به نام خدا

دانشکده‌ی مهندسی برق و کامپیوتر دانشگاه تهران

طراحی و تحلیل الگوریتم‌ها، نیم‌سال اول سال تحصیلی ۹۷-۹۶

تمرین شماره ۳ **(الگوریتم‌های حریصانه)** – پاسخ تشریحی

به نکات زیر توجه فرمایید:

- الگوریتم خود را به طور کامل توضیح دهید؛ اگر در صورت سوال خواسته نشده نیازی به نوشتن شبه کد نیست.
- در هر سوال باید پیچیدگی زمانی و پیچیدگی حافظه مصرفی الگوریتم خود را نیز محاسبه کنید.
- سعی کنید الگورتیم با کمترین پیچیدگی را بدست آورید.

۱. حل کردن مسئله داده شده معادل با این است که به هر بازه یک رنگ اختصاص دهیم به گونه‌ای که هیچ دو بازه‌ای که اشتراک دارند رنگ یکسانی دریافت نکرده باشند. هر رنگ را با یک عدد بزرگتر از صفر نشان می‌دهیم. این الگوریتم را در نظر بگیرید: بازه‌ها را طبق زمان شروع (s_i) در نظر می‌گیریم و به هر بازه کوچکترین رنگی که به هیچکدام از بازه‌هایی که تا الان رنگ شده‌اند و با آن اشتراک دارند داده نشده را اختصاص می‌دهیم. واضح است که این الگوریتم یک رنگ‌آمیزی صحیح از بازه‌ها ارایه می‌کند (اگر دو بازه اشتراک داشته باشند رنگ یکسانی نمی‌گیرند). حال اثبات می‌کنیم این الگوریتم حداقل تعداد رنگ را استفاده می‌کند: از روی بازه‌های داده شده گراف G را به این شکل می‌سازیم که به ازای هر بازه در G یک راس قرار می‌دهیم و بین دو راس یک یال اضافه می‌کنیم اگر و تنها اگر بازه‌های متناظر آن دو راس با هم اشتراک داشته باشند. اجرای الگوریتم داده شده روی این گراف را در نظر بگیرید. ادعا می‌کنیم اگر این الگوریتم به یک راس رنگ k را نسبت دهد آنگاه این گراف یک خوشه با اندازه k دارد. راسی را در نظر بگیرید که به آن رنگ k داده‌ایم. با توجه به اینکه بازه‌ها را به ترتیب شروع بررسی می‌کنیم، هر بازه‌ای که رنگ شده باشد و با این بازه اشتراک داشته باشد، حتما با نقطه شروع آن اشتراک دارد، با توجه به اینکه این بازه رنگ k خورده است حداقل $k-1$ بازه با ابتدای آن اشتراک داشته‌اند که به همراه خود این بازه تشکیل یک خوشه با اندازه k می‌دهند. با توجه به اینکه برای رنگ‌آمیزی رئوس یک گراف حداقل به تعداد اندازه بزرگترین خوشه آن رنگ نیاز داریم، رنگ‌آمیزی ارایه شده توسط الگوریتم بهینه است. تنها بخشی که از مسئله می‌ماند این است که روشی ارایه دهیم که برای هر بازه کوچکترین رنگی که می‌توان به آن داد را در زمان مورد نیاز پیدا کند. برای انجام این کار ابتدا تمام اعداد 1 تا n را در یک درخت جستجوی دودویی قرار می‌دهیم سپس تمام نقاط (شروع و پایان) را به ترتیب در نظر می‌گیریم و یک لیست پیوندی شامل تمام بازه‌هایی که به ابتدای آن‌ها رسیده‌ایم ولی به انتهایشان نه را نگه می‌داریم. با دیدن هر نقطه، اگر نقطه شروع بود، کوچکترین عدد داخل درخت را حذف کرده و آن رنگ را به بازه‌ای که دیده‌ایم نسبت می‌دهیم، اگر نقطه پایان بود، آن را از لیست پیوندی حذف کرده و رنگی که به آن داده بودیم را به درخت اضافه می‌کنیم (هر نقطه شروع و پایان به بازه متناظر با خود در لیست پیوندی یک اشاره‌گر دارد.) زمان مورد نیاز برای مرتب‌سازی نقاط $O(n\ lg\ n)$ است و به ازای هر بازه یک بار به درخت اضافه، یک بار در آن جستجو و یک بار از آن حذف می‌کنیم که زمان مورد نیاز برای جمع این‌ها نیز $O(n\ lg\ n)$است.

۲. نقاط را به صورت صعودی مرتب می‌کنیم و در هر مرحله یک بازه به طول واحد با شروع از نقطه‌ای که بین نقاط پوشش داده نشده کمترین x را دارد اضافه می‌کنیم. انجام این کار در زمان خواسته شده ساده است، در زمان $O(n\ lg\ n)$ نقاط را مرتب می‌کنیم، با شروع از نقطه با کمترین x بازه‌ها را اضافه می‌کنیم و پس از اضافه کردن هر بازه، تا جایی که نقاط توسط این بازه پوشش داده شده‌اند جلو می‌رویم. برای اثبات اینکه این روش بهینه‌است به این شکل عمل می‌کنیم: هر پاسخ به این مسئله را با یک دنباله صعودی از اعداد مشخص می‌کنیم که هر عدد نقطه شروع یک بازه به طول واحد است. فرض کنید پاسخ تولید شده

توسط الگوریتم با دنباله S نشان داده شود. فرض می‌کنیم که این پاسخ بهینه نیست. شبیه‌ترین پاسخ بهینه به S را پاسخ S' می‌نامیم که اولین المانی که در آن S و S' با هم تفاوت دارند بیشترین اندیس را داشته باشد. این نقطه تفاوت را در نظر می‌گیریم که در S مقدار a و در S' مقدار a' وجود دارد. با توجه به نحوه عملکرد الگوریتم a' نمی‌تواند بیشتر از a باشد چراکه در این صورت S' حداقل یک نقطه را پوشش نداده است. پس داریم $a' < a$، اما در این صورت اگر a' را با a در S' جایگزین کنیم باز هم یک جواب درست داریم که تعداد بازه‌های استفاده شده در آن با S' مساوی است چراکه با توجه به عملکرد الگوریتم بین a' و a نمی‌تواند نقطه‌ای وجود داشته باشد و در نتیجه یک پاسخ بهینه‌ست اما به S شبیه‌تر است. این یک تناقض است، پس S یک پاسخ بهینه بوده.

۳.

a. به صورت حریصانه هنگامی که به پمپ بنزین شماره‌ی iم رسیدیم، اگر به‌اندازه‌ی رسیدن به پمپ بنزی شماره‌ی $i+1$بنزین در باک داشتیم در این پمپ نمی‌ایستیم و تا پمپ بعدی می‌رویم. اما اگر مقدار بنزین در باکمان کافی نبود به‌ناچار در پمپ iام ایستاده، باک را پر می‌کنیم. برای اثبات اینکه این روش بهینه‌است به این شکل عمل می‌کنیم: پاسخی که الگورتیم خروجی می‌دهد را S بگیرید و فرض کنید S بهینه نباشد، از بین پاسخ‌های بهینه، نزدیک‌ترین پاسخ به S را برمی‌گزینیم و آن را S'می‌نامیم (نزدیکی دو پاسخ را بزرگترین iای می‌گیریم که هر دو پاسخ تا رسیدن به پمپ بنزین iام، در ایستادن یا نایستادن، همانند هم رفتار کرده باشند). فرض کنید Sو S' تا پمپ $k-1$ام همانند هم رفتار کرده اند و در پمپ kام رفتار گوناگونی دارند در این صورت باید S در پمپ kام توقف نکرده اما S' توقف کرده باشد. در این صورت پاسخی مانند S'' را در نظر بگیرید که در پمپ kام نمی‌ایستد و در پمپ $k+1$ سوخت‌گیری می‌کند از آنجا به بعد نیز مانند پاسخ بهینه رفتار می‌کند. شمار ایستادن‌های S'' با S' برابر است اما به S نزدیک‌تر است و این خلاف فرض ما بود.

b. اینجا نیز یک روش حریصانه ارائه می‌دهیم. فرض کنید در پمپ‌بنزین iام ایستاده‌ایم، دقیقا به میزانی بنزین می‌زنیم که به‌همراه بنزین باقی‌مانده در باک به نخستین پمپ بنزینی برسیم که قیمت بنزین آن ارزان‌تر است و به آن پمپ می‌رویم . اگر با یک باک پر به چنین پمپ بنزینی نمی‌رسیم، باک را پر می‌کنیم و به پمپ بنزین بعدی می‌رویم. در آنجا همین‌کار را تکرار می‌کنیم. برای اثبات اینکه این روش میزان پول مصرفی را کمینه می‌کند به این شکل عمل می‌کنیم: پاسخی که الگورتیم می‌دهد را S بگیرید و فرض کنید S بهینه نباشد، از بین پاسخ‌های بهینه، نزدیک‌ترین به S را برمی‌گزینیم و آن را S'می‌نامیم. نخستین جایی که در نظر بگیرید که S مانند S'رفتار نمی‌کند. دو حالت ممکن است رخ دهد:

i. اگر S در حالتی باشد که باک را کامل پر کرده چون نمی‌توانسته با یک باک پر به یک پمپ بنزین با قیمت پایین‌تر برسد. در این صورت S'میزان کم‌تری بنزین زده است. چون با این میزان بنزین قطعا مجبور به بنزین زدن در جای گران‌تر است S' را می‌توان با پر کردن باک در این مرحله بهتر کرد که تناقض است.

ii. اگر S در حالتی باشد که دقیقا به میزانی بنزین زده شده که تا نخستین پمپ بنزین ارزان تر برسیم: اگر S' میزان کم‌تری بنزین زده باشد، قطعا در یک پمپ بنزین با قیمت بالاتر توقف کرده و بنزین زده که می‌توان با جایگزینی S' را بهتر کرد. اگر S'میزان بیشتری بنزین زده باشد، میزان اضافه را می‌توان در پمپ بنزین با قیمت پایین‌تر زد و این هم تناقض است.

برای پیاده‌سازی الگورتیم باید برای هر پمپ بنزین محاسبه کنیم که نخستین پمپ بنزین کم‌قیمت‌تر پس از آن کدام است و چقدر تا آنجا راه است. این محاسبه هم باید از $O(n)$باشد. برای این کار می‌توان از یک پشته بهره‌گرفت: از پمپ ۱ام شروع می‌کنیم و آن را در پشته می‌گذاریم، سپس در هر مرحله پیش از گذاشتن پمپ iام در پشته، تا

جایی که قیمت پمپ iام از قیمت پمپ سر پشته پایین‌تر است، سر پشته را برمیداریم. (pop می‌کنیم) برای هر پمپی که از سر پشته برداشته می‌شود می‌توان مقدار نخستین کم‌قیمت‌ترین پمپ و فاصله‌ی آن را حساب کرد (که برابر پمپی است که مایه‌ی برداشتن آن از پشته شده) پس از برداشتن پمپ‌های گفته شده، پمپ iام در سر پشته گذاشته می‌شود. بدین‌ترتیب در پشته همیشه پمپ‌ها به ترتیب قیمت افزایشی چیده شده‌اند که درستی مقدار محاسبه‌شده را تضمین می‌کند.

۴.

a. فرض کنید که این متن را در m خط حروف‌چینی کرده‌ایم. این m خط کلا گنجایش mk حرف را دارند. از این تعداد $\sum_{i=1}^{n} l_i$ حرف برای نوشتن کلمات استفاده شده، $n-1-m$تا برای فاصله جداساز کلمات روی یک خط استفاده شده و مابقی گنجایش جمع گنجایش باقی‌مانده خطوط یا همان $\sum_{i=1}^{m} r_i$ است. مشاهده می‌کنیم که این مقدار متاثر از نحوه حروف‌چینی نیست. بنابراین با توجه به اینکه هدف کمینه کردن $\sum_{i=1}^{m-1} r_i$ است، حروف‌چینی‌ای را می‌خواهیم که کمترین تعداد خط را استفاده کند و خط آخر آن کمترین تعداد حرف را داشته باشد. اینگونه عمل می‌کنیم: هر خط را تا جای ممکن پر می‌کنیم و سپس به خط بعدی می‌رویم. حال برای اثبات اینکه این الگوریتم زیباترین حروف‌چینی را می‌دهد: هر حروف‌چینی را با یک دنباله S نشان‌می‌دهیم که دارای n عنصر است و عنصر iام آن مشخص‌کننده خطی است که کلمه iام روی آن قرار گرفته. فرض کنید S پاسخ الگوریتم باشد و S یک زیباترین حروف‌چینی نباشد. شبیه‌ترین زیباترین حروف‌چینی به S (حروف‌چینی‌ای که اولین تفاوت آن با S بیشترین اندیس را داشته باشد) را در نظر می‌گیریم و S' می‌نامیم. این اولین تفاوت را در نظر می‌گیریم، فرض کنید در اندیس i رخ داده باشد. با توجه به نحوه کار الگوریتم می‌دانیم که $S'_i > S_i$، حروف‌چینی S'' را دقیقا مانند S' در نظر می‌گیریم با این تفاوت که $S''_i = S_i$. S'' یک حروف‌چینی معتبر است چرا که با توجه به عملکرد الگوریتم خط S_i حتما گنجایش کلمه iام را داشته که روی آن قرار گرفته، علاوه بر این S'' نه تعداد خطوط آن از S' بیشتر است و نه در خط آخر حروف بیشتری از S' دارد (بجز در حالتی که تعداد خطوط آن از S' کمتر باشد که در این صورت S'' از S' زیباتر است). یعنی یک زیباترین حروف‌چینی یافته‌ایم که از S' به S شبیه‌تر است. اما این یک تناقض است پس S یک زیباترین حروف‌چینی بوده.

b. این مسئله را با استفاده از برنامه‌ریزی پویا حل می‌کنیم. $prettiest(i)$ را تعریف می‌کنیم برابر با دوتایی حروف‌چینی‌ای از i کلمه اول که کمترین مقدار $\sum_{i=1}^{m} r_i$ را دارد و خود مقدار $prettiest - \sum_{i=1}^{m} r_i$. $partial(i)$ را تعریف می‌کنیم برابر با دوتایی حروف‌چینی از i کلمه اول که کمترین مقدار $\sum_{i=1}^{m-1} r_i$ را دارد و خود مقدار $\sum_{i=1}^{m-1} r_i$. پاسخ مسئله برابر است با $prettiest - partial(n)$. برای محاسبه $prettiest - partial(n)$ اینگونه عمل می‌کنیم: فرض کنید یک خط می‌تواند کلمات nام تا $n-p$ام را را در خود جای دهد، در این صورت:

$$prettiest - partial = \min_{i \in \{1,\ldots,p\}} (prettiest(n-i-1))$$

به طور مشابه برای محاسبه $prettiest(n)$ داریم:

$$prettiest(n) \; = \; \min_{i \in \{1,\ldots,p\}} (k - \sum_{j=1}^{i} (l_{n-j} + 1) + prettiest(n-i-1))$$

۵. کارها را به ترتیب صعودی مهلت انجام مرتب می‌کنیم و به همین ترتیب انجام می‌دهیم. برای اثبات بهینه بودن پاسخ این الگوریتم به این شکل استدلال می‌کنیم: ابتدا تعریف می‌کنیم که یک برنامه‌ریزی از کارها دارای یک نابجایی است اگر i و j وجود داشته باشند به گونه‌ای که $d_i < d_j$ اما کار j زودتر از کار i انجام شود. روشی که ما ارایه داده‌ایم یک برنامه‌ریزی بدون نابجایی تولید می‌کند. دقت کنید که مقدار L برای تمام برنامه‌ریزی‌های بدون نابجایی یکسان است (در صورتی که دو یا چند

کار ضرب‌الاجل یکسانی داشته باشند بیشتر از یک برنامه‌ریزی بدون نابجایی وجود دارد) چرا که در یک برنامه‌ریزی بدون نابجایی تمام کارهایی که مهلتشان زمان d است پشت سر هم قرار دارند و حداکثر دیرکرد آن‌ها برابر با دیرکرد آخرین آن‌هاست و این مقدار به ترتیب این کارها بستگی ندارد. حال نشان می‌دهیم اگر یک برنامه‌ریزی دارای نابجایی باشد، حتما دو کار متوالی در این برنامه وجود دارند که کاری که زودتر انجام می‌شود مهلت دیرتری دارد، یعنی یک نابجایی متشکل از دو کار که پشت سر هم آمده‌اند. دو کار i و j را در نظر بگیرید که $d_i < d_j$ اما کار j قبل از کار i انجام شده. اگر این دو کار پشت سر هم باشند یک نابجایی شامل دو کار پشت سر هم پیدا شده، در غیر این صورت از کار j شروع کرده و به جلو می‌رویم، از آن‌جا که نهایتا به کار i می‌رسیم که $d_i < d_j$ در این بین حتما یک کار وجود دارد که از مهلتش از کار قبلی عقب‌تر است و این یک نابجایی متشکل از دو کار پشت سر هم است. حال آماده اثبات ادعایمان هستیم: یک برنامه‌ریزی بدون نابجایی، یک برنامه‌ریزی بهینه است. فرض می‌کنیم چنین نیست. یک برنامه‌ریزی که بهینه که دارای حداقل تعداد نابجایی است را در نظر می‌گیریم. از آن‌جا که این برنامه‌ریزی بدون نابجایی نیست، حتما دارای یک نابجایی متشکل از دو کار متوالی است. این دو کار را کارهای i و j می‌نامیم که $d_i < d_j$ اما کار jام دقیقا قبل از کار iام انجام شده. زمان پایان کار i در این برنامه را با f_i و دیرکرد آن را با l_i نشان می‌دهیم. حال ترتیب انجام شدن کارهای i و j را تعویض می‌کنیم. در برنامه جدید که از تعویض این دو به دست آمده زمان پایان کار iرا با f'_i و دیرکرد آن را با l'_i نشان می‌دهیم. با تعویض ترتیب انجام شدن کارهای i و j تغییری در دیرکرد کارهای دیگر ایجاد نمی‌شود، علاوه بر این دیرکرد کار i هم افزایش پیدا نمی‌کند، تنها چیزی که می‌ماند دیرکرد کار j است. دیرکرد کار j ممکن است زیاد شود اما این حداکثر دیرکرد را زیاد نمی‌کند پس از جابجایی، کار j در زمان f_i (زمان پایان کار i در برنامه اولیه) تمام می‌شود. اگر کار j در این برنامه‌ریزی دیرکرد داشته باشد، دیرکرد آن برابر است با $l'_j = f'_j - d_j = f_i - d_j$ اما نکته مهم این است که کار j نمی‌تواند در کار جدید دیرکرد بیشتری نسبت به دیرکرد کار i در برنامه اصلی داشته باشد. به طور خاص فرض ما مبنی بر اینکه $d_i < d_j$ نتیجه می‌دهد $l'_j = f_i - d_j < f_i - d_i = l_i$. از آنجا که دیرکرد برنامه اصلی برابر با $L \geq l_i > l'_j$ بود، این تغییر باعث افزایش حداکثر دیرکرد نمی‌شود و تعداد نابجایی‌ها را یکی کم می‌کند که با فرض قبلی در تناقض است، یعنی برنامه بهینه‌ای که نابجایی نداشته باشد وجود دارد.

۶. واضح است که هر بازه $[z_i, z_i + 1]$ باید پوشانده شود، اگر تعداد این بازه‌ها کمتر مساوی k باشد، با تعداد بازه لازم فقط این بازه‌ها را می‌پوشانیم. اما اگر تعداد این بازه‌ها، n، بیشتر از k باشد آنگاه باید با بزرگتر کردن بازه‌ها این بازه‌ها را به هم متصل کنیم. به طور خاص با متصل کردن هر دو بازه به هم یکی از تعداد بازه‌های استفاده شده کم می‌شود یعنی برای حل مسئله باید $n - k$ بار دو بازه به هم متصل شوند. به این شکل عمل می‌کنیم: فاصله‌های بازه iام و بازه $i + 1$ام را با l_i نشان می‌دهیم. حال l_iها را به صورت صعودی مرتب می‌کنیم و با متصل کردن $n - k$تا بازه‌ای که کمترین فاصله‌ها را دارند مسئله را حل می‌کنیم. دقت کنید که متصل کردن دو بازه، فقط فاصله بین آن دو بازه را از بین می‌برد و تاثیری روی فاصله بقیه بازه‌ها با هم ندارد. برای اثبات بهینه بودن این روش به این شکل استدلال می‌کنیم: هر پاسخ به مسئله شامل متصل کردن تعدادی از بازه‌هاست. هر پاسخ را با یک دنباله S که نشانگر این است که فاصله بین کدام بازه‌ها پر شده است نشان می‌دهیم. S به ترتیب صعودی طول فاصله‌ها مرتب شده است. پاسخ تولید شده توسط الگوریتم را با S نشان می‌دهیم و فرض می‌کنیم این پاسخ بهینه نیست. شبیه‌ترین پاسخ به S (پاسخی که اولین تفاوت آن با S بیشترین اندیس را دارد) را در نظر بگیرید و S' بنامید. نقطه اولین تفاوت S و S' را در نظر بگیرید. در این نقطه در S فاصله بین بازه iام و $i + 1$ام برای پر شدن انتخاب شده ولی در S' فاصله بین i' و $i' + 1$. طبق نحوه کار الگوریتم می‌دانیم که $l_i \leq l_{i'}$ بنابراین از روی S' پاسخ S'' را به این شکل می‌سازیم که بجای پر کردن فاصله بین i' و $i' + 1$ فاصله بین i و $i + 1$ را پر می‌کنیم، با این کار تعداد بازه‌های استفاده شده زیاد نشده و جمع طول بازه‌های استفاده شده نیز افزایش نمیابد یعنی S'' نیز یک پاسخ بهینه است اما این فرض ما که S' شبیه‌ترین پاسخ بهینه به S است در تناقض است، بنابراین S یک پاسخ بهینه بوده.

پیروز و سربلند باشید