه کننده ی ۱ در هر دو حالت سیر نیست و اگر مراجعه کننده ی α در تخصیص $A*$ سیر باشد، $g(\alpha) \leqslant s(m) \leqslant s(\beta)$ پس در تخصیص A نیز سیر است بنابر این تعداد مراجعه کننده ی سیر در تخصیص $A*$ و A باهم برابر است.

ثابت می‌شود در هر آزمایش با دو حالت بالا می‌توان برای گرسنه‌ترین فرد تصمیم حریصانه گرفت، آن عضو را از مجموعه حذف کرد و با دو مجموعه مراجعه کننده $n - ۱$ نفره و تعداد غذای $m - ۱$ الگوریتم حریصانه را ادامه داد.

شبه‌کد الگوریتم بالا را در زیر مشاهده می‌کنید.

```
FoodGive(People[1..n],Food[1..m]).
  Sort People by greed , Food by size ,Largest to Smallest.
  I<-1,J<-m.
  for k=1 to N do:
    if sIgk then Assign[K]=I ,I++
    else Assgn[K]=J , J--
```

```
return Assign
```

زمان نیاز برای مرتب سازی اولیه $O(nlogn + mlogm)$ و لوپ داخلی برابر $O(n)$ است پس از آنجایی که $n \geqslant m$ پیچیدگی زمانی کلی برابر $O(mlogm)$ است. $\triangleright$

مسئله‌ی ۲. رشته های شنگدباو

شنگدباو n تا رشته دارد که همگی از کاراکترهای a و b تشکیل شده‌اند. او می‌خواهد این رشته‌ها را به ترتیبی به هم بچسباند به طوری‌که تعداد جفت مکان‌هایی مثل i و j که $i < j$ است و در جایگاه i کاراکتر a ظاهر شده ولی در جایگاه j کاراکتر b ظاهر شده کمینه باشد.

برای مثال اگر دو رشته abb و aab داشته‌باشیم، می‌توانیم $abbaab$ یا $aababb$ را بسازیم: در اولی تعداد این جفت جایگاه‌ها ۵ تا و در دومی ۸ تاست بنابراین شنگدباو حالت اول را ترجیح می‌دهد. فرض کنید جمع طول رشته‌ها برابر S است. الگوریتمی از مرتبه‌زمانی $S + nlogn$ برای پیدا کردن این ترتیب ارائه کنید.

حل. فرض کنید ابتدا به ازای هر رشته این جفت ها را حساب می‌کنیم. محاسبه کردن این جفت‌ها به ازای هر رشته به صورت جداگانه راحت است. کافیست از ابتدای رشته به ترتیب پیمایش کنیم و تعداد a ها از آن جایگاه به قبل را داشته باشیم. هر سری که به یک b رسیدیم، تعداد این جفت‌ها به اندازه تعداد a های قبل از این جایگاه زیاد می‌شود.

حال فرض کنید این تعداد جفت را X بنامیم.

به ازای رشته iام تعداد a ها را برابر a_i و تعداد bها را برابر b_i در نظر می‌گیریم. تعداد این جفت‌ها برابر مقدار زیر است:

$$X + \sum_{i=۱}^{n} b_i * \sum_{j=۱}^{i-۱} a_i$$

ادعا: اگر رشته‌ها را بر حسب $\frac{b_i}{a_i}$ مرتب کنیم آن وقت مقدار عبارت بالا کمینه می‌شود. برای اثبات این ادعا برهان خلف می‌زنیم. فرض کنید ترتیبی از رشته ها وجود دارد که بر حسب $\frac{b_i}{a_i}$ها نزولی نیست ولی مقدار عبارت بالا برایش کمینه است. حال سعی می‌کنیم هر سری دو تا عضو متوالی مثل i و $i+۱$ را که $\frac{b_i}{a_i} < \frac{b_{i+۱}}{a_{i+۱}}$ است جابه‌جا کنیم. با این جابه‌جایی اگر قبلا تعداد جفت‌ها برابر A بود اکنون برابر $A - b_{i+۱} \times a_i + b_i \times a_{i+۱}$ خواهد بود که می‌توانید ببینید اگر $\frac{b_i}{a_i} < \frac{b_{i+۱}}{a_{i+۱}}$ برقرار باشد این مقدار از A کمتر می‌شود. بنابراین ترتیب ثانویه مقدارش کمتر است و به تناقض می‌رسیم.

بنابراین کافیست ابتدا با $O(S)$ مقدار X را بیابیم و سپس رشته‌ها را به ترتیب گفته‌شده با $O(nlogn)$ مرتب کنیم. $\triangleright$

مسئله‌ی ۳. کمبود پارکینگ

در یک پارکینگ تعدادی ماشین و تعدادی محل برای پارک ماشین وجود دارند. در واقع می‌توانید یک پارکینگ را به شکل آرایه‌ای در نظر بگیرید که در هر خانه از آن C به نشانهٔ ماشین یا P به نشانه محل پارک قرار دارد. هر ماشین فقط می‌تواند در یکی از محل‌های پارک ماشین پارک کند ولی رانندگان ترجیح می‌دهند که ماشین خود را حداکثر در شعاع k از محل فعلیش پارک کنند و در صورتی که بیشتر از k فاصلهٔ بین ماشین و محل پارک باشد از پارک کردن ماشین خود منصرف می‌شوند!

الگوریتمی حریصانه برای یافتن بیشترین تعداد ماشینی که در این پارکینگ می‌توان پارک کرد ارائه دهید. برای مثال خروجی برای پارکینگ زیر و شعاع $k = ۲$ برابر ۳ خواهد بود.

$$\{P, P, C, C, P, C\}$$

حل. از راه حل حریصانه برای حل مساله استفاده می‌کنیم؛ ابتدا کوچک‌ترین اندیس P و کوچک‌ترین اندیس C را پیدا می‌کنیم؛ اگر فاصله این دو عدد کوچکتر مساوی k بود ماشین را در این قسمت پارک می‌کنیم و به کوچک‌ترین اندیس P و C جدید می‌رویم؛ اما در صورتی که بزرگ‌تر از k بود، عدد کوچک‌تر بین اندیس‌های P و C را انتخاب می‌کنیم و به جای آن عدد بعدی پس از آن را قرار می‌دهیم. مادامی که P و C وجود داشته باشند این الگوریتم را ادامه می‌دهیم.

مثال بالا را در نظر بگیرید، ابتدا اندیس P برابر صفر و اندیس C برابر ۲ است؛ فاصله این دو کمتر مساوی k است پس ماشین اول در خانهٔ صفرم پارک می‌کند. و اندیس P جدید برابر ۱ و اندیس C جدید برابر ۳ خواهد بود. باز هم فاصلهٔ ماشین و محل پارک کمتر مساوی k است. پس ماشین دوم نیز در محل پارک با اندیس یک پارک خواهد کرد. ماشین سوم نیز به دلیل مشابه در خانه با اندیس چهار پارک می‌کند. شبه‌کد این الگوریتم بالا را در زیر مشاهده می‌کنید:

```
car_parks = indices of 'P's in list
cars = indices of 'C's in list
r = 0 , l = 0
while l < len(car_parks) and r < len(cars):
    # can park the car
    if (abs( car_parks[l] - cars[r] ) <= k):
        res += 1
        l += 1
        r += 1

    # increment the minimum index
    elif car_parks[l] < cars[r]:
        l += 1
    else:
        r += 1

return res
```

▷

سوالات اضافی

مسئله‌ی ۴. تورنمنت کیوان

کیوان دوست خالی‌بند شنگدباو است! او ادعا می‌کند در یک تورنمت فوتبال که دیشب در آن حضور داشته هر دو تیم دو به دو با هم بازی کرده‌اند و تیم iام d_i بار از بقیه برده‌است (توجه کنید در این تورنمنت در هر بازی یک بازنده و یک برنده داریم). شنگدباو چون می‌داند سابقه کیوان خراب است می‌خواهد ادعایش را راستی‌آزمایی کند. به عبارتی می‌خواهد بفهمد آیا تورنمنتی وجود دارد که در آن نفر iام دقیقا d_i بار برده باشد یا خیر.

شنگدباو به الگوریتمی با $O(n^۲)$ برای راستی آزمایی کیوان نیاز دارد زیرا اگر بیشتر از این طول بکشد خوابش می‌برد! به شنگدباو کمک کنید تا الگوریتم مورد نظرش را بیابد.

حل. اولاً باید جمع این d_i ها دقیقا برابر $\frac{n\times(n-۱)}{۲}$ باشد. در ابتدا این را چک می‌کنیم. حال بدون کاستن از کلیت فرض کنید:

$$d_۱ \leqslant d_۲ \leqslant d_۳ \leqslant ... \leqslant d_n$$

سپس ادعا می‌کنیم اگر تورنمنتی با این درجه‌های خروجی وجود داشته‌باشد. آن‌وقت تورنمتی وجود دارد که در آن d_n از $d_۱$، $d_۲$ و ... d_{d_n} برده‌باشد. یعنی اگر x رأسی با بیشترین درجه خروجی (برد) باشد از x تا رأس دیگر که کمترین درجه خروجی را دارند برده است و از دیگران باخته‌است.

برای اثبات این ادعا فرض کنید یالی از n به uای و از vای به n داریم به طوریکه $d_v < d_u$ یعنی یکی از آن مینیمم خروجی‌ها از n برده‌باشد. در این‌صورت چون درجه v از u کمتر است، رأسی مثل w وجود دارد که w از v برده‌باشد و به u باخته باشد. در این‌صورت n, v, w, u یک دور جهتدار به طول ۴ تشکیل می‌دهند. اگر این دور را برعکس کنیم درجه‌های خروجی ثابت باقی می‌مانند. فرض کنید این روند را تا جایی ادامه می‌دهیم که دیگر چنین جفتی وجود نداشته باشد. در اینصورت یعنی n به d_n تا درجه خروجی مینیمم یال دارد.

برای پیاده‌سازی چنین چیزی همه درجه‌ها را به صورت مرتب‌شده هر سری نگه می‌داریم و عضو آخر را حذف می‌کنیم. و از $n - ۱ - d_n$ عضو آخر یک واحد کم می‌کنیم. پس از اینکه کم کردیم دوباره همه مجموعه را مرتب می‌کنیم. توجه کنید می‌توانیم در این حالت خاص $O(n)$ مرتب‌سازی کنیم (چرا؟) این روند را ادامه می‌دهیم اگر وسط کار به تناقضی نخوردیم (مثلا از رأسی که درجه اش صفر است بخواهیم یک واحد کم کنیم یا بزرگترین درجه بیشتر یا مساوی n باشد) کیوان خالی بسته و در غیر اینصورت ممکن است درست بگوید. $\triangleright$

مسئله‌ی ۵. قباد و جایزه‌ی شریف!

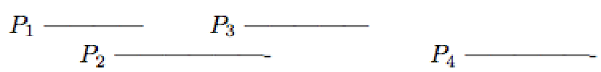
قباد پس از حل چالش‌های کاری‌اش در سلف ماندگار شد اما اکنون با مشکل جدیدی مواجه شده است! سلف تنها در زمان نهار فعالیت دارد و بعد از آن قباد بیکار است و حوصله اش سر می‌رود. او به تازگی متوجه شده است که ایلان ماسک اسپانسر رویداد های زمستان دانشکده شده است و می‌خواهد با قرعه‌کشی میان شرکت‌کنندگان در سمینارها یک نفر را به سفر تفریحی به فضا بفرستد! هر سمینار دارای مدت برگزاری h ساعت است که در صورت شرکت در آن تعداد v بلیط قرعه‌کشی نصیب شرکت کننده می‌شود هرچند شرکت‌کننده می‌تواند سمینار را ترک کند. برای مثال اگر زمان برگزاری سمیناری ۵ ساعت باشد و تعداد بلیط آن ۱۰ باشد، شرکت‌کننده‌ای که بعد از ۳ ساعت سمینار را ترک کند $۶ = ۱۰/۵ * ۳$ بلیط قرعه‌کشی می‌گیرد. قباد علاقه‌ی زیادی دارد که به فضا برود ولی مشکل اینجاست که به حد کافی باهوش نیست تا بفهمد به کدام سمینارها برود تا بیشترین مقدار بلیط را بگیرد و شانس خود را بیشتر کند.

از آنجا که قباد دانشجوی کامپیوتر بوده سراسیمه به دانشکده می‌رود و از هرکسی که درس طراحی الگوریتم را دارد کمک می‌خواهد. به قباد کمک کنید الگوریتمی برای شرکت در سمینارها پیدا کند تا بیشترین مقدار بلیط را به دست بیاورد. از آنجا که قباد بسیار شکاک است و این قرعه‌کشی برایش بسیار مهم است بنابراین اپتیمال بودن روش خود را به او ثابت کنید و پیچیدگی زمانی راه حل خود را به دست آورید.

حل. از الگوریتم حریصانه برای حل این مسئله استفاده می کنیم. در ابتدا برای هر سمینار p_i مقدار چگالی $\bar{v} = \frac{v_i}{h_i}$ را حساب می کنیم و سپس با Flat سورت زمان شروع و پایان هر سمینار را سورت می کنیم و سپس به طور حریصانه در هر قدم اقدام به انتخاب سمینار با بیشترین چگالی می کنیم.

قبل از شروع حل به صورت جدی به مثال زیر دقت کنید:

ورودی مثال به شکل زیر است:



$$P = \{p_۱, p_۲, p_۳, p_۴\},\ \bar{v_۱} = ۲,\ \bar{v_۲} = ۳,\ \bar{v_۳} = ۴,\ \bar{v_۴} = ۲$$

با اجرای فلت سورت داریم:

$$s_۱ < s_۲ < f_۱ < s_۳ < f_۲ < f_۳ < s_۴ < f_۴$$

می‌دانیم s_i و f_i نمایانگر شروع و پایان P_i هستند. با اشاره‌گر بالا اقدام به ساخت درخت AVL می‌کنیم. با اسکن flat list با هربار رسیدن به هر المان (s_i و f_i) اشاره‌گر بالای درخت:

- اگر $Max = Null$ نودی با چگالی المان را وارد درخت کند.

- اگر $Max! = Null$ اختلاف زمان بین المان در حال پردازش و المان قبلی را محاسبه کند، تعداد بلیط ها را وارد کند و المان با چگالی بیشتر را وارد درخت کند.

- اگر المان درحال پردازش به پایان زمان خود رسید تعداد بلیط ها آپدیت شود و نود های متناظر از درخت حذف شوند.

راه مناسب برای بالانس کردن درخت، اجرای بالانس بعد از هر اضافه کردن یا کاستن است. فرض کنیم s جواب مسئله‌ی ماست، برای حل به اخرین متغیر برای ذخیره و اخرین المان در زمان نیاز داریم. در هر زمان تنها یک المان درخت مورد پردازش قرار می‌گیرد:

برای مثال ابتدای سوال داریم:

- در مثال ما در ابتدا s_1 وارد می‌شود، اشاره‌گر بالا $Null$ است پس نود با مقدار چگالی ۲ وارد درخت می‌شود و $Last = s_1$.

- سپس به پردازش s_2 می‌رسیم و $Max! = Null$ پس مقدار S را به صورت زیر به روزرسانی می‌کنیم: $(s_2 - Last) \times max = (s_2 - Last) \times \bar{v_1}$ پس $\bar{v_2}$ را وارد درخت می‌کنیم چون $\bar{v_2} > \bar{v_1}$ و بعد از آن به روزرسانی انجام می‌دهیم: $Last = s_2$.

- سپس به پردازش f_1 می‌رسیم و $Max! = Null$ پس مقدار S را به صورت زیر به روزرسانی می‌کنیم: $(f_1 - Last) \times max = (f_1 - Last) \times \bar{v_2}$ پس $\bar{v_1}$ را از درخت خارج می‌کنیم و اشاره‌گر بالا هم $\bar{v_2}$ می‌ماند و بعد از آن به روزرسانی انجام می‌دهیم: $Last = f_1$.

- سپس به پردازش s_3 می‌رسیم و $Max! = Null$ پس مقدار S را به صورت زیر به روزرسانی می‌کنیم: $(s_3 - Last) \times max = (s_3 - Last) \times \bar{v_2}$ پس $\bar{v_3}$ را وارد درخت می‌کنیم چون $\bar{v_3} > \bar{v_2}$ و بعد از آن به روزرسانی انجام می‌دهیم: $Last = s_3$.

- سپس به پردازش f_2 می‌رسیم و $Max! = Null$ پس مقدار S را به صورت زیر به روزرسانی می‌کنیم: $(f_2 - Last) \times max = (f_2 - Last) \times \bar{v_3}$ پس $\bar{v_2}$ را از درخت خارج می‌کنیم و اشاره‌گر بالا هم $\bar{v_3}$ می‌ماند و بعد از آن به روزرسانی انجام می‌دهیم: $Last = f_2$.

- سپس به پردازش f_3 می‌رسیم و $Max! = Null$ پس مقدار S را به صورت زیر به روزرسانی می‌کنیم: $(f_3 - Last) \times max = (f_3 - Last) \times \bar{v_3}$ پس $\bar{v_3}$ را از درخت خارج می‌کنیم و اشاره‌گر بالا هم ریست شده و صفر می‌شود و بعد از آن به روزرسانی انجام می‌دهیم: $Last = .$.

- s_4 وارد می‌شود، اشاره‌گر بالا $Null$ است پس نود با مقدار چگالی ۲ وارد درخت می‌شود و $Last = s_4$.

در اینجا الگوریتم ما به پایان می‌رسد. برای بالانس کردن درخت به $O(logn)$ عملیات نیاز داریم چون تنها اضافه و حذف داریم. برای سورت کردن $2n$ المان به $O(nlogn)$ عملیات نیاز داریم و چون این کار را برای تمامی المان‌ها انجام می‌دهیم پس پیچیدگی زمانی الگوریتم ما $O(nlogn)$است.

اپتیمال بودن کد با برهان خلف ثابت می‌شود. فرض کنید بازه‌ای موجود است که جواب بهتری نسبت جواب فعلی می‌دهد پس تعداد بلیط اشاره شده در اشاره‌گر باید کمتر از آن باشد و از آن نتیجه می‌گیریم چگالی بازه انتخابی ما کمتر از بازه ی بهینه است ولی این غیر ممکن است چون در هر حالت بیشترین چگالی را انتخاب می کنیم.

```
Initialize T the AVL tree, Max, Last and S
Sort the end point Of P in nondecreasing order -> L will be sorted list
For i = 1 …n :
   ~v = vi / hi
End for

For i = 1 …2n:
   if ai is an sj for some pj then:
      Push ~vj to T and balance T
      if max is Null then:
         Max = ~vj and last = ai
      else
         S = S + (ai - Last) * Max
      if ~vj>max then:
Max = ~vj
Last = ai
else ai is an f(i) for some pj:
   S = S + (ai-Last) * Max
   Remove ~vj From T and balance T
   Update max if necessary
   if max is null then:
   Last = 0

return S
```

▷

موفق باشید :)