علیرضا کریمی ۸۱۰۱۰۱۴۹۲

۱) اندازه مجموعه سکه ها $|V|$ است. در ابتدا لازم است مجموعه را sort کنیم که زمان آن $O(|V| \log |V|)$ است بعد فرض کنید میزانی که میخواهیم پول جمع کنیم S باشد. و فرض کنید داریم:

$$V_n > V_{n-1} > V_{n-2} > V_{n-3} > \dots > V_1$$

در ابتدا به مقدار $\left\lfloor \frac{S}{V_n} \right\rfloor$ از سکه V_n برمی داریم. حال کافی است باقی مانده پول را بقیه سکه ها تکمیل کنیم

باقی مانده پول

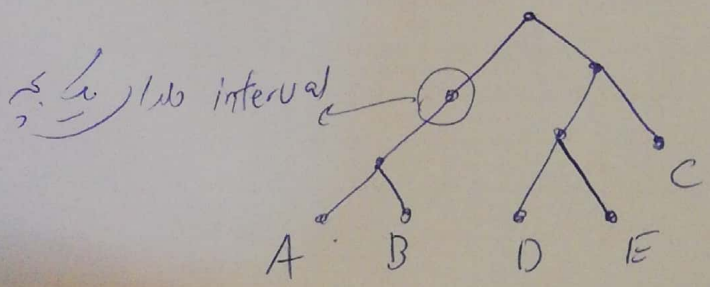
(به قول $S' = S \% V_n$ در زبان برنامه نویسی C++) حال مقدار جدید را در نظر گرفته و به میزان $\left\lfloor \frac{S'}{V_{n-1}} \right\rfloor$ از سکه های نوع V_{n-1} برمی داریم و دوباره داریم $S'' = S' \% V_{n-1}$ و همینطور به همین روند ادامه می دهیم و تا آنجایی که مقدار آنها باید برابر ۱ باشد چون مثلا اگر پول ما ۳ باشد و سکه های ما ۲ تومنی و ۵ تومنی، هیچ وقت پول ما خرد نمی شود). اگر مجموعه را حوصله مشخص کنیم می توانیم مجموعه های مختلفی را انتخاب کنیم اما انتخاب اینکه این الگوریتم حریصانه روی آن مجموعه درست است یک مسئله NP و open است. ولی اگر حوصله انتخاب کنیم مثلا می توانیم ۱ و ۵، ۱۰، ۲۵ را انتخاب کنیم که در کتاب CLRS اثبات شده می توان با آنها این الگوریتم حریصانه را اجرا کرد (یعنی هر دفعه با بزرگترین سکه ممکن پولمان را خرد کنیم.). در هر مرحله این خرد کردن از $O(1)$ است و چون $|V|$ نوع سکه داریم و $|V|$ بار این اتفاق رخ می دهد لذا $O(|V|)$ داریم که پیچیدگی کل برابر با $O(|V| \log |V|) + O(|V|) = O(|V| \log |V|)$ می شود.

۲) برای بعضی مجموعه ها این الگوریتم جواب نمی دهد مثلا $\{1, 3, 4\}$ را در نظر می گیریم و می خواهیم ۶ را خرد کنیم طبق الگوریتم ۴ و ۱ و ۱ را انتخاب می کنیم ولی با ۳ و ۳ هم قابل خرد شدن است

(۲

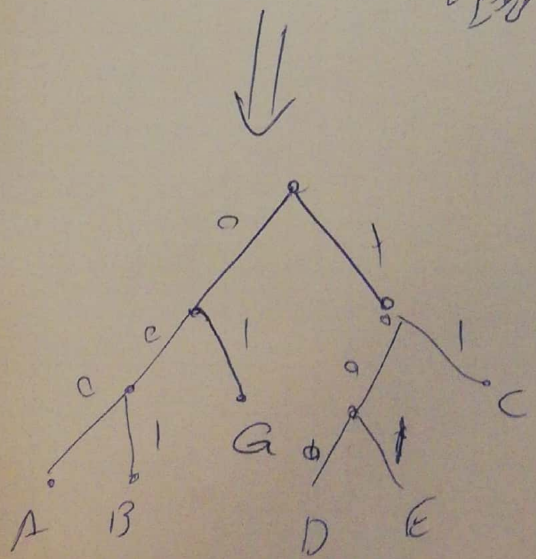
ب و الف) طبق صحبت با TA، ابتدا باید لازم حروف نقطه هاش را پیدا کنیم که هزینه آن $O(n \log n)$ است و هزینه ساخت آن را در نظر نمی‌گیریم و فرض می‌کنیم آن را داریم. حال مثال یک node برگردیم که interval است، یعنی راس میانی است که فقط یک بچه دارد و دقیقا یک بچه ندارد، چون در هافمن هر interval node باید دو تا بچه داشته باشد. و اگر دیدیم که یک بچه دارد فقط، یعنی آن یک بچه دیگر را گذشته نداریم و همان است که دنبالش برگردیم و آن را آن جا اضافه کنیم. برای پیدا کردن این node، می‌توان BFS (و حتی DFS) زدیم و اولین node که برگ نبود و یک بچه داشت، آن حرف را به عنوان بچه دوم آن اضافه کنیم. زمان BFS از مرتبه $O(V+E)$ است که خطی است نسبت به ورودی

مثال



کد صبت مشه

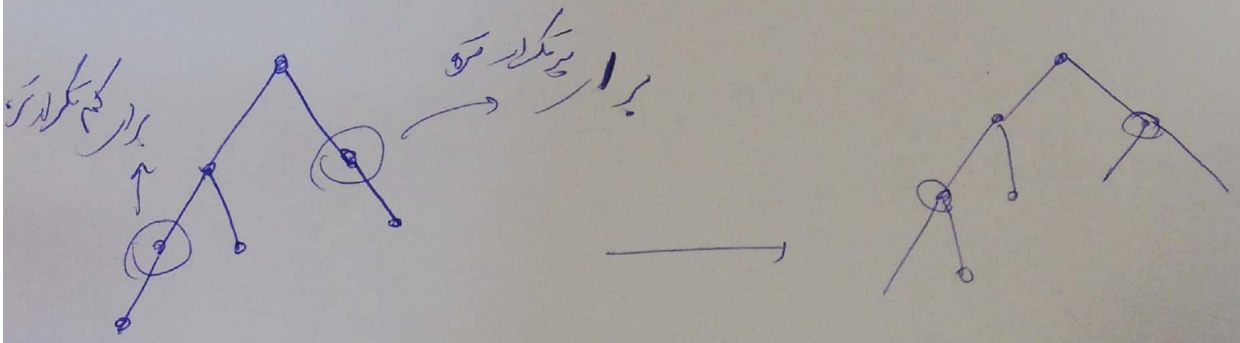
برای پیدا کردن کد G هم از بالا (root) برگشتیم تا G برسیم و آن را بنویسیم و پیدا کنیم



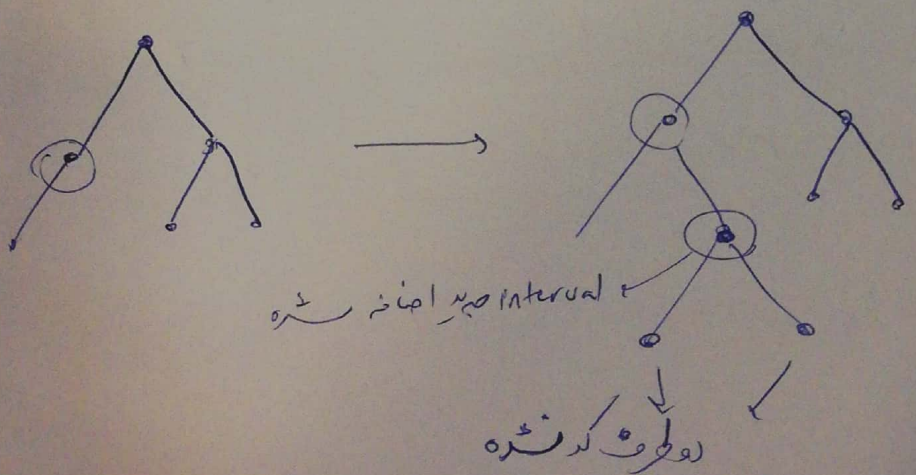
$\Rightarrow G = 011$

ج) (۲ BFS می‌زنیم، دو حالت به وجود می‌آید،

حالت اول) یا دو تا interval node پیدا می‌کنیم که یک فرزند دارند فقط و هر کدوم از آن‌ها دو تا حرف را به عنوان فرزند دوستون قرار می‌دهیم (البته نسبت به اولویت تکرار اون دو حرف، اول حرفی که تکرار بیشتری داشته، در بخش اول و بالاتر درخت قرار می‌گیرد.



حالت دوم) یک interval node پیدا کنیم که تک فرزند دارد. فقط در این حالت یعنی آن دو حرفی که کدشون پاک شده با هر sibling بودن، پس به اون interval، یک فرزند اضافه می‌کنیم و به اون فرزند (که یک interval خواهد شد) دو تا حرفمان را اضافه می‌کنیم به عنوان فرزندان آن



3) ابتدا N_i ها را sort میکنیم از بزرگترین به کوچکترین و داریم:

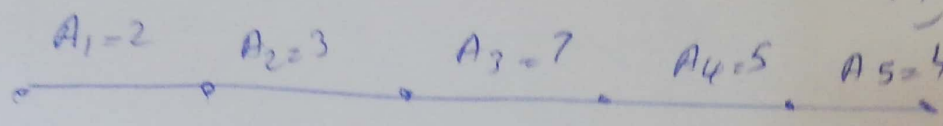
$$N_i > N_{i-1} > N_{i-2} > \cdots > N_1$$

حال از بزرگترین N_i تا کوچکترین بررسی میکنیم. سودوکد آن به این شکل است

```
sum = 0
for int i = k to 1        // آخرین ... اولین
    if sum + Si <= Ni
    {
        sum += Si;
    }
else
    return max(sum, (Ni) + 1);
return sum
```

اگر S_i کوچکتر از N_{i+1} باشد، S_i تا شیر در روز اول میزاریم دور اژدها تا تا قبل از اینکه اصلا شروع به تولید کند، خورده شود. برای بعدی چک میکنیم اگر S_{i-1} تا برای آن بذاریم و S_i تا هم اضافه از قبل داشتیم، آیا مجموع آن‌ها کمتر از $(N_{i-1} + 1)$ است یا نه. اگر کمتر بود پس یعنی خوبه، ما با شیرهای کمتر از N_i و N_{i-1} توستیم اژدهای اول و دوم رو نابود کنیم. اما اگر $S_i + S_{i-1} \geq N_{i+1}$ شد، S_i و N_{i+1} را return میکنیم. یعنی چی، یعنی اگر N_{i+1} از S_i بیشتر بود اول S_i تا برای اژدهای بعدی، برای روزهای بعد همه شیرها را تجمیع میکنیم و به بزرگترین اژدها (منظور بیشترین rate تولید را هم نابود میکنیم) چون مرگ از تولد سریعتر است، نابود میشود. اگر هم $S_i > N_{i+1}$ دوباره همین

4) مسئله‌ی ما حل شده است اگر $[\max_{A_i}, Sam_A]$ را با binary search بزنیم. A_i فاصله‌ی بال i ام است. مثلا داریم:



$\max A_i = 7$ ← یعنی اگه زمان ما کمتر از 7 باشه، هیچ وقت نمی‌تونیم این بال رو طی کنیم. پس حداقل باید این باشه. حداکثر هم مجموع زمان تمام بال‌هاست.

حال برای هر عدد که می‌زنیم چک می‌کنیم. مثلا عدد K. چک می‌کنیم که چند تا بال رو می‌تونیم کامل طی کنیم. مثلا اگر $K = 8$ باشه، دو بال اول کامل طی می‌شه و فقط تا وسط بال سوم می‌تونیم بریم پس دوباره از ابتدای بال سوم restart می‌شیم (یعنی فقط بال اول و نیم طی شدن) حال دوباره این کار رو می‌کنیم. در مرحله دوم فقط بال سوم طی می‌شه، در مرحله سوم فقط بال چهارم و در مرحله 5، بال 5. پس 5 مرحله طول کشید. یعنی به حداقل 5 تیم نیاز داریم. اگر 5 از تعداد گروه مطلوب ما بیشتر بود، یعنی زمان را باید بیشتر کنیم. در غیر اینصورت اگر از تعداد گروه مطلوب ما کمتر شد، یعنی زمان زیادی به دو تیم دادیم، پس احتمال می‌تونیم زمان را کمتر کنیم. این کار را طبق الگوریتم binary search انجام می‌دهیم.

(5

و دانیم که دوست داریم b_iهای کوچک‌تر در ابتدا بیایند و از طرفی هم دوست داریم که P_iهای کوچک‌تر در آخر باشند و هم‌چنین P_iهای بزرگ‌تر دیرتر از ابتدا حرکت بگیرند. پس با شهود می‌توان گفت اونی که P_i بیشتر و b_i کمتر را دارد دوست داریم در ابتدا بیاید. پس مقدار $\frac{P_i}{b_i}$ مهم است. حال تمام فایل‌ها را طبق $\frac{P_i}{b_i}$ sort می‌کنیم. پس اولین فایل بیشترین $\frac{P_i}{b_i}$ را دارد و آخرین فایل کمترین $\frac{P_i}{b_i}$ را دارد. این روش greedy ماست. حال باید اثبات کنیم که این روش بهینه است. قبل از آن هم باید بگویم پیچیدگی این کار $O(n \log n)$ است.

اثبات درستی: فرض می‌کنیم یک OPT وجود دارد که جواب بهینه است اما جواب حریصانه نیست ولی بین جواب‌های OPT بیشترین شباهت را به جواب greedy دارد. شباهت در این است که بیشترین تعداد Pair از ابتدای مکان را دارد که این رابطه برقرار است را انتخاب می‌کنیم. حال اولین Pair از opt که $\frac{P_i}{b_i} > \frac{P_{i+1}}{b_{i+1}}$

$$\text{greedy}: \quad \frac{P_1}{b_1} > \frac{P_2}{b_2} > \frac{P_3}{b_3} > \cdots > \frac{P_i}{b_i} > \frac{P_{i+1}}{b_{i+1}} > \cdots > \frac{P_n}{b_n}$$

$$\text{opt}: \quad > \quad > \quad > \quad > \frac{P_i'}{b_i'} \leq \frac{P_{i+1}'}{b_{i+1}}$$

↓

اولین جایی که کوچک‌تر از بعدی است.

ادعا می‌کنیم در صورت جابجایی $\frac{P_i'}{b_i'}$ با $\frac{P_{i+1}'}{b_{i+1}}$ به یک جواب بهتر و یا شبیه‌تر به greedy می‌رسیم و اینگونه فرض خلف ما باطل می‌شود و حکم ثابت.

حال طبق هر کدام را انتخاب می‌کنیم.

opt: P_t (t) | P_i' (b_i') | P_{i+1}' (b_{i+1}) | بقیه

opt': P_t (t) | P_{i+1}' (b_{i+1}) | P_i' (b_i') | بقیه

(برای t تا اول به طور مثال P_t را برابر با راستی در نظر می‌گیریم)

(از بعد این سه بخش را تغییر نمی‌دهیم)

از اول $P_t(t) + P_i'(t + b_i') + P_{i+1}(t + b_i' + b_{i+1})$

از دوم $P_t(t) + P_{i+1}'(t + b_{i+1}') + P_i'(t + b_{i+1}' + b_i')$

ادامه 5)

مقایسه اول و دوم

دوم: $p_i' b_{i+1}'$ ○ $p_{i+1}' b_i'$:اول

$\Rightarrow$

$$\frac{p_{i+1}'}{b_{i+1}'} \bigcirc \frac{p_i'}{b_i'}$$

$\Rightarrow$

می دانیم که $\frac{p_{i+1}'}{b_{i+1}'} \geq \frac{p_i'}{b_i'}$ پس می توان فهمید که عبارت دوم کوچکتر از عبارت اول است. اگر کوچکتر باشد که یعنی opt بهتر از opt است و opt اصلا جواب بهینه نبوده که تناقض است، اگر هم مساوی باشد یعنی به یک جواب بهینه تر رسیدیم (یادآوری شباهت دراولین اندیس بود که sort شده بود یعنی بیشترین هر دو اولین اندیس در آن یا به یعنی شبیه تره به greedy) و البته این کار را می توانیم به طور مداوم انجام دهیم، عوض کردن همان pair را می کنیم (یعنی مقایسه $\frac{p_{i+1}'}{b_{i+1}'}$ یا $\frac{p_{i-1}'}{b_{i-1}'}$ اگر sort برگرداند) و می به جواب بهتر برسیم. پس اول opt که گفتیم بهینه ترین بود که تناقضه باز

۲) راهنمایی غلط، مثال نقض:

پروانه‌ها (موقعیت‌شون): 0 3

گل‌ها (موقعیت‌شون): 2 5

راه حلی: (3 و 2) و (0 و 5) انتخاب میشن که میشه 6 در مجموع

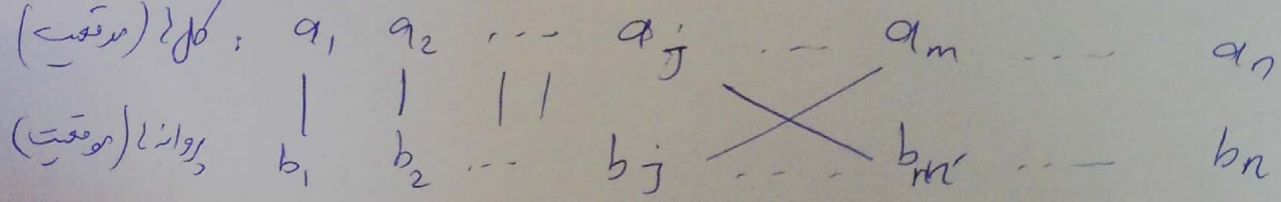
راه بهینه: (2 و 0) و (3 و 5) باهم انتخاب میشن که مجموع فاصلشون میشه 4 و کمتر از 6 است.

راه درست: پروانه‌ها را sort کنیم. گل‌ها را هم sort کنیم. به ترتیب گل نارنجی و پروانه نارنجی را با هم match کنیم:

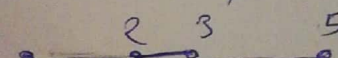
گل‌ها: a_1 a_2 a_3 a_4 ... a_i ... a_n

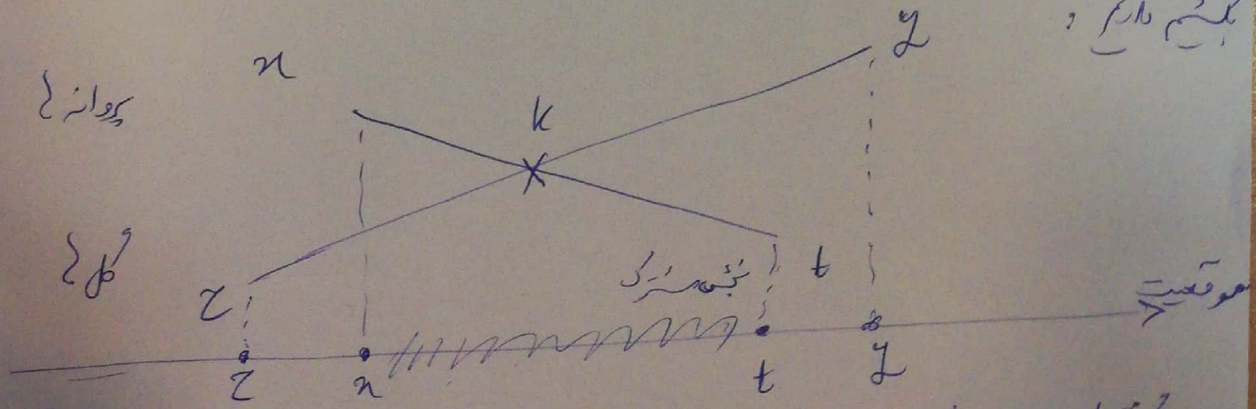
پروانه‌ها: b_1 b_2 b_3 b_4 ... b_i ... b_n

اثبات با برهان خلف: فرض کنیم یک جواب بهینه OPT وجود دارد که با greedy فرق داره اما از بین تمام OPT های دیگر بیشترین شباهت رو به greedy داره، یعنی بیشترین match ابتدایی (a_i و b_i) را دارد. حال فرض کنید اولین جایی که فرق میکنه در اندیس j در نظر میگیریم و فرض کنید داریم:



یعنی a_j به جای match شدن با b_j، با پروانه‌ای جلوتر از آن match میشود و همین شکل، گل جلوتر از گل a_j هم باید با بالاخره با b_j match شود. پس اینها در نهایت باهم تلاقی خواهند داشت (مثل همان مثال (5 و 0) و (2,3) در بالا که از ۲ تا ۳ تلاقی داشتن یعنی مثلا اگر مورب بکشیم داریم:





اگر نقطه k را انتخاب کنیم حتما بین تلاقی این دو خواهد افتاد همانطور که میبینید. حال ثابت میکنیم اگر x را match کنیم و y را با t match کنیم، با distance کمتر مساوی میشود

چون k بین y و z قرار دارد، هر دو مثبت یا هر دو منفی می‌شوند صدر در صد

دارم: $d = |x - t| + |y - z|$

$= |(x - k) + (k - t)| + |(y - k) + (k - z)|$

چون k بین x و t است هر دو مثبت یا هر دو منفی می‌شوند

$\Rightarrow d = |x - k| + |k - t| + |y - k| + |k - z|$

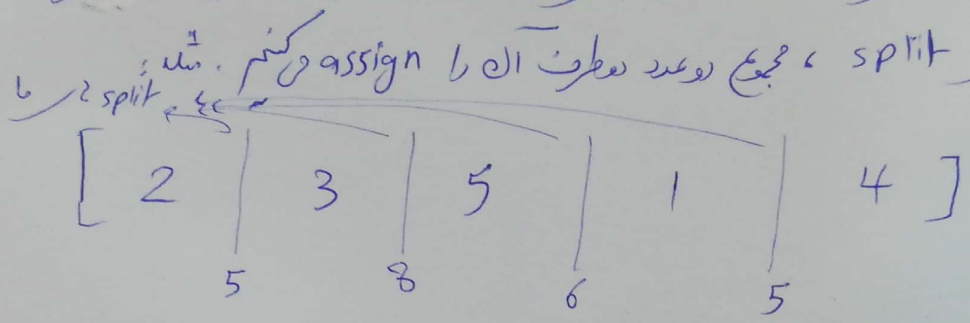
$\Rightarrow d = (|x - k| + |k - z|) + (|k - t| + |y - k|)$

$\Rightarrow d \geq |x - z| + |y - t| = d'$

(ادامه 6)

ادامه سوال 6) سپ نوشتیم فاصلهٔ کوتاه‌تر یا مساوی درست کنیم که اگر فاصله کوتاه‌تر باشد که یعنی به جواب بهتر پیدا کردیم و opt ما بهینه نبوده سپ تناقضه، اگر هم مساوی باشه، چون z_n و z_m هم وصل شدن یعنی a_j و b_j به هم وصل شدن، سپ به جواب شبیه‌تری رسیدیم به greedy سپ opt شبیه‌ترین نبوده و سپ تناقضه درنتیجه بهترین جواب همون جواب greedy است

7) اول چند تا چیز بگم. ما k تا گروه می‌خوایم. پس اگر k-1 تا split بکنیم جدا شدن رو، k تا گروه ساخته میشه و به‌جوری هر split، انتهای بازه سمت چپ و ابتدای بازه سمت راست می‌شه، پس هر دو طرف split در مجموع ما خواهد آمد. پس به هر split، مجموع دو عدد دو طرف آن را assign می‌کنیم. مثلا:



حال ما باید k-1 تا از این split ها انتخاب کنیم. اگر می‌خوایم ماکسیمم مجموع را داشته باشیم برای k تا گروه k-1 تا بزرگترین مقادیر split را انتخاب می‌کنیم به علاوه دو عدد ابتدا و انتها (که تو هر نوع گروه‌بندی تو مجموع ما هستن) و اگر minimum مجموع را می‌خوایم، باید k-1 تا کوچک‌ترین مقادیر split رو انتخاب کنیم (با عدد ابتدا و انتها) و اختلاف این دو می‌شه مجموع k-1 تا بزرگترین منهای مجموع k-1 تا کوچک‌ترین. برای پیدا کردن k-1 تا بزرگترین یا k-1 تا کوچکترین می‌توان از الگوریتم k_select به کمک quick sort استفاده کرد که اردر متوسط آن $O(n)$ و اردر بدترین حالت آن $O(n^2)$ است (و در فصل اول گفته شده) راهی شبیه $T(n) = T(n/2) + O(n) = O(n)$