ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان 99]

جلسه ۲۹: درخت پوشای کمینه و hash

نگارنده: رضا پیشکو

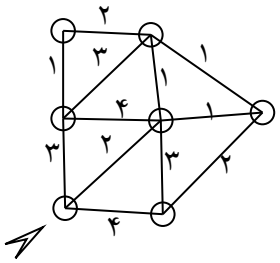
در جلسه قبل الگوریتم هایی برای پیدا کردن درخت پوشا در درخت های وزن دار گفته شد. و یک سری الگوریتم برای این کار ارایه شد که یکی از آنها Kruskal بود که در هر مرحله یال های با کم ترین وزن را بر میدارد به طوری که دور تشکیل نشود. و الگوریتمی دیگر که در این جلسه معرفی می کنیم prim است.

۱ الگوریتم prim

داده ساختار مورد استفاده در این الگوریتم هیپ و لیست است. که لیست برای نگه داری یال های موجود در درخا پوشا(همان مجموعه ی فعلی در توضیحات زیر) و استفاده هیپ نیز توضیح داده شده است. این الگوریتم به این طریق است که از یک راس شروع می کند و کمترین یال از مجموعه ی راس های فعلی به بیرون را به درخت پوشا مینیمم اضافه میکند.

برای مینیمم یال پیدا کردن از مین هیپ استفاده میکنیم. در هر مرحله ممکن است تعدادی یال کاندید به مین هیپ اضافه یا از آن حذف شوند. در واقع با اضافه شدن هر راس جدید یال هایی که از آن خارج می شوند به هیپ اضافه می شود. هر بار که مینیمم را از هیپ خارج میکنیم ممکن است از داخل به بیرون این مجموعه نباشد تا جایی حذف مینیمم را انجام می دهیم که یالی از داخل به بیرون مجموعه انتخاب شود.

مثال:

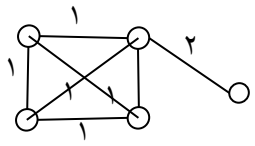


اگر از راس مشخص شده شروع کنیم یال هایی با انداره ۲ و ۳ و ۴ از آن خارج می شوند که طبق الگوریتم یال ۲ را انتخاب می کنیم. در ادامه یال هایی کی می توانیم انتخاب کنیم یال های خارج شده از هر دو راس موجود در مجموعه فعلی هستند که یال با اندازه ۱ که از راسی که تازه اضافه شده خارج شده است مینیمم بین یال های خارج شده از این مجموعه است و همین روند ادامه پیدا میکند تا همه راس ها به مجموعه اضافه شوند. ⊠

۲ تحلیل اردر الگوریتم prim

اگر درخت زیر را در نظر بگیرید که یک درخت کامل است که وزن هر یال آن ۱ است و یک یال دو به آن اضافه شده. اگر الگوریتم را روی این درخت انجام دهیم تمام یک ها باید خارج شوند تا بالاخره یال ۲ بتواند اضافه شود. در نتیجه e تا درج مینیمم داریم. و e تا هم حذف مینیمم داریم که هر کدام $\log e$ هزینه دارد زیرا هیپ حداکثر e راس دارد. و اضافه شدن یال انتخاب شده به انتهای لیست هم $O(1)$ است. پس اردر الگوریتم برابر است با $e.\log e$.

در تحلیل بالا در نظر گرفته شد که e از v بزرگ است زیرا در غیر این صورت درخت پوشا نداریم اما در حالت کلی اردر الگوریتم بالا است $eloge + v$.



۳ بهینه سازی الگوریتم prim

این بار مین هیپ به جای یال راس نگه می دارد و برای هر راس خارج از مجموعه فعلی نگه میدارد که مینیمم مقدار یالی که از مجموعه فعلی به آن راس هست چند است. و برای هر راس هم اشاره گر راس در مین هیپ که نشان می دهیم راس انتخاب شده است یا نه را نگه می داریم. در واقع می دانیم که هر راس در مین هیپ کجا قرار گرفته است. و اگر زمانی خواستیم مقدار راسی را کم کنیم میتوانیم روی آن راس decrease key اجرا کنیم. در ابتدا مین هیپ یک راس دارد. که راسی است که میخواهیم الگوریتم را از آن شروع کنیم و وزن آن صفر است. در گام اول این راس از مین هیپ حذف می شود و همسایه های آن اضافه می شوند(زیرا راس هایی هستند که از مجموعه راس های فعلی به آنها یال داریم) پس در کل وارد شدن راس به داخل مین هیپ v بار انجام می شود که از اردر $v.\log v$ است. عملیات دیگری که در این الگوریتم انجام می دهیم استخراج راس است. مانند گام اول که راس اولیه استخراج شد و راس های همسایه آن اضافه شدند. که این عملیات نیز v بار انجام مي شود که در کل از اردر $v.\log v$ است. در ادامه هر بار که یک راس را از هیپ خارج می کنیم چک میکنیم که آیا باعث کوچک تر شدن طرف مقابل یال هایی که از آن راس خارج شده اند می شود یا نه. بعضی از این یال ها ممکن است به داخل مجموعه فعلی باشند که اردر یک چک می شوند پس در کل در اردر e انجام مي شود زیرا هر یال حداکثر یک بار چک می شود. حالا ممکن است که مقدار تمام این راس ها کمتر شود و برای همه decrease key انجام دهیم که هزینه آن $e.\log v$ است. زیرا هر یال فقط یک بار چک می شود که مقدار آن کمتر شده است یا نه.

پس در کل اردر الگوریتم این بار برابر با $e.\log v$ مي شود که فقط در یک ضریب ۲ با الگوریتم قبلی متفاوت است. اما اگر از فیبوناچی هیپ استفاده کنیم decrease key در اردر یک انجام می شود. و عملیا های decrease key به جای $e.\log v$ در اردر e انجام مي شوند. و اردر الگوریتم پریم $v.\log v + e$ مي شود.

۴ Hash

مجموعه ی $D = v_1, v_2, \ldots, v_n$ را که اعضای آن لزوما عدد نیستند در نظر بگیرید. تابع هش تابعی پوشا است که اعصای مجموعه ما را به اعداد طبیعی می برد. و معمولا بازه ای که مجموعه به آن می رود بازه ای محدود است اما می تواند محدود نباشد. از دیگر ویژگی های این تابع یک طرفه بودن آن است یعنی اگر عضوی از مجموعه را به آن بدهیم خروجی را در اعداد طبیعی می دهد یا به عبارتی اگر x را به آن بدهیم $f(x)$ را خروجی می دهد اما اگر $f(x)$ را داشته باشیم نمی شود x معادل آن را یافت. تابع هش خوب تابعی است که تا جای ممکن

یونیفرم باشد و تا جای ممکن از دید بیرونی رندوم باشد زیرا باعث می شود که در هر مسئله ای کاربرد داشته باشد و نسبت به مسئله خاصی ضغیف نباشد چون اگر بدانیم تابع هش چیست ممکن است برای یک سری از مسئله ها خوب نباشد مثلا اعداد طوری در آن مسئله پخش شده باشند که تابع هش تعداد زیادی عدد را به یک خانه بفرستد. ولی اگر تابع مستقل از مسئله ارایه شده باشد و کسی که مسئله را ارایه می دهد دانشی نسبت به تابع هش نداشته باشد مثل این است که تابع هش به صورت تصادفی نسبت به مسئله توزیع شده و در این حالت ما فقط می توانیم تحلیل تصادفی انجام دهیم. در این شرایط علاوه بر اینکه تابع هش یک طرفه هست اگر تصادفی هم توزیع شده باشد آنگاه مسئله لازم نیست به این فکر کند که تابع هش چگونه است. ما به صورت تصادفی می دانیم که شانس همه عناصر در مسئله یکسان است. که به این universality هش مي گويند.

مثال: نمونه اي از شبه رندوم (pseudo random) این است که یک عدد اول خیلی بزرگ در نظر می گیرند. فرض کنید که به این صورت عمل میکند که v را آن عدد بزرگ(p) ضرب میکنیم و به فضای هش (|H|) باقي مانده میگیریم.

از دید کسی که بیرون است و در مورد p نظری ندارد. تقریبا در مورد رابطه چیزی نمی تواند بفهمد. ولی اگر بداند که فرمول ما از این جنس است می تواند p را به دست آورد. خاصیت p این است که از |H| بزرگ تر باشد و دست نیافتنی باشد. زیرا اگر که از H کمتر باشد میتوان تا قبل از |H| الگو و تکرار پیدا کرد ☒