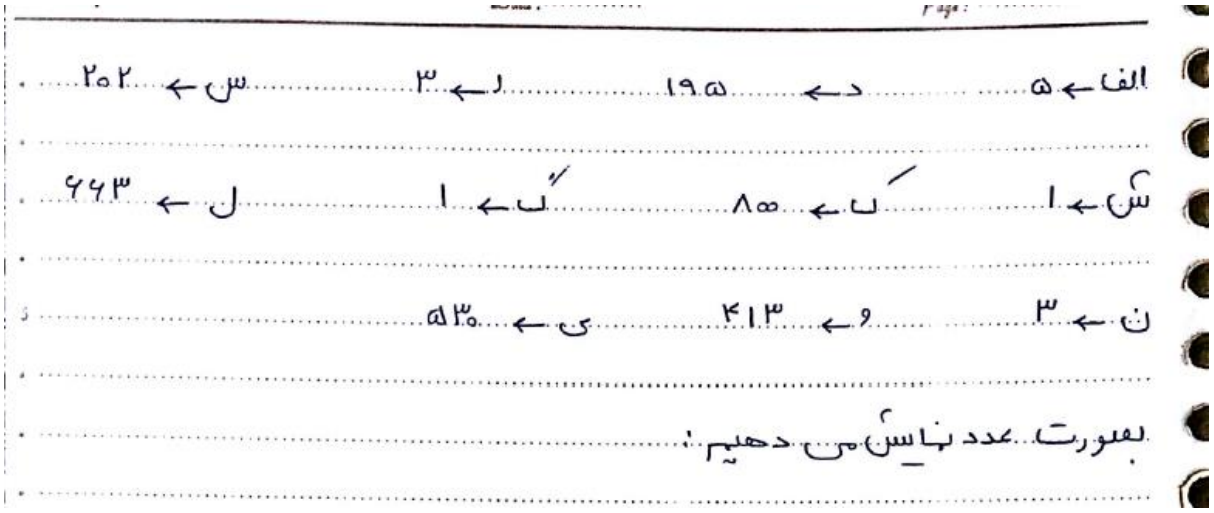
حوری دهش 9821413

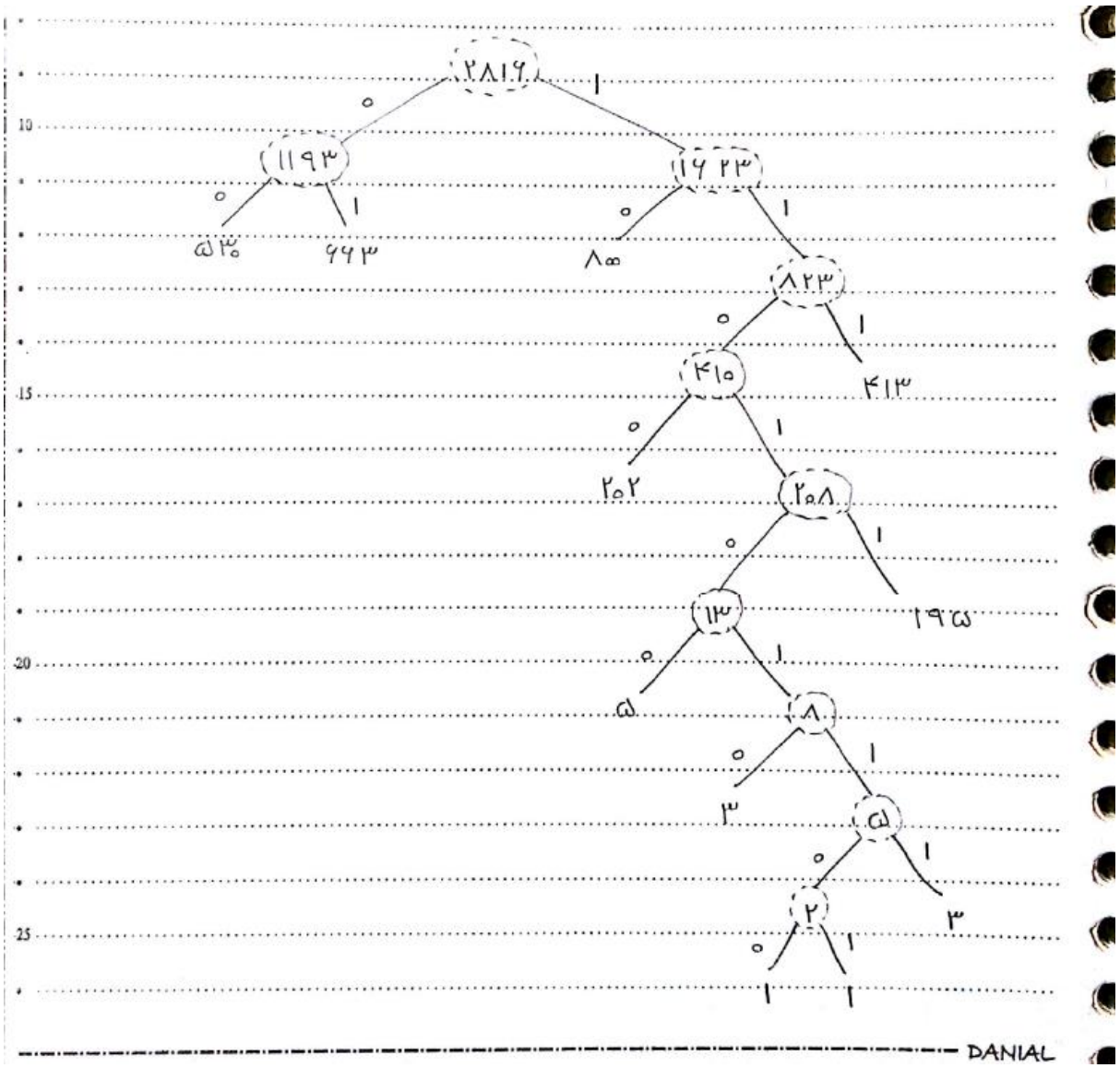
سوال 1:

ابتدا درخت هافمن را می کشیم:

ابتدا نودها رو مرتب میکنیم و از بین نودها دو نود کمتر را انتخاب میکنیم که این برگ ما میشود سپس این دوتا نود را با هم جمع میکنیم و توی این مرحله دوباره نودها رو مرتب میکنیم و باز از بین نودها ان دو نودی که از همه کمتر است را برمیداریم و دوباره با هم جمع می کنیم و به همین صورت ادامه می دهیم تا به ریشه برسیم

درخت نهایی به صورت زیر میشود:





برای این که حداقل تعداد بیت را داشته باشیم می ابیم تعداد تکرار هر حرف را در تعداد بیت هایی که از ریشه تا اون عدد مورد نظر داره ضرب میکنیم و در اخر جمع این هارو به دست می اوریم:

(530*2)+(663*2)+(800*2)+(413*3)+(202*4)+(195*5)+(5*6)+(3*7)+(3*8)+(1*9)+

(1*9) = 7101

سوال 2:

الگوریتم حریصانه ای که ارائه می دهیم به صورت زیر است:

1. تعیین تعداد تکرار هر کاراکتر:

- برای هر کاراکتر در رشته، تعداد تکرار آن را محاسبه می کنیم و در یک جدول یا دیکشنری ذخیره میکنیم.

2. ساخت توالی جدید:

- ایجاد یک لیست یا رشته جدید برای ذخیره ترتیب جدید کاراکترها.

- از کاراکتری شروع میکنیم که بیشترین تعداد تکرار را دارد و آن را به رشته جدید اضافه می کنیم

- سپس کاراکتری را که بیشترین تعداد تکرار را دارد و با آخرین کاراکتر اضافه شده در رشته جدید متفاوت است، پیدا میکنیم و به رشته اضافه میکنیم.

- این عملیات را تا زمانی که همه کاراکترها اضافه شوند یا دیگر نتوان کاراکتری را اضافه کرد ادامه می دهیم.

- در صورتی که نتوانیم کاراکتری را اضافه کنیم به مرحله بعد می رویم.

3. تخصیص باقی مانده:

- در صورتی که تمام کاراکترها به رشته اضافه شوند مسئله حل شده است.

- در غیر این صورت، باید کاراکترهای باقی مانده را به رشته اضافه کنیم.

- برای هر کاراکتر باقی مانده، آن را بین دو کاراکتری که در رشته اضافه شده اند قرار میدهیم که متفاوت از آن‌ها باشد (در صورت امکان).

- این عملیات را تا زمانی انجام می دهیم که تمام کاراکترهای باقی مانده اضافه شوند.

اثبات بهینگی:

برای اثبات بهینگی الگوریتم حریصانه برای تغییر ترتیب کاراکترها به صورتی که هیچ دو کاراکتر متوالی یکسان نباشند، از استقرا روی طول رشته استفاده می کنیم.

فرض کنید S یک رشته با طول n باشد. می خواهیم الگوریتم حریصانه را بر روی این رشته اجرا کنیم تا ترتیبی بدست آید که هیچ دو کاراکتر متوالی یکسان نداشته باشد.

حالت پایه:

اگر طول رشته S برابر با 1 باشد (n =1) ، به وضوح هیچ تغییری در ترتیب کاراکترها نمی توان انجام داد زیرا فقط یک کاراکتر وجود دارد و هیچ دو کاراکتری برای مقایسه وجود ندارد بنابراین، ترتیب اولیه برابر با ترتیب ورودی است و هیچ نیازی به تغییر نیست.

فرض استقرا:

حال فرض الگوریتم حریصانه بر روی رشته‌های با طول کمتر از n کار می کند و ترتیبی بدست می آورد که هیچ دو کاراکتر متوالی یکسان ندارد.

گام استقرا:

حال به مرحله n برای رشته S می رسیم. الگوریتم حریصانه برای رشته S را اجرا می کنیم و ترتیب جدیدی بدست می آوریم که هیچ دو کاراکتر متوالی یکسان ندارند.

اگر آخرین کاراکتر در ترتیب جدید برابر با S[n] باشد، آنگاه دو حالت می توانیم در نظر بگیریم:

1- اگر S[n] در ترتیب اولیه رشته S بود آنگاه S[n-1] نمی تواند در ترتیب جدید کنار S[n] قرار گیرد (به دلیل همسایگی دو کاراکتر یکسان) بنابراین، ترتیب S[n-1] در ترتیب جدید قبل از S[n]قرار می گیرد.

2- اگر S[n] در ترتیب اولیه رشته S نبود آنگاه S[n-1] می تواند در ترتیب جدید کنار S[n] قرار گیرد (بدون وجود همسایگی دو کاراکتر یکسان) بنابراین، ترتیب S[n-1] در ترتیب جدید بعد از S[n] قرار می گیرد.

در هر دو حالت، الگوریتم حریصانه به درستی ترتیب جدید را بدست می آورد که هیچ دو کاراکتر متوالی یکسان ندارند.

بنابراین با استقرا می توانیم نشان دهیم که الگوریتم حریصانه بهینه است و در تغییر ترتیب کاراکترها به صورتی که هیچ دو کاراکتر متوالی یکسان نباشند عملکرد مناسبی دارد.

سوال 3:

ابتدا هر دو ارایه را به صورت صعودی مرتب میکنیم و بعدش مکان سوراخ اول را به مکان موش اول میدهیم به همین صورت می رویم جلو مثلا در همین مثالی که اورده شده مکان موش ها به صورت صعودی میشود:

4 2 4- و مکان سوراخ ها میشود: 5 4 0 در نهایت:

موش در مکان 4- به سوراخ در مکان 0 می رود

موش در مکان 2 به سوراخ در مکان 4 می رود

موش در مکان 4 به سوراخ در مکان 5 می رود

در این حالت کمینه، بیشینه مسافتی که موش ها طی می کنند برابر با 4 میشود.

اثبات بهینگی:

برای اثبات بهینگی الگوریتم حریصانه برای اختصاص سوراخ‌ها به موش‌ها به طوری که بیشینه مسافت طی شده توسط موش‌ها کمینه شود از استقرا بر روی تعداد موش ها استفاده می کنیم.

فرض میکنیم n موش و n سوراخ در اختیار داریم. می خواهیم الگوریتم حریصانه را بر روی این موش‌ها و سوراخ‌ها اجرا کنیم تا اختصاص موش‌ها به سوراخ‌ها را به نحوی انجام دهیم که بیشینه مسافت طی شده توسط موش ها کمینه شود.

حالت پایه:

اگر تنها یک موش و یک سوراخ وجود داشته باشد (n = 1)، بدون در نظر گرفتن هر گونه الگوریتم، مسافت طی شده توسط موش برابر با مسافت بین موش و سوراخ است. بنابراین تخصیص حریصانه در این حالت بهینه است.

فرض استقرا:

فرض میکنیم الگوریتم حریصانه برای n-1 موش و n-1 سوراخ کار کند و تخصیصی را بدست آورد که بیشینه مسافت طی شده توسط موش ها کمینه باشد.

گام استقرا:

به مرحله n برای موش‌ها و سوراخ‌ها می رسیم. الگوریتم حریصانه را اجرا می کنیم و تخصیص جدیدی بدست می آوریم که بیشینه مسافت طی شده توسط موش ها کمینه شود.

فرض می کنیم موش i به سوراخ j اختصاص یابد. در این حالت، تخصیص حریصانه موش i را به سوراخ j در نظر می گیریم این تخصیص بر اساس مسافت بین موش i و سوراخ j انجام می شود. همچنین با توجه به فرض استقرا تخصیص های حریصانه برای n-1 موش و n-1 سوراخ نیز بهینه بوده اند.

حال دو حالت را در نظر می‌گیریم:

1. اگر موش n به سوراخ j اختصاص یابد، مسافت طی شده توسط موش n برابر با مسافت بین موش n و سوراخ j است. با توجه به حالت‌های قبلی، تخصیص موش‌های دیگر به سوراخ ها نیز بهینه بوده است. بنابراین تخصیص حریصانه موش n نیز بهینه است.

2. اگر موش n به سوراخ k (k ≠ j) اختصاص یابد، بنابراین تخصیص حریصانه موش n را به سوراخ j در نظر نمی گیریم با توجه به فرض استقرا، تخصیص های حریصانه برای n-1 موش و n-1 سوراخ نیز بهینه بوده است پس تخصیص حریصانه موش n به سوراخ k نیز بهینه است.

با توجه به دو حالت بالا می‌توانیم نتیجه بگیریم که الگوریتم حریصانه برای n موش و n سوراخ نیز بهینه است و مسافت طی شده توسط موش‌ها را کمینه می کند.

سوال 4:

در ابتدا دو ارایه در نظر میگیریم در یکی از ارایه ها s(j) ها که نشان دهنده سیرکردن هر فرد بود به صورت صعودی مرتب میکنیم و در ارایه دیگر g(i) ها را که نشان دهنده گرسنگی هر فرد بود به صورت نزولی مرتب میکنیم در نهایت از ابتدای هر دو ارایه شروع میکنیم و تا جایی پیش می رویم که مقدار

$\frac{s(j)}{g(i)} >= 1$ شود در این حالت دیگر شخصی گرسنه نیست ولی اگر کسر ما کمتر از 1 شد یعنی هنوز فرد گرسنه است پس به سراغ غذای بعدی می رویم که s(j) بیشتری دارد و برای بقیه افراد هم به همین صورت عمل می کنیم.

اثبات بهینگی:

فرض میکنیم جواب بهینه را برای سیر کردن بیشترین تعداد فرد ممکن داریم و الگوریتم بالا تا i-1 امین نفر عمل می کند اما غذایی که باقی مانده است نمی تواند فرد i ام را سیر کند یعنی g(i)>s(j) است که دو حالت رخ می دهد: یا اصلا از اول غذایی وجود نداشته که فرد i ام را سیر کند پس هر الگوریتم دیگری هم ارائه دهیم باز نمی تواند این فردا را سیر کند یا اینکه غذایی وجود داشته که در این حالت الگوریتمی که ارائه میشود (این الگوریتم باعث میشود الگوریتمی که بالا گفتیم رد شود) می گوید باید غذایی که به فرد i-m دادیم (1<m<i-1) را به فرد i ام بدهیم که در این حالت به تناقض می رسیم چون طبق فرض ما نفرات را براساس گرسنگی به

صورت نزولی مرتب کردیم یعنی افرادی که قبل از i ام هستند مقدار گرسنگی بیشتری دارند پس این غذایی که داریم با مقدار s(j) اگر نتواند فرد i ام را سیر کند قطعا نمی تواند افراد قبل از i ام را هم سیر کند.

در نهایت الگوریتم ما بهینه است.

سوال 5:

برای تعمیم الگوریتم Dijkstra برای گرافی که به هر یال آن یکی از دو رنگ قرمز یا آبی منتسب است و با فرض این که رنگ یالی که با آن به هر راس وارد می‌شویم متفاوت از رنگ یالی است که با آن از آن راس خارج می‌شویم، می‌توانیم یک الگوریتم تعمیم یافته برای مسیریابی در این گراف ارائه دهیم. این الگوریتم برای پیدا کردن مسیر کوتاهترین فاصله با رعایت قید رنگ استفاده می‌شود.

نکته: فاصله به یک رأس بر اساس رنگ آخرین یالی که وارد آن شده، تعریف می‌شود. بنابراین دو مجموعه داریم: یک مجموعه برای راس هایی که آخرین یال قرمز بوده و یک مجموعه برای رأس‌هایی که آخرین یال آبی بوده است.

توضیح الگوریتم:

1- مجموعه ای از راس های بازدید نشده را تعریف می کنیم و فاصله به هر راس را برابر با بی نهایت قرار میدهیم.

2- فاصله به رأس مبدأ را برابر با 0 قرار می دهیم.

3- دو صف اولویت (priority queue) را تعریف میکنیم یکی برای راس هایی که آخرین یال وارد آنها قرمز بوده و دیگری برای راس هایی که آخرین یال وارد آنها آبی بوده است.

4- راس مبدا را با فاصله 0 به هر دو صف اولویت قرمز و آبی اضافه میکنیم.

5- تا زمانی که هر دو صف اولویت قرمز و آبی خالی نباشند:

- راسی که کمترین فاصله را دارد را از صف اولویت، متناسب با رنگ آخرین یال وارد شده انتخاب میکنیم. آن را راس فعلی (current_vertex) می نامیم.
- راس current_vertex را از صف اولویت، متناسب با رنگ آخرین یال حذف میکنیم.
- اگر راس current_vertex قبلاً بازدید شده است، به تکرار بعدی می رویم.
- راس current_vertex را به عنوان بازدید شده علامت گذاری میکنیم.
- برای هر یال متصل به current_vertex:
  - اگر رنگ یال وارد شده به current_vertex متفاوت با رنگ آخرین یال وارد شده از current_vertex باشد:
    - فاصله ممکن از مبدا به راس مقصد از طریق current_vertex را محاسبه میکنیم.
    - اگر فاصله ممکن کوچکتر از فاصله فعلی به راس مقصد باشد:
      - ✓ فاصله به راس مقصد را با فاصله ممکن اپدیت میکنیم.
      - ✓ رأس مقصد را به صف اولویت، متناسب با رنگ یال متصل به current_vertex اضافه میکنیم.

6- هنگامی که الگوریتم پایان یافت، فاصله به هر راس نشان دهنده کوتاهترین فاصله با رعایت قید رنگ است.

برای اثبات بهینگی الگوریتم، باید مراحل زیر را طی کنیم:

1. اثبات صحت:

- از طریق تعمیم برهان با استفاده تناقض نشان میدهیم که در هر مرحله از الگوریتم، فاصله به راسی که انتخاب می‌شود کمترین فاصله ممکن است.

- در هر مرحله، اگر راسی را انتخاب کنیم که کمترین فاصله را دارد، مطمئن می‌شویم که مسیرهای کوتاه تر را ابتدا بررسی کردیم.

- با توجه به قید رنگ، مسیرهایی که با توجه به رنگ آخرین یال وارد راس می‌شوند، جدا از مسیرهایی هستند که با توجه به رنگ آخرین یال از راس خارج می‌شوند.

- پس از پایان الگوریتم، فاصله به هر راس نشان دهنده کوتاهترین فاصله با رعایت قید رنگ است.

- بنابراین الگوریتم تعمیم یافته Dijkstra برای این گراف با رعایت قید رنگ بهینه است.

2. اثبات کامل بودن:

- اگر الگوریتم Dijkstra اصلی که برای گراف های بدون قید رنگ استفاده می‌شود را در نظر بگیریم، به این نتیجه می‌رسیم که هر مسیر کوتاه را پیدا می کند.

- با توجه به قید رنگ الگوریتم تعمیم یافته Dijkstra نیز همه مسیرهای کوتاه را با رعایت قید رنگ پیدا می کند.

- پس الگوریتم تعمیم یافته Dijkstra نیز کامل است.

با توجه به اثبات صحت و کامل بودن می‌توانیم بهینگی الگوریتم Dijkstra تعمیم یافته را اثبات کنیم.

سوال 7:

مرحله 1: ابتدا تراکنش ها رو به صورت صعودی یا نزولی مرتب می کنیم.

مرحله2: سپس به ترتیب تراکنش ها رو بررسی میکنیم و اگر ان تراکنش منجر به کمینه شدن مقدار پول جابه جا شده بین افراد شد ان را انتخاب میکنیم --> برای این کار هر تراکنش را بررسی می کنیم و اگر شخصی که باید پول بدهد (بدهکاری) و شخصی که باید پول بگیرد (بستانکاری) هم انتخاب میکنیم و پول را بین ان ها منتقل می کنیم در نهایت بدهکاری و بستانکاری هر شخص را به روز می کنیم.

مرحله 3: مرحله 2 را انقدر تکرار میکنیم که تمام تراکنش ها بررسی شوند.

مثلا روی مثالی که در سوال مطرح شده می توانیم بگوییم:

تراکنش 1: P1 به P2، 500 تومان بدهد

تراکنش 2: P2 به P3، 2500 تومان بدهد

تراکنش 3: P1 به P3، 1000 تومان بدهد

مرحله 1: به صورت نزولی مرتب میکنیم:

تراکنش 2

تراکنش 3

تراکنش 1

مرحله 2: انجام تراکنش ها

انتخاب تراکنش 1: P1 به P2، 500 تومان بدهد پس P1 بدهکار است یعنی 500- و P2 بستانکار است یعنی 500

انتخاب تراکنش 2: P2 به P3، 2500 تومان بدهد پس P2 بدهکار است یعنی 2000- و P3 بستانکار است یعنی 2000

انتخاب تراکنش 3: P1 به P3، 1000 تومان بدهد پس P1 بدهکار است یعنی 0 و P3 بستانکار است یعنی 1500

مرحله 3: همه تراکنش ها بررسی شده اند

سپس مقدار پول جابه‌جا شده بین افراد به کمترین مقدار ممکن می رسد و بدهی‌ها و بستانکاری‌ها برای هر شخص به شرح زیر است:

P1 : 0 تومان

P2 : -2000

P3 : 1500

اثبات بهینگی:

فرض می کنیم الگوریتم حریصانه ما جواب بهینه را به دست آورده است حالا فرض می کنیم الگوریتم دیگری وجود دارد که جوابی غیر از جواب حریصانه را به دست می اورد.

دو حالت داریم:

حالت اول: الگوریتم جدید کمترین تراکنش‌ها را انجام می‌دهد: اگر الگوریتم جدید جوابی کمتر از جواب حریصانه به دست آورده باشد، در این صورت با انجام یک تراکنش اضافی می‌توانیم جواب حریصانه را به دست آوریم که باعث می‌شود مقدار پول جابه جا شده بین افراد کمتر شود این نیز با فرض بر اینکه الگوریتم جدید جواب بهینه است در تناقض است.

حالت دوم: الگوریتم جدید بیشترین تراکنش‌ها را انجام می‌دهد: اگر الگوریتم جدید بیشترین تراکنش‌ها را انجام داده باشد مقدار پول جابه جا شده بین افراد به طور قطعی بیشتر از جواب حریصانه خواهد بود این نیز با فرض بر اینکه الگوریتم جدید جواب بهینه است در تناقض است.

بنابراین با توجه به حالت های بالا می‌توانیم نتیجه بگیریم که الگوریتم حریصانه بهینه است و جواب بهینه را به دست می اورد.