به نام خداوند بخشایشگر مهربان

تمرین سوم الگوریتم پیشرفته محمد تنهایی ۸۷۲۰۲۴۴۶ نرم افزار

برنامه ریزی خطی:

۲۹.۱-۷

ابتدا مسئله را به شکل slack در می آوریم ، سپس با استفاده از الگوریتم simplex آنرا حل می کنیم ، نشان می دهیم که X_1-X_2 دارای جواب بی نهایت است.

$$z = x_1 - x_2$$
$$x_3 = -1 + 2x_1 - x_2$$
$$x_4 = -2 + x_1 + 2x_2$$

به طور شهودی :از شکل عبارت کاملاً مشخص است که جواب unbound است ، زیرا ضریب x_1 در هر سه جمله مثبت است ، می توانیم x_1 را به هر اندازه ای که خواستیم زیاد کنیم و همچنان محدودیت های ما ارضا شود ،

و بر اساس الگوریتم : در الگوریتم simplex ابتدا یک جواب اولیه محاسبه می شود ، به این صورت که یک $-x_0$ به عبارات اضافه می شود ، و عبارت اول به صورت $z=-x_0$ تبدیل می شود ،

$$z = -x_0$$
$$x_3 = -1 + 2x_1 - x_2 + x_0$$
$$x_4 = -2 + x_1 + 2x_2 + x_0$$

حال نقش x_0 و x_4 عوض می شود ،

$$z = -2 + x_1 + 2x_2 - x_4$$
$$x_0 = 2 - x_1 - 2x_2 + x_4$$
$$x_3 = 1 + x_1 - 3x_2 + x_4$$

الگوریتم اکنون x_1 را انتخاب کرده و به اندازه ۲ واحد آنرا افزایش می دهد ، و جای آن با x_0 عوض می شود:

$$z = -x_0$$
$$x_1 = -x_0 - 2x_2 + x_4$$
$$x_3 = 3 - x_0 - 5x_2 + 2x_4$$

با حذف x_0 به عبارت زیر می رسیم :

$$z = -3x_2 + x_4$$
$$x_1 = -2x_2 + x_4$$
$$x_3 = 3 - 5x_2 + 2x_4$$

الگوریتم simplex در خط ۵ مقدار بی نهایت برای دلتای ۱ و ۳ محاسبه کرده و در خط ۱۰ الگوریتم با توجه به اینکه کمترین مقدار دلتا برابر بی نهایت است جواب unbound بر می گرداند.

۲۹.۲-۵

برای حل این مسئله باید توجه داشته باشیم که به جای اینکه برای هر u,v شرط بگذاریم ، برای هر u,v که دو سر یک یال در گراف است چنین محدودیتی بگذاریم بنابراین الگوریتم به شکل زیر اصلاح می شود:

$f(u,v) \le c(u,v) \qquad for\ each\ u,v\ so\ (u,v) \in E\ ,$

$f(u,v) = -f(v,u) \qquad for\ each\ u,v\ so\ (u,v) \in E\ ,$

$\sum_{v\ so\ that\ (u,v) or\ (v,u) \in E} f(u,v) = 0 \qquad for\ each\ u\ \in V - \{s,t\}\ .$

در LP جدید ، تعداد محدودیت های ایجاد شده در خط اول برابر E ، در خط دوم نیز برابر E و خط آخر نیز برابر با V است . اگر شرط سیگما را تغییر نمی دادیم هنوز تعداد محدودیت های خط آخر برابر V بود ، اما با شرط جدید تعداد متغیرهایی که در شرط باید چک شوند بسیار کاهش می یابد و سرعت الگوریتم بالاتر می رود.

بنابر این تعداد محدودیت ها برابر است با O(E+V)

۲۹.۳-۲

نشان می دهیم که در خط ۱۳ ام pivot مقدار v هیچگاه کم نمی شود . در این خط داریم $\hat{v} = v + C_e\widehat{b_e}$ کافی است که نشان دهیم مقدار $C_e\widehat{b_e}$ هیچ گاه منفی نمی شود ، و خود به خود اثبات ما کامل می شود.

فرض کنیم که مقدار v کم می شود ، در نتیجه باید $C_e\widehat{b_e}$ منفی شود ، می دانیم که C_e منفی نیست ، زیرا اگر منفی باشد ایندکس آن هیچ وقت برای pivot انتخاب نمی شود (خط ۳ الگوریتم simplex این چک را انجام می دهد) ، بنابراین باید $\widehat{b_e}$ منفی باشد ، $\widehat{b_e}$ برابر است با تقسیم b_l/a_{le} همچنین از خط ۵ الگوریتم simplex داریم که $a_{le}>0$ بنابراین تنها راه منفی شدن عبارت منفی شدن b_l است و این تناقض آشکار است ، زیرا اگر b_l منفی باشد متغیر basic متناظر با آن دارای مقدار منفی می شود ، در این حالت LP از ابتدا یک حل feasible نداشته است! (تمرین ۳-۲۹.۳)

اثبات مثبت بودن b_l در هر دور در صفحه ۷۹۸ کتاب در بخش Maintenace آمده است ، در صورتی که b_l بتواند منفی باشد ختم الگوریتم و درستی آن زیر سوال می رود!!

با توجه به برهان خلف گفته شده $C_e\widehat{b_e}$ هیچگاه منفی نمی شود ، بنابراین مقدار v هرگز کم نمی شود .

۲۹-۱

a. این حالت به سادگی قابل حل است ، اگر ما یک الگوریتم برای linear programming داشته باشیم به راحتی می توانیم linear-inequality feasibility را حل کنیم ،به این صورت که تابعی که می خواهیم maximize شود را برابر صفر قرار داده و از الگوریتم linear programming استفاده می کنیم ، اگر این الگوریتم

infeasible برنگرداند مسئله inequality نیز feasible است ، و در غیر این صورت مسئله inequality ، infeasible خواهد بود.

فرض کنیم که $Ax \leq b$ مسئله inequality باشد ، به سادگی می توانیم با m محدودیت و n متغیر (در حالتی که بخواهیم حتماً متغیر ها از صفر بزرگتر باشند باید از 2n متغیر استفاده کنیم) مسئله را حل کنیم.

b. در این حالت عکس حالت قبل را داریم ، اگر جواب مسئله LP بینهایت نباشد می توانیم از یک روش تکرار استفاده کنیم ، به این صورت که یک عدد که می دانیم بالاترین جواب ممکن است را در نظر گرفته ، و یک inequality از شرط موجود می سازیم ، به عنوان نمونه اگر F بزرگترین عدد ممکن باشد ، و عبارتی که می خواهیم maximum کنیم برابر $a_1x_1+a_2x_2+\ldots\ldots+a_nx_n$ باشد عبارت زیر را می سازیم :

$$a_1x_1+a_2x_2+\ldots\ldots+a_nx_n \geq F$$
$$Ax \geq b$$
$$x_1,x_2,x_3,x_4,\ldots x_n \geq 0$$

اگر چنین عبارتی جواب infeasible داد F را نصف می کنیم و .... از یک جستجوی باینری استفاده می کنیم تا به جواب برسیم.

در ابتدای الگوریتم برای چک کردن feasible بودن LP می توانیم بدون افزودن هیچ شرط اضافی ، تنها شرط های عبارت LP را به وسیله الگوریتم linear-inequality feasibility حل کنیم ، در صورتی که جواب feasible داد مسئله LP ما نیز جواب feasible خواهد داشت.

در این حالت تعداد متغیر ها برابر با n ، و تعداد محدودیت ها برابر 2m+1 خواهد بود.

هندسه محاسباتی:

۳۳.۲-۵

برای حل این مسئله از sweep line استفاده می کنیم ، البته ابتدا باید چک کنیم کـه یکـی از چنـد ضـلعی هـا داخـل دیگری نباشد ، زیرا در این صورت جارو کردن خطوط جوابگو نیست ، برای این منظور می تـوان از ایـده جـالبی اسـتفاده کرد ، خطوط ۲ چند ضلعی را روی هم ریخته و convex hull آنرا محاسبه می کنیم ، اگر convex hull برابر یکـی از چند ضلعی های اولیه بود که نشان دهنده قرار گرفتن یکی در دیگری است و الگوریتم اعلام می کند که ۲ چنـد ضـلعی دارای اشتراک هستند O(nlogn) ، اما اگر برابر یکی از شکل های اولیه نبود از sweep line استفاده می کنیم :

دو چند ضلعی ساده در صورتی اشتراک دارند که یکی از اضلاع چند ضلعی اول ، یکی از اضلاع چند ضـلعی دوم را قطـع کند ، و می دانیم که تشخیص اشتراک خطوط به سادگی توسط الگوریتم sweep line قابل انجـام اسـت ، بـا توجـه بـه فرض مسئله (که تعداد راس ها n تا است) تعداد n خط داریم که در زمان O(nlogn) می تـوان نقـاط تقـاطع آن هـا را محاسبه کرد ، پس از به دست آوردن نقاط تقاطع آنها را با رئوس چند ضلعی مقایسه می کنیم اگر نقطه ای وجود داشـت که جز رئوس چند ضلعی ها نبود الگوریتم جواب بله بر می گرداند (البته مـا همچنـان شـرط sweep line کـه عبـارت است از اینکه حداکثر ۲ خط همدیگر را قطع می کنند را در اینجا داریم)

پیچیدگی الگوریتم برابر است با O(nlogn+nlogn) که همان O(nlogn) می شود.

۳۳.۳-۵

با اندکی تغییرات می توان مسئله را در زمان n^2 حل کرد ، مشکل الگوریتم تکرار این است که از اطلاعات قبلـی اسـتفاده نمی کند ، می توانیم مکان نقطه جدید را بـین نقـاط قبلـی جسـتجو کـرده (در زمـان logn) و در زمـان n نیـز CH را بسازیم:

فرض کنیم که نقطه جدید p به مجموعه نقاط قبلی اضافه شده است ، مکان این نقطه را بین مجموعـه نقـاط convex polygon $<p_0,p_1,p_2,\ldots.p_k>$ پیدا می کنیم ، اگر نقطه داخل CP باشد که لازم نیست کار اضافی انجـام دهـیم ، در غیر این صورت مجموعه نقاط $<p_0,p_1,p_2,\ldots.p_k>$ اجتماع p را به دست می آوریم کـه یـک CP را تشـکیل دهـد.ایـن نقاط بر اساس minimum y –coordinate و یا leftmost در یک حالت متصل (tie) مرتب شده اند.

حال خطوط ۳ تا ۱۰ الگوریتم Graham-Scan(NS) را اجرا می کنیم ، همان طـور کـه در CLRS اثبـات شـد ، اگـر نقاط ما به درستی مرتب شده باشند یافتن Convex Hull در این حالت پیچیدگی برابر O(n) دارد و نشان مـی دهـیم که جستجو و اضافه کردن نقطه p نیز در زمان logn قابل انجام است ، بنـابراین کـل عملیـات در زمـان n+logn قابـل انجام است ، و چون که n بار این کار را انجام می دهیم مجموع کار ها برابر با $O(n^2)$ است.

اما هنوز نشان نداده ایم که چه طور می توان در زمان logn نقطه p را به مجموعه نقـاط قبلـی اضـافه کـرد ، بـرای ایـن منظور جستجویی شبیه باینری انجام می دهـیم ،Search(Q,p) فـرض کنـیم کـه $Q=<p_0,p_1,p_2,\ldots p_n>$ یـک CP است ، که p0 بر اساس minimum y –coordinate و یا leftmost در یک حالت متصل (tie) مرتـب شـده اسـت ،

فرض کنیم که نقطه p را می خواهیم اضافه کنیم ، ابتدا چک می کنیم که $\overrightarrow{p_0p}$ نسبت به $\overrightarrow{p_0p_{n/2}}$ ساعتگرد است یا پاد ساعت گرد ؟ (با نقطه گردش p0) اگر ساعتگرد بود جواب Search(<p_0,p_1,p_2,…$p_{k/2}$>,p) را بر می گردانیم و در غیر این صورت Search(<$p_{k/2+1}$,….,p_n>,p) را بر می گردانیم . این جستجو در بهترین حالت به ۳ نقطه ختم می شود ، در این حالت به راحتی می توانیم چک کنیم که آیا p داخل Q قرار دارد یا خارج آن ، بنابراین ما می توانیم جستجو را در زمان $T(n) = T(n/2) + O(1)$ انجام دهیم و $T(3) = 1$ است ، پس پیچیدگی کار برابر است با O(logn) ، کارهای بعد از این مرحله را در پاراگراف ابتدایی شرح دادیم.

۳۳.۴-۳

الگوریتم closest pair نیاز به کمی دستکاری دارد ، این تغییرات به قرار زیر هستند :

- در فاز اولیه الگوریتم : $|P|<3$ باید ما نزدیکترین نقطه را بر اساس فاصله منهتن برگردانیم.
- در فاز ادغام نیز به جای مقایسه با ۷ نقطه باید با ۹ نقطه مقایسه شود (۹ نقطه که به فاصله δ منهتن از نقطه P قرار گرفته اند)

درستی الگوریتم دقیقاً مانند قبل اثبات می شود ، چیزی که ما باید نشان دهیم این است که حداکثر ۱۰ نقطه به فاصله δ*2δ از نقطه P قرار دارند. (از این به بعد هر جا اسمی از فاصله بردیم منظور فاصله منتهن است)

قسمت سمت چپ نقطه P را PL می نامیم ، نشان می دهیم که حداکثر ۵ نقطه می توان در این مربع δ*δ قرار داد . برای این منظور مربع را به چهار مربع δ/2*δ/2 تقسیم می کنیم ، اگر ۵ یا بیشتر از ۵ نقطه داشته باشیم در یکی از مربع ها باید حداقل ۲ نقطه داشته باشیم ، برای اینکه این ۲ نقطه فاصله δ از هم داشته باشند باید هر دو تا در گوشه مربع قرار گرفته باشند ، اگر به همین منوال ادامه دهیم متوجه می شویم که در یک مربع δ*δ می توانیم حداکثر ۵ نقطه که یک نقطه در وسط و ۴ نقطه دیگر در گوشه های مربع هستند داشته باشیم ، برای مربع سمت راست نیز همین داستان تکرار می شود ، بنابراین حداکثر ۱۰ نقطه در اطراف نقطه P قرار دارند.

الگوریتم های تقریبی:

۱.۳

ابتدا نشان می دهیم که الگوریتم جواب VC را می دهد ، اگر از الگوریتم DFS استفاده کنیم و یک درخت بسازیم ، مجموعه رئوسی این درخت را S بنامیم (به جز رئوسی که برگ هستند) چنین مجموعه ای خاصیت VC را خواهد داشت ، زیرا هر یال حداقل یک راس آن در S خواهد بود ، هر یالی که در مجموعه یالهایی که DFS جستجو کرده وجود ندارد ، به این خاطر به درخت اضافه نشده است که هر یکی از رئوس آن قبلاً زیارت! شده است. نمی تواند ۲ راس یک یال هر دو برگ باشد ، زیرا در چنین حالتی DFS حتماً یکی از راس ها را ادامه می داد و چنین اتفاقی هیچ گاه امکان رخ دادن ندارد.

بنابراین چون هر یال G یک راس در S دارد ، S خاصیت VC را دارد.

نشان می دهیم که $|S|<2*OPT$

باید نشان دهیم که matching ای وجود دارد که دارای $|S|/2$ راس است ، از طرفی می دانیم که $|m|<OPT$ و در نتیجه $|S|/2<|m|<OPT$ که می توان نتیجه گرفت : $|S|<2*OPT$

اثبات $|S|/2<|m|$: دو نوع matching تعریف می کنیم ، M_{odd} و M_{even} ، M_{odd} تطبیقی است که از رئوسی که در عمق فرد بازدید شده اند با یک نود همسایه آن، ساخته می شود و M_{even} تطبیقی است که از رئوسی که در عمق زوج بازدید شده اند با یک نود همسایه آن ، ساخته می شود. می دانیم که این matching درست است ، زیرا یک گره زوج هیچگاه با گره زوج دیگر تطبیق پیدا نمی کند (در مورد فردها هم همین طور است)

چون که هر راس در درخت S دارای حداقل یک فرزند است ، بنابراین رئوس S ، حتماً یا در M_{odd} و یا M_{even} و یا در هر دو وجود خواهند داشت ، بنابراین یکی از این تطبیق ها شامل حداقل $|S|/2$ راس خواهد بود و اثبات کامل می شود.

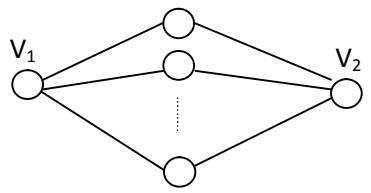
۲.۱

۱.حداکثر مقدار OPT برابر تعداد یالهای گراف است ، ما از این واقعیت استفاده کرده و ضریب الگوریتم تقریبی رو محاسبه می کنیم ، $OPT\leq E$ در نتیجه $OPT/2 \leq E/2$

در الگوریتم حریصانه ، برای هر راس ما $d(v,A)$ و $d(v,B)$ را محاسبه می کنیم ، و هر کدام از مجموعه ها که بیشترین کاردینالیتی را به ما بدهد انتخاب می کنیم ، اگر مجموع کاردینالیتی را برابر S قرار دهیم پس از این کار S برابر با $S+\max(d(v,A),d(v,B))$ می شود و اگر کاردینالیتی که از دست داده ایم را با F نشان دهیم ، F برابر می شود با $F+\min(d(v,A),d(v,B))$ بنابراین همیشه $F\leq S$ و چون که F+S برابر است با E پس $S\geq E/2$ است.

از طرفی داریم که $OPT/2 \leq E/2$ پس در نتیجه $S\geq OPT/2$ و این یعنی ضریب الگوریتم تقریبی برابر است با 1/2

۲.مثال tight :



در این حالت ، حالت OPT برابر است با اجتماع v_1 و v_2 با هم و بقیه رئوس هم با هم ، در حالی که الگوریتم دو مجموعه یکی شامل v_1 و یکی شامل v_2 و بقیه رئوس می سازد که دقیقاً نصف جواب بهینه را جواب می دهد.

۳.بیشترین مقدار OPT در گراف های ۲ بخشی اتفاق می افتد که برابر با تعداد یالهای گراف می شود. همان طور که گفته شد ما از upper bound ای برابر E استفاده کرده ایم.

۴.برای ساخت چنین گرافی باید توجه کنیم که تعداد یالها زوج باشد ، گراف کامل K_n چنین ویژگی دارد.n از یک عددی بیشتر بشود OPT به نصف تعداد یال ها نزدیک می شود ، به عنوان نمونه برای K_{200} ، OPT کمتر از 0.501E است.

۵.برای جنرال کردن مسئله برای گراف وزن دار می توانیم چنین مسئله را تعریف کنیم : یک گراف وزن دار بی جهت داریم ، می خواهیم که بیشترین وزن بین دو مجموعه S و V-S را به دست آوریم ، الگوریتم حریصانه همان است که توضیح داده شد با این تفاوت که d(v,A) مجموع وزن بین راس v و مجموعه A را بر می گرداند.

۲.۶

۱.این الگوریتم را به شیوه زیر می سازیم ، هر راس شامل دلتا +۱ مجموعه رنگ و درجه راس است.

- راس با بزرگترین درجه را انتخاب کن
- از مجموعه رنگ های این راس یک رنگ را انتخاب کن و درجه راس را برابر صفر قرار بده.
- رنگ انتخاب شده را از مجموعه رنگ های همسایه های این راس کم کن ، همچنین یکی از درجه این رئوس کم کن.
- الگوریتم را تکرار کن تا زمانی که راس رنگ نکرده ای باقی نمانده باشد.

۲.بخش مشکل زا این است که ممکن است درجه یک راس بالاتر از $\sqrt{n}$ باشد ، و در نتیجه الگوریتم قبلی جواب مناسبی ندهد ، برای این منظور می توان از یک تکنیک استفاده کرد ، اگر یک گراف را بتوان با ۳ رنگ ، رنگ کرد ، همسایه های یک راس تشکیل یک گراف دو بخشی می دهند ، زیرا یک رنگ به راس جاری اختصاص داده می شود و در نتیجه همسایه ها باید با ۲ رنگ ، رنگ شوند و می دانیم که گرافی که با ۲ رنگ قابل رنگ شدن باشد دو بخشی است . بنابراین می توانیم در الگوریتم کارهای زیر را انجام داده سپس گراف را به وسیله الگوریتم قبلی رنگ می کنیم:

۱. بزرگترین راسی که درجه راس آن از $\sqrt{n}$ بیشتر است را انتخاب کن.

۲. از مجموعه رنگ های این راس یک رنگ را انتخاب کن و درجه راس را برابر صفر قرار بده.
۳. رنگ انتخاب شده را از مجموعه رنگ های همسایه های این راس کم کن ، همچنین یکی از درجه این رئوس کم کن.

- برای همه همسایه های این راس ۲ و ۳ را اجرا کن
- تا زمانی که راسی با درجه بیشتر از $\sqrt{n}$ وجود دارد این الگوریتم را تکرار کن در غیر این صورت الگوریتم بخش اول سوال را با $\sqrt{n}$ رنگ تکرار کن.

۳.۲

۱. Sender-Receiver I Problem is in P: با توجه به hint ای که داده شده است ، یک راس f به گراف G اضافه می کنیم و آنرا G' می نامیم ، این راس را به همه sender ها با یالی به وزن صفر متصل می کنیم ، حال MST-Based Algorithm را با شرط required برای راس جدید و رئوس receiver و شرط Steiner روی بقیه رئوس باقی مانده ، اجرا می کنیم .MST-Based Algorithm را قبلاً در کتاب vazirani بحث کرده ایم. حلی که برای G' به دست می آوریم با حذف f و یال متصل به آن یک حل برای گراف G است.

MST-Based Algorithm : برای هر دو راس (u,v) ، کوتاهترین فاصله را بین دو راس به دست آورده و یالی با همین وزن بین (u,v) به گراف G' اضافه می کنیم ،MST را روی G' به دست می آوریم ، سپس از روی این MST همه رئوس G را که به این رئوس مربوط هستند را انتخاب می کنیم.

feasible solution : تنها راه رسیدن به f رد شدن از یک sender است ، و چون که همه receiver ها به f متصل هستند (زیرا یک MST داریم) بنابراین هر receiver حداقل به یک sender متصل است.

optimal solution : چون که ما MST را روی گراف G' اجرا کردیم مطمئن هستیم که یالهای دیگری وجود ندارند که بتوانند با هزینه کمتری این مجموعه رئوس را به هم متصل کنند ، از طرفی چون که هر راس یک sender یا receiver است راس دیگری وجود ندارد که بتوان به این مجموعه رئوس اضافه کرد و همچنین راسی وجود ندارد که بتوان آنرا حذف کرد.

polynomial time : واضح است ، افزودن یک راس در زمان چند جمله ای انجام می شود ، همچنین می دانیم که MST در زمان چند جمله ای به راحتی قابل محاسبه است.

بنابراین Sender-Receiver I problem در P است.

۲.Sender-Receiver II Problem is in NP-hard :

برای اثبات NP بودن این مسئله باید یک reduction از مسئله Steiner به این مسئله انجام دهیم که در زمان چند جمله ای قابل انجام باشد.

یکی از رئوس required از مسئله Steiner tree را به روی یکی از رئوس sender نگاشت مـی کنـیم ، و بقیـه رئـوس required را بر روی رئوس receiver نگاشت می کنیم ، بقیه رئوس non-required نیز بدون تغییر باقی می مانند.
Steiner tree می کوشد که رئوس اجباری را با کمترین هزینه به همدیگر متصل کند ، مسئله را به گونه ای تغییر دادیم که یکـی از sender ها را به همـه receiver ها متصل کند ، همه این کارها در زمان چند جمله ای انجام شد ، بنابراین این مسـئله NP-hard است . زیرا می دانیم که Steiner tree یک مسئله NP-hard است.

factor 2 approximation algorithm.۳

این الگوریتم همان است که در قسـمت اول ذکـر شـد ، امـا در اینجـا فـاکتور آنـرا محاسـبه مـی کنـیم . بـا اسـتفاده از Theorem 3.3 از کتاب وزیرانی می دانیم که MST ایی که انجام داده ایم از 2*OPT ایی است که بـرای Steiner tree به دست آورده ایم . در بخش قبل ثابت کردیم که چگونه می توان Steiner tree را تبدیل به مسئله کنونی کـرد ، بنابراین OPT دو مسئله تفاوتی با هم ندارد بنابراین این الگوریتم برای مسئله کنونی نیز از درجه 2*OPT است.

۴.۴

گراف کامل به این شیوه ساخته می شود ، برای هر دو راس u,v یک یال با وزن f(u,v) متصل می کنیم ، این وزن برابـر کمترین وزن برش بین u,v است ، که از روی گراف G محاسبه می شود ، حال روی چنین گرافی که آنرا G' می نـامیم یک maximum weight spanning tree می زنیم ، باید نشان دهیم کـه ایـن درخـت خاصـیت Gomory Hu را دارد ، این کار به سادگی امکان پذیر است ، با توجه به hint داده شده ، فرض کنیم که مسیر v بـه u شـامل $w_1,w_2,...$ باشـد یعنـی $<v,w_1,w_2,....w_k,u>$ چـون کـه مـا درخـت مـاکزیمم را سـاخته ایـم داریـم :
$f(u,v) \leq \min\{f(u,w_1),f(w_1,w_2),....f(w_k,u)\}$ از طرفی می دانیم که f(u,v) کمتـرین بـرش بـین u,v اسـت ، اثبات می کنـیم کـه $f(u,v) = \min\{f(u,w_1),f(w_1,w_2),....f(w_k,u)\}$ زیـرا اگـر f(u,v) کمتـر از min باشـد بـه تناقض می رسیم : فرض کنیم که مینیمم عبارت f(w1,w2) باشد ، در ایـن صـورت حتمـاً f(w1,w2)=f(u,v) زیـرا مسیری از v به u وجود دارد که حتماً از این ۲ راس می گذرد ، بنابراین برش کمینه حتماً باید این دو راس را از هم جدا کند ، یعنی $f(u,v) \geq f(w1,w2)$ زیرا در غیر این صورت f(w1,w2) می نیمم عبارت موجـود نخواهـد بـود ، و یـک عبارت کوچک تر وجود خواهد داشت ، بنابراین داریم : $f(u,v) \geq f(w1,w2)$ و $f(u,v) \leq f(w1,w2)$ و می تـوان از این ۲ نامساوی نتیجه گرفت که $f(u,v) = f(w1,w2)$

با توجه به اثبات انجام شده ، مشخص می شود که درخت ساخته شده خاصیت Gomory Hu دارد ، زیـرا بـرای هـر ۲ راس اندازه می نیمم یال مسیر بین دو راس برابر می نیممم کات گراف متناظر بین ۲ راس است.

۵.۳

۱.الگوریتم تقریبی با ضریب ۲ :

الگوریتم : یک راس دلخواه بردار ، به عنوان مرکز کلاستر اول نام گذاری کن ، بقیه k-1 مرکز را به این صورت انتخاب کن که هر بار دورترین راس ممکن به مجموعه رئوسی که تا به حال انتخاب شده را انتخاب کن ، پس از انتخاب k مرکز ، بر اساس نزدیکی هر راس به یکی از این مراکز آنها را جز دسته این مراکز قرار بده (هر راس را داخل مجموعه نزدیکترین مرکز قرار بده)

Feasibility : ما رئوس را به k کلاستر تقسیم کردیم! (یعنی feasible است الگوریتم)

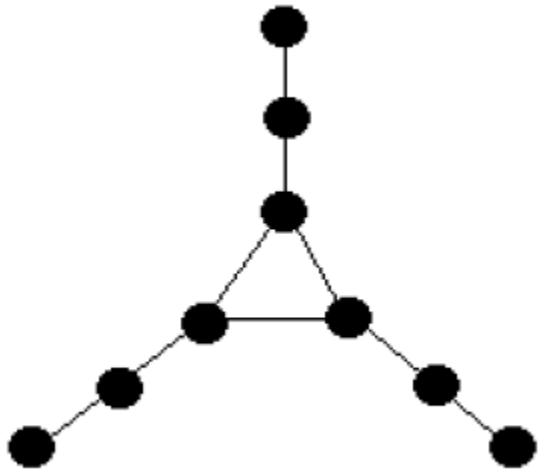
آنالیز ضریب تقریب : فرض کنیم که z دورترین نقطه از k مرکز باشد ، فاصله z از نزدیکترین مرکز (یعنی مرکز کلاستر خودش) را Δ می نامیم ،

ادعای ۱ : بیشترین هزینه این الگوریتم از 2Δ بیشتر نیست ، اثبات به سادگی قابل انجام است ، چون که z دورترین نقطه از مراکز بود ، بنابراین فاصله بقیه نقاط از مرکز خودشان حداکثر Δ خواهد بود ، چون که نا مساوی مثلثی را داریم ، می توانیم ادعا کنیم که قطر یک کلاستر بیشتر از 2Δ نیست ، زیرا شعاع آن حداکثر Δ است.و طبق نا مساوی مثلثی عرض آن حداکثر می تواند برابر یا کوچکتر از $\Delta+\Delta$ باشد.

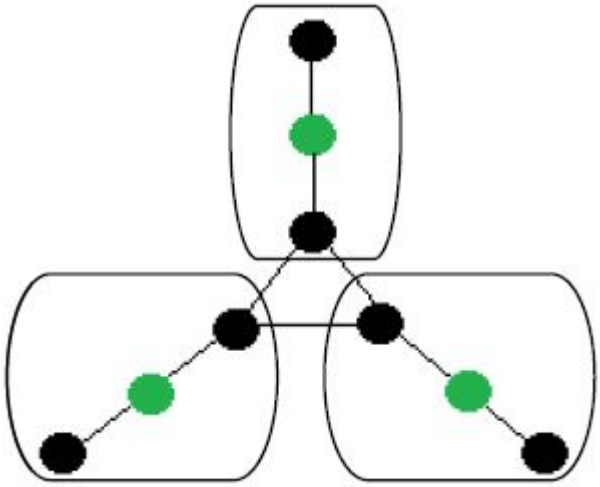
ادعای ۲ : هزینه OPT حداقل Δ است ، اثبات فرض کنیم که k+1 نقطه داریم ، فاصله نقطه z از هر مرکز حداقل Δ است ، با توجه به روند الگوریتم که هر بار دور ترین نقطه را به عنوان مرکز انتخاب می کرد می دانیم که این فاصله کمترین فاصله ممکن است ، طبق اصل لانه کبوتری حداقل ۲ نقطه در یک خوشه قرار خواهند گرفت ، و می دانیم که کمترین فاصله ممکن هم Δ است ، بنابراین جواب OPT حداقل بزرگتر یا مساوی Δ خواهد بود ، $OPT \geq \Delta$

ضریب الگوریتم : $\Delta \leq OPT$ در نتیجه $2\Delta \leq 2OPT$ و می دانیم که هزینه الگوریتم ما کمتر یا مساوی 2Δ است پس ضریب الگوریتم ۲ است.

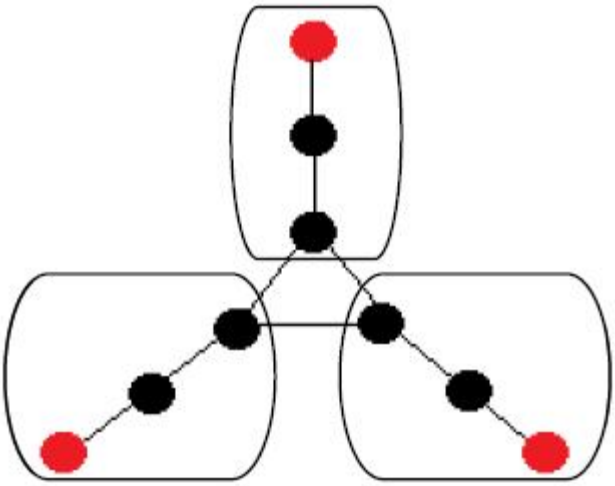
مثال tight :



فرض کنیم که فاصله همه از هم d است.



حالت بهینه ، که قطر برابر 2d است.



جواب الگوریتم ما که قطر برابر 4d است

۲. اثبات اینکه الگوریتم ما بهینه ترین است و الگوریتم با ضریب کمتر وجود ندارد:

برای این منظور از reduction استفاده می کنیم ، از مسئله Dominating Set که می دانیم NP است استفاده می کنیم ، یک نمونه از این مسئله را روی یک گراف G(V,E) با n راس و یک عدد مثبت k را در نظر می گیرم ، از روی این گراف یک گراف کامل G' می سازیم . راس ها همان راس های گراف G هستند ، اما یالها را اینگونه می سازیم : اگر در گراف G یک یال داشتیم در گراف G' یک یال به وزن 1 قرار می دهیم و اگر یالی وجود نداشت یالی با وزن ۲ قرار می دهیم ، توجه داریم که این گراف نا مساوی مثلثی را ارضا می کند حال در این گراف داریم :

- اگر در گراف G یک DS با اندازه کمتر یا مساوی k داشته باشیم ، گراف G' حتماً k خوشه با وزن ۱ دارد و

- اگر در گراف G یک DS با اندازه بیشتر از k داشته باشیم ،جواب بهینه در گراف k G’ خوشه با وزن ۲ است.

در حالت اول اگر یک الگوریتم با ضریب تقریب ε-2 داشته باشیم ، الگوریتم نباید یالهـای بـا وزن ۲ را بپـذیرد و در ایـن حالت جوابی با وزن ۱ می دهد ، به عبارت بهتر الگوریتم ما می تواند DS را تشخیص دهد ، یعنی یک مسئله NP را بـه وسیله یک الگوریتم P تشخیص می دهیم و این غلط است مگر اینکه P=NP باشد!!

پایان