

۱) از dp استفاده می‌کنیم و تعریف می‌کنیم $dp[u][d]$ یعنی تعداد فرزندان راس u در فاصله (ارتفاع، عمق) d از آن (البته مثلاً می‌توان هر راس را فرزند خودش در ارتفاع صفر در نظر بگیریم.) پس

$dp[u][d]$: تعداد راس‌ها در عمق d از راس u

$$1 \leq u \leq n, 0 \leq d \leq k$$

$$dp[u][0] = 1, \quad dp[u][d] = 0$$

حالت پایه:

$$d \neq 0$$

که خود همان راس u

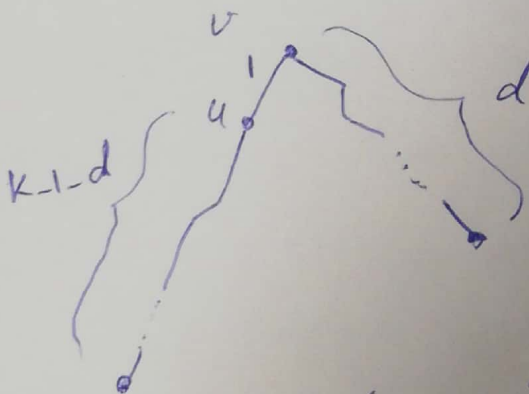
روشن می‌کردن dp به این شکل است که dfs می‌زنیم. زمانی که کارمان با یک راس $child$ تموم شد و خواستیم به $parent$ آن راس برگردیم، این‌ها را نسبتاً را انجام می‌دهیم: (متغیر $result$ تعداد جفت‌های u و v را نگه می‌دارد که فاصله آنها k است. فرض کنید راس پدر را با v و راس فرزند را با u بنامشیم. پس داریم:

for d in range (0 و $k-1$):

$$result += dp[v][d] * dp[u][k-1-d]$$

$O(k)$

که برای d از 0 تا $k-1$



در واقع به صورت مستقیم می‌توانیم:

$$\underbrace{k-1-d}_{\text{فرزند } u} + \underbrace{d}_{\text{فرزند } v} = k-1$$

فاصله u و v

زیردرخت

و بعد از آن باید راس u را به زیردرخت v اضافه کنیم. پس داریم:

for d in range (1, k):

$$dp[v][d] += dp[u][d-1]$$

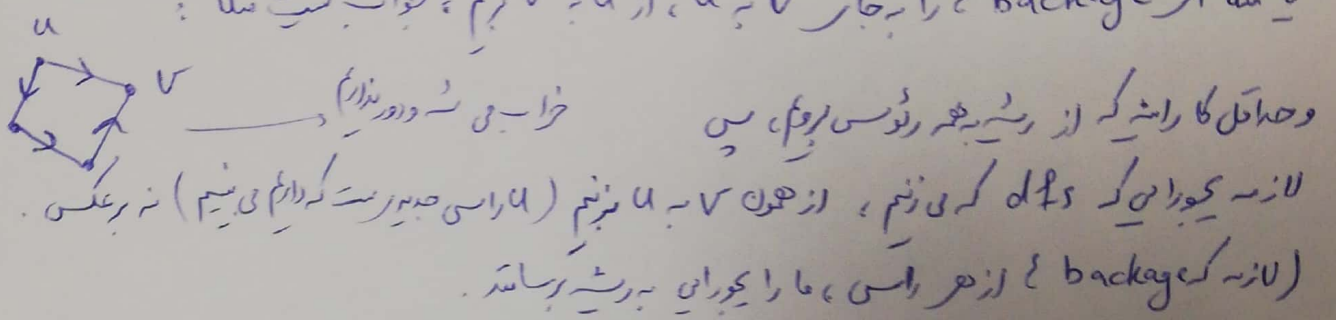
$O(k)$

$$1 \leq d \leq k$$

این کار را تا زمانی انجام می دهیم که تمام راس ها را دیده باشیم. بعد از آن مقدار result را گزارش می کنیم. از آنجا که هر راس را یکبار merge کردیم و هزینه merge کردن آن به بدنه هم از $O(k)$ بود و n راس داریم \Rightarrow در مجموع این کار از $O(nk)$ است. البته چون $d \leq k$ و در نهایت در خود d هم از $O(n) = O(k + v)$ است و جابر نگارنی نیست.

(نکته: مقدار $dp[v][d]$ دارد، از merge شدن یک پر قبلی است یا آن بدست آمده)

(۲) اثبات این سوال: دهمین مسئله در راه حل و رویه DFS میزنیم و به ترتیب
 سبک یالها را آنها را حسب دهمی میزنیم. مثلا وقتی داریم از v به u میرویم، حسب رابطه $u \rightarrow v$
 در نظر میگیریم (چون u را قبلا ندیده باشیم، چون u را قبلا ندیده باشیم و یال ما $backedge$ باشد. کلا
 $backedge$ بخورشان دهند (در است). با DFS وزن یخیز معنی می شود که از ریشه به تمام راسها می
 داریم حداقل. البته اول کار هم می توانیم برابر اینکه معنی شود گراف $connected$ هست یا نه، یک DFS یا BFS
 میزنیم. به چیز دیگری هم که نمیدانیم این که غیر DFS جواب نیست. مثلا اگر BFS میزنیم جواب نیست. مثلا
 یا مثلا اگر $backedge$ را به جابجا v به u ، از u به v میزنیم، جواب نیست. مثلا:



یک نکته هم باید بررسی کنیم که گراف ما یال برشی نداشته باشد چون اگر یال برشی داشته باشد
 قطعا (با هر الگوریتمی) نمی توانیم به چهره حقیقت را بکنیم که هم از u به v هم از v به u داریم. اثبات اینکه یک گراف یال برشی
 دارد یا نه به کمک DP و به کمک DFS قابل انجامه. به این صورت که ما هنگام DFS وزن، چند تا متغیر نگه می داریم.
 یکی $discovery[u]$ که معنی اولین زبانی که راس u رو $visit$ می کنیم. و یکی هم این که وقتی از بچه ها بر اون راس u
 $backedge$ میزنیم - راس u را بر u تر، اون راسی خودر بالاتر هست. که این رو با $lowest[u]$ یعنی بالاترین (کم ارتفاع
 ترین) راسی که قبلا دیده شده، از u ، حال

اگر $lowest[u] < discovery[u]$ یعنی بچه u یا خودش، یک $backedge$ وزن به یک راس بالاتر
 (کم ارتفاع تر) از خودش. پس راس u در دور هست. پس کل یالها را این دور تشکیل شده هیچ کدام $cut edge$ نیست
 اگر $lowest[u] > discovery[u]$ ، یعنی بچه u $backedge$ ان به راسی پایین تر از u رفته و
 در در شرکت نکرد. پس $cut edge$ داریم. این اثبات در CLRS فکر کنیم هست چون مفاهیم $discovery$ و
 غیو او با مطرح شده. اولا به صفتی بعد.

اداره سوال ۱۲ نکته اگر که می‌مونه در آید است کردن lowest است. یعنی lowest [u] چگونه بدست می‌آید که برابر است با

$$\text{lowest}[u] = \min(\text{lowest}[v_i])$$

↓
 v_i فرزندان u هستند

این براساس است که خودش backedge نژده باشد و بچه‌هاش زده باشند. اگر خودش backedge زده باشد اون مقدار هم با این مقدار معادله می‌شه و minimum مقدار (بالاترین ارتفاع) به lowest [u] assign می‌شه اگر cut edge نداشته باشیم طبق این روش، یعنی هر راس در زیر مجموعه خودش که backedge داره به بالاتر از خودش. یعنی غیر از میر که شده از u به v اوسم با dfs (یا v یا u یا u هست)، از طریق بچه‌ها v backedge داریم به u یعنی می‌تونیم از v به u هم بریم از طریق بچه‌هاش.

حال روش ما چه بود؟ این بود که حسب دهن ما، طبق بچاها dfs باشد و اگر از u به v رفتیم حسب $u \rightarrow v$ را می‌کنیم و اگر v را قبلاً دیده بودیم (backedge)، باز هم $v \rightarrow u$ را می‌کنیم. انگونه مطمئن هستیم که از ریشه به همه راس‌ها می‌رویم. طبق این اصل که چک کردیم cut edge نداشته باشیم. اگر cut edge نداشته باشیم یعنی از هر راس به راس بالاتر از خودش هم می‌توانیم از مسیر این backedge برویم. می‌بینی ترتیب به ازای هر راس، می‌توانیم با backedge به راس بالاتر از آن برویم. این کار را انقدر می‌دهیم و بالا می‌رویم تا از هر راس به ریشه برسیم. می‌توانیم از هر راس هم به ریشه برسیم. می‌توانیم بین ریشه و هر راس دیگر پس تمام راس‌ها در دور هستند و هر راس به راس دیگر قابلیت دسترسی دارد.

۱۳) اول به صورت نمودار راه حل را توضیح می دهیم. فرض کنید می خواهیم از راس S به راس t برویم. فرض کنید که راس t دیگر را با u نشان می دهیم. اگر u در تمام مسیر از S به t وجود داشته باشد، یعنی اول از S به u رفته و بعد از u به t رفته. پس تعداد مسیر از S به t برابر حاصل ضرب تعداد مسیر از S به u و تعداد مسیر از u به t است (اصل ضرب). در صورتی که تمام مسیر از u نگذشته باشند، یعنی مسیری وجود دارد که از S به t رفته اما از u نگذشته است پس طبق اصل جمع، در این حالت، تعداد مسیر از S به t بیشتر از حاصل تعداد مسیری است که از S به u و از u به t رفته (که آن هم برابر بود با حاصل ضرب تعداد مسیر از S به u و در تعداد مسیر از u به t). در صورتی که u در هیچ کدام از این مسیر صافی وجود نداشته باشد یعنی u از تمام مسیر از S به t را باز می کشیم، راس u در هیچ کدام یک از آنها نیست.

راه حل: دایکسترا می زنیم و علاوه بر متغیر $dist$ (که نشان دهنده فاصله برد)، متغیر $count$ (که نشان دهنده تعداد مسیر از S به u است) و متغیر $parent$ (که نشان دهنده پدر u است) را هم به ازای هر راس، نگه می داریم. (متغیر $count[i]$ نشان دهنده تعداد مسیر از S به i است و متغیر $parent[i]$ نشان دهنده پدر i است). توجه شود که تا اینجا که صحبت از مسیری کردیم، منظورمان کوتاه ترین مسیری بوده است. راه حل این است که یک بار از S به t دایکسترا می زنیم یک بار از t به S دایکسترا می زنیم. به ازای هر راس ابتدا بررسی می کنیم اگر $dist[u] + dist'[u] \neq dist[t]$ (متغیر $dist'$ ، دایکسترا از t به S و متغیر $dist$ ، دایکسترا از S به t است). یعنی اگر کوتاه ترین فاصله از S به u به علاوه کوتاه ترین فاصله از t به u برابر با کوتاه ترین فاصله از S به t نشد، یعنی اولی راس، در هیچ مسیری از S به t نیست. حال در صورتی که $dist[u] + dist'[u] = dist[t]$ بود، یعنی u در مسیری از S به t وجود دارد، حال باید بررسی کنیم که آیا در تمام مسیر از S به t هست یا در بعضی از مسیر از S به t هست. فرض کنید داشته باشیم که $count[u]$ برابر تعداد مسیری که کوتاه ترین از S به u و $count'[u]$ مادی تعداد مسیری که کوتاه ترین از t به u است. در صورتی که $count[u] * count'[u] = count[t]$ (طبق اصل ضرب که توضیح دادیم) یعنی u در تمام مسیر از S به t هست و در غیر اینصورت یعنی u در بعضی از مسیر از S به t است (یعنی وقتی داریم $count[t] < count[u] * count'[u]$)

ادامه صحنه بعد

حال نکتہ ہم این است کہ چگونه $count[u]$ را بدست آوریم و آن را آپدیت کنیم؟ فرض کنید می خواهیم از S به t را کمترین را بنویسیم. ابتدا $count[u]$ را برابر با 1 رُوس میزنیم و فقط $count[S] = 1$ ، حال طبق الگوریتم را کمترین را بدست می آوریم. درختی آپدیت خواهیم داشت:

```

if dist[v] > dist[u] + w(u,v) {
    dist[v] = dist[u] + w(u,v);
    count[v] = count[u];
}
else if dist[v] == dist[u] + w(u,v) {
    count[v] += count[u];
}

```

معنی تعداد مسیرها را تا u آپدیت می کنیم
 به تعداد مسیرها تا u و از آن هم با $w(u,v)$ (اصد ضرب)
 و $count[v] += count[u]$ →
 یعنی اگر از u هم بیاییم، با همین فاصله به v می رسم. پس طبق اصل جمع تعداد مسیرها را رسیدن به u را به تعداد مسیرهای که
 مثلا به v رسیدیم، اضافه می کنیم.
 اکنون متغیر $count$ را بدست می آوریم. به همین ترتیب می توان با را کمترین از t به S ، متغیر $count$ را نیز
 بدست آورد

(۴) بر اثبات این سوال، ابتدا باید ثابت کنیم عدد هر راس یا برابر l_i یا r_i هست. این اثبات را در آخر به کمک greedy انجام می دهیم. حال فرض کنیم که می دانیم عدد راس i یا r_i یا l_i هست. حال به کمک dp بیشترین زیانی را بدست می آوریم. تعریف می کنیم آرایه $dp[i][2]$ را که $dp[i][0]$ یعنی بیشترین زیانی زیر درخت راس i وقتی که r_i باشد و $dp[i][1]$ یعنی بیشترین زیانی زیر درخت راس i وقتی که عدد راس i l_i باشد.

$$dp[x][0] = dp[x][1] = 0 \quad \text{هنگامی که } x \in \text{leaves} \quad \text{ابتدا برای برگ ها داریم}$$

حال برای آیدیت داریم:

$$dp[u][0] = \sum_{v \in \text{children}[u]} \max(dp[v][0] + |r_u - r_v|, dp[v][1] + |r_u - l_v|)$$

$$dp[u][1] = \sum_{v \in \text{children}[u]} \max(dp[v][0] + |l_u - r_v|, dp[v][1] + |l_u - l_v|)$$

می دانیم هر راس، فرزند یک پدر است. پس هر راسی یک بار در این میانه خواهد آمد. پس در درخت میانه ها $O(n)$ است. (چون درخت هست، در درخت میانه درخت هم برابر $O(n)$ است) $O(V+E) = O(n)$ است. حال باید برابر $E = V - 1$

$$\text{result} = \max(dp[\text{root}][0], dp[\text{root}][1])$$

حال باید ثابت کنیم که چرا عدد هر راس را فقط r_i یا l_i گرفتیم و مقدار میانی این دو را نگرفتیم. همانطور که گفتیم روش greedy به ما گفت که فقط r_i هست یا l_i . فرض کنید یک جواب

OPT داریم که greedy نیست. یعنی یک عدد راس $r_i < a_i < l_i$ داریم ولی این جواب OPT بهترین شباهت را به جواب greedy ما دارد که یعنی بیشترین تعداد راس i را دارد که مقادیر آن r_i یا l_i هستند یا l_i . (واقع

داریم، $a_1, a_2, a_3, \dots, a_i, \dots, a_n$: جواب greedy

OPT " $a'_1, a'_2, a'_3, \dots, a'_i, \dots, a'_n$

که در آن a'_i اونی جایی هست که نه r_i هست و نه l_i (قبلی یا r_i هست یا l_i)

می دانیم که این راس i یک سری یال دارد به همایه i در خودش. حال نزدیک ترین عدد از همایه i به a_i را پیدا می کنیم. هم نزدیک ترین از چپ (ϵ_l) و هم نزدیک ترین از راست (ϵ_r) از همایه i ، تعداد چپ a_i و تعداد راست a_i را هم پیدا می کنیم. می دانیم که اگر عدد از همایه i برابر همان a_i بود، اگر آن را (a_i) می به چپ حرکت می دهیم چپ به راست، $a_i - a_i = 0$ از صفر بزرگتر خواهد شد و به مقدار زیادی ریف افزایش خواهد شد. حال اگر داشته باشیم تعداد اعداد که چپ a_i هست، از تعداد اعداد که دور a_i یا راست a_i هست، کمتر باشد، یعنی

$$\text{if } \text{equal}[a_i] + \text{greater}[a_i] \geq \text{lower}[a_i]$$

حال در این صورت می توانیم جواب OPT را با این به این صورت که OPT هست فقط برای راس i که به چپ a_i می گذاریم (ϵ_l) تا به چپ i (از آجایی که داریم):

$$0 < (\text{equal}[a_i] + \text{greater}[a_i] - \text{lower}[a_i]) \epsilon_l$$

پس به زیاده ریف اضافه شد. پس OPT جواب بهینه نبود. پس فرض خلف باطل است. حال چون تمام $\text{equal}[a_i]$ ، $\text{greater}[a_i]$ و $\text{lower}[a_i]$ در $\text{greater}[a_i - \epsilon_l]$ هستی پس بزرگ داریم:

$$\text{equal}[a_i - \epsilon_l] + \text{greater}[a_i - \epsilon_l] > \text{lower}[a_i - \epsilon_l]$$

این کار را مقدار i را به $i - \epsilon_l$ می دهیم و مقدار چپ i را به $i - \epsilon_l$ می دهیم. هم جواب بهتر ساخته می شود جواب ما به greedy شبیه تر شد. در صورتی که از همان اول داشتیم:

$$\text{if } \text{equal}[a_i] + \text{lower}[a_i] > \text{greater}[a_i]$$

آنگاه مثل همین عمل را انجام می دهیم، فقط نه آجایی راست می ریفیم به اندازه ϵ_r . یعنی می ریفیم به عدد $a_i + \epsilon_r$. و این کار را مقدار i را به $i + \epsilon_r$ می دهیم. هر دفعه به جواب بهتر نزدیک تر به greedy می برسیم. در یک حالت استثنای که $\text{greater}[a_i] = \text{lower}[a_i]$ و $\text{equal}[a_i] = 0$ ، فرض نمی کند به کدام سمت برویم. به یک سمت انتخاب می دهیم چون به همان میزان که ϵ کم می شود به همان میزان هم ϵ به جواب اضافه می شود. این کار تا زمانی که به i یا $i + \epsilon_r$ برسیم انجام می دهیم. شاید جواب بهتر نگرفته باشیم اما جواب ما به greedy شبیه تر شد. پس OPT شبیه ترین جواب به greedy بود پس فرض خلف باطل و حکم greedy ثابت شد.

(۵) کم جایش ترین میر، همان کوتاه ترین میر است. ما به ترتیب می دانیم که (فرضی آفاق، فرضی آفاق، ... تا n این آفاق که حذف می شوند، کدام آفاق هستند؟ فرض کنید که $n-1$ امین آفاق را حذف کنیم. الان فقط می توانیم از آفاق n استفاده کنیم. زمانی که داشتیم آفاق $n-2$ ام را حذف می کردیم، فقط می توانستیم از آفاق n و $n-1$ ام استفاده کنیم. پس به گونه ای که ما داریم از الگوریتم فعلی وراثت استفاده می کنیم اما به صورت برعکس. یعنی اگر از آخر به اول در نظر بگیریم، اول فقط n ام را داریم، بعد می توانیم از آفاق $n-1$ ام هم در میرمان استفاده کنیم، بعد می توانیم از آفاق $n-2$ ام هم استفاده کنیم و تا آخر. پس این قسمت الگوریتم فعلی وراثت را می توانیم به این صورت نوشت

استفاده از الگوریتم تغییر بدیم: یعنی هیچ را نمی استفاده نکردیم $A[i, j, n+1] = w[i, j]$ متغیر دهی

for $k = n$ to 0

for $i = 1$ to n

for $j = 1$ to n

$$A[i, j, k] = \min(A[i, j, k+1], A[i, k, k+1] + A[k, j, k+1])$$

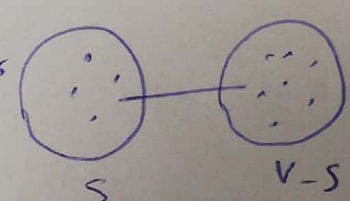
البته به جای این کار را می توانستیم اندکی گذار میگویند. یعنی n امین خانه حذف شده را 1 ، $n-1$ امین خانه حذف شده را 2 و ... قرار دهیم و بعد همان فعلی وراثت را اجرا کنیم. این روش فکر کنیم راحت تره. بعد مثلا $A[i, j, k]$ یعنی تا مرحله $n-k$ حل می کنیم و تنها k تا آفاق داریم که می توانیم از آن استفاده کنیم.

(۹) ابتدا راس‌ها را طبق عدد جابجایی شدن sort می‌کنیم $\xrightarrow{\text{merge sort}}$ $n \log n$
 بعد بر هر راس یک disjoint set جدا دقت می‌کنیم و ابتدا. قبل از شروع اثباتم بگذارید بگویم که ما می‌دانیم:

$$\text{result} = \sum_{\text{Path}} \max(\text{Path}) - \min(\text{Path}) = \sum_{\text{Path}} \max(\text{Path}) - \sum_{\text{Path}} \min(\text{Path})$$

یعنی ما می‌خواهیم بررسی کنیم که هر راسی که می‌خواهیم به عنوان \max آمده و اینها را با هم جمع کنیم. از طرف دیگر می‌خواهیم بررسی کنیم هر راسی که می‌خواهیم به عنوان \min آمده و اینها را از جواب نهایی کم کنیم. اول بگویم برابر می‌باشد $\sum \max(\text{Path})$ ، یعنی جمع کردن \max در هر مسیر. برابر این کار از disjoint set استفاده می‌کنیم و آرایه sort شده عدد جابجایی می‌دانیم که مثلاً یک راس است و در راس دیگر می‌دانیم که اگر طایفه باشیم

، یعنی اگر راس را به دو بخش S و $V-S$ تقسیم کنیم، تعداد



مسیرهایی که از S به $V-S$ می‌رود برابر است با $|S| \times |V-S|$. با دقت گرفتن همه این توصیفات، به سراغ اثبات می‌رویم. (برای اثبات از ChatGPT کمک گرفتیم). ابتدا هر راس را یک disjoint set دقت می‌کنیم. حال، آرایه sort شده عدد جابجایی را طبق صعودی، به نمایش می‌کنیم. به ازای هر راس، اینهایی آن را چک می‌کنیم که قبلاً به نمایش شده یا نه (کافیه مقدار جابجایی شدن از این راس کمتر باشد، چون داریم به صورت صعودی به نمایش می‌کنیم، اون راس قبلاً به نمایش شده). حال فرض کنید داریم راس u را بررسی می‌کنیم و می‌دانیم آن که عدد جابجایی آنی از u کمتر است، راس v است. حال یک سری محاسبات باید انجام دهیم که در ادامه توضیح می‌دهم و بعد از آن disjoint set مربوط به v را به disjoint set مربوط به u اضافه می‌کنیم. در واقع این کاری است که تا آخر انجام می‌دهیم. یعنی اگر راستم برابر u بررسی می‌کردیم و دیدیم که v قبلاً process شده، بعد از یک سری عملیات، disjoint راس v را به disjoint راس u اضافه می‌کنیم پس مطمئن هستیم بزرگترین عدد هر disjoint set عدد جابجایی همان راس است که در آن disjoint set است. حال بگذارید در مورد محاسبات قبل از union بگویم

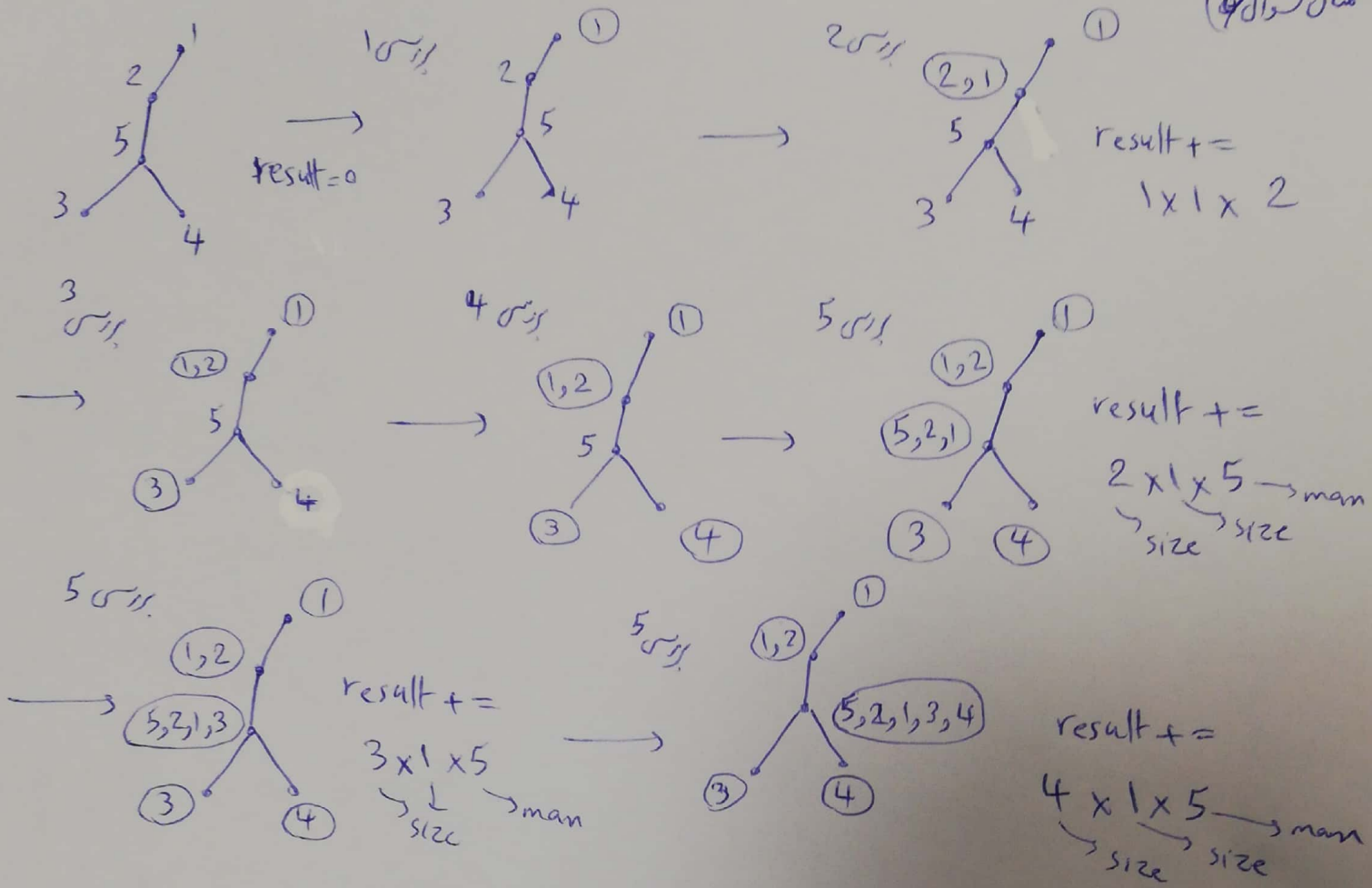
می دانیم که اعضای $disjoint_set$ مربوط به V شامل رأس هر صفحه به V یا مربوط به آن با عدد (یا عددی) جاری
 کمتر است (طبق توضیحات که دادیم). همواره $disjoint_set$ مربوط به u شامل هر یک از $process$ شده اطراف
 است که مقدارشان از u کمتر است. قبول داریم که اگر بخواهیم از هر کدام از اعضای $disjoint_set$ رأس u به هر کدام
 از اعضای $disjoint_set$ رأس u برویم، بزرگترین عدد جاری همان u است! پس u ، ماکسیم مقدار
 در همه این $path$ است که تعداد آن برابر $(disjoint_set[v].size() \times disjoint_set[u].size())$ است.
 پس به مقدار $(disjoint_set[v].size() \times disjoint_set[u].size() \times A[u])$ به جواب نهایی می‌افزایم
 بعد از آن هم $disjoint_set$ مربوط به v را به $disjoint_set$ مربوط به u $union$ می‌کنیم.
 مقدار عدد جاری u است.

این کار را ادامه می‌دهیم تا تمام رأس v را به ترتیب صعودی به این شکل $process$ کنیم. این برابر $\sum_{Path} \max(Path)$
 حال برابر $\sum_{Path} \min(Path)$ ، همین روند را به ترتیب نزولی روی آرایه $sort$ شده عدد جاری انجام
 می‌دهیم و همین روند را طی می‌کنیم. در این حالت مطمئن هستیم که هر رأس u در $disjoint_set$ مربوط به خودش،
 از همه رأس u دیگر کوچک‌تر است. مثل قبل، مقدار زیر را حساب می‌کنیم:

$$A[u] \times disjoint_set[u].size() \times disjoint_set[v].size()$$

اما برخلاف قبل، باید این مقدار را از جواب کلیمان کم کنیم چون داشتیم $\sum_{Path} \min(Path)$.
 بعد هم $disjoint_set$ مربوط به v (همان u است) را که قبلاً $process$ شده و الان داریم با u آن را بررسی می‌کنیم،
 به $disjoint_set$ مربوط به u اضافه می‌کنیم. این کار را هم به صورت نزولی تا جایی که رئیس انجام می‌دهیم

لذا آن جا که به ازای هر u ، (بررسی داریم و مسئله ما رفته است، از $O(n)$ بررسی داریم و چون در هر مرحله
 عملیات $find$ و $union$ انجام می‌دهیم از ما $O(\log n)$ می‌گیرند، پس در کل بار این عملیات از
 $O(n \log n)$ به دلیل خرج می‌کنیم. $sort$ هم $O(n \log n)$ بود. در کل می‌شود $O(n \log n)$.



به این شکل در هر مرحله تا آنجا که می‌توانیم، اگر process شد، بعد قبلاً، با انجام یک سر
حساب، disjoint آن را به خود آن راس اضافه می‌کنیم. برای minimum هم همین کار را انجام می‌دهیم
مستقلاً به صورت نزولی عدد را چاپ می‌کنیم