فصل پنجم

رویکرد حریصانه

(Greedy Approach)

روش حریصانه – مقدمه

- یک الگوریتم حریصانه، به ترتیب عناصر را انتخاب کرده، هر بار آن عنصری را که طبق معیاری معین «بهترین» به نظر می‌رسد، بدون توجه به انتخاب‌هایی که قبلا انجام داده یا در آینده انجام خواهد داد، بر می‌دارد.
- الگوریتم‌های حریصانه، غالبا برای حل مسائل بهینه‌سازی به کار می‌روند.
- در روش حریصانه، تقسیم به نمونه‌های کوچک‌تر صورت نمی‌پذیرد.
- الگوریتم حریصانه با انجام یک‌سری انتخاب، که هر یک در لحظه‌ای خاص، بهترین به نظر می‌رسد، عمل می‌کند، یعنی انتخاب در جای خود بهینه است. امید این است که یک حل بهینه سراسری یافت شود، ولی همواره چنین نیست.
- برای یک الگوریتم مفروض باید تعیین کرد که آیا حل همواره بهینه است یا خیر.
- نتیجه نهایی مجموعه‌ای از داده‌ها است که ممکن است ترتیب آن‌ها نیز اهمیت داشته باشد.
- مجموعه جواب به صورت مرحله‌ای است و در هر مرحله یک مولفه از جواب حاصل می‌شود.
- جواب نهایی باید تابع هدف را بهینه کند ماکزیمم یا مینمم
- تصمیم نهایی در مورد انتخاب یا عدم انتخاب توسط روال Select جواب قطعی و غیر قابل بازگشت می‌باشد.
- الگوریتم حریصانه، کار را با یک مجموعه تهی آغاز کرده و به ترتیب عناصری را به مجموعه اضافه می‌کند تا این مجموعه جوابی برای نمونه‌ای از یک مسئله را نشان دهد

ویژگی‌ها

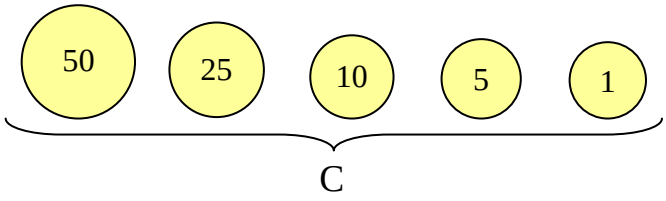
۱. روال انتخاب (Select)، برای انتخاب مولفه‌های بعدی جواب از مجموعه انتخاب‌های ممکن
۲. بررسی امکان‌سنجی (Feasible)، تعیین می‌کند که آیا مجموعه جدید برای رسیدن به جواب، عملی است یا خیر.
۳. بررسی راه حل (Solution) برای بررسی اینکه مشخص کند در نهایت جواب حاصل شده است یا خیر.
۴. یک تابع هدف: هدف، بهینه کردن این تابع است.

الگوریتم روش حریصانه

set greedy (C)	Greedy (c) : نام الگوریتم (C مجموعه اولیه است)
{	
s = ∅	
while (!solution(s) && c! =∅)	تا زمانیکه S به جواب مورد نظر نرسیده است و c هم هنوز خالی نیست
{	
X = select (c);	یک عنصر از C بردار و در x بریز.
c = c - {x};	مقدار x را از c کم کن در c بریز
if (feasible (s,x))	بررسی کن آیا با اضافه شدن x به S جواب بدست می‌آید یا خیر
s = s ⋃ {x}	اگر درست بود x را به s اضافه کن
}	
if (solution (s))	
Return s;	
else return ∅;	
}	

مسئله خرد کردن پول

میخواهیم باقی پول مشتری را با حداقل تعداد سکه‌ها، پس بدهیم. در این مثال، میخواهیم ۵۷ سنت را به مشتری پس بدهیم:



- $\overbrace{\{\}}^{S}$ اولیه
- $\{50\}$
- $\{50,50\}$
- $\{50,25\}$
- $\{50,10\}$
- $\{50,5\}$
- $\{50,5,5\}$
- $\{50,5,1\}$
- $\overbrace{\{50,5,1,1\}}^{S}$ جواب

تحلیل

- آیا این روش همیشه جواب می‌دهد. مثلا اگر اندازه سکه‌ها به شکل دیگری بود، ممکن است جواب ندهد.
- روش حریصانه در مورد هر مسئله‌ای جواب نمیدهد. بلکه باید اثبات شود.

درخت‌های پوشای کمینه

مثال: فرض کنید، طراح شهری می‌خواهد چند شهر معین را با جاده به هم وصل کند، به صورتی که مردم بتوانند از هر شهر به شهر دیگر بروند. اگر محدودیت بودجه‌ای وجود داشته باشد، ممکن است طراح بخواهد این کار را با حداقل مقدار جاده‌کشی انجام دهد.

- برای مسائلی مانند مسئله فوق دو الگوریتم حریصانه متفاوت پریم (Prim) و کروسکال (Kruskal) بررسی می‌شود.
- هر یک از این الگوریتم‌ها از یک ویژگی بهینه محلی استفاده می‌کند.
- ثابت می‌شود که الگوریتم‌های کروسکال و پریم همواره درخت‌های پوشای کمینه را ایجاد می‌کنند.
- نکته: تضمینی وجود ندارد که یک الگوریتم حریصانه همواره جواب بهینه بدهد.

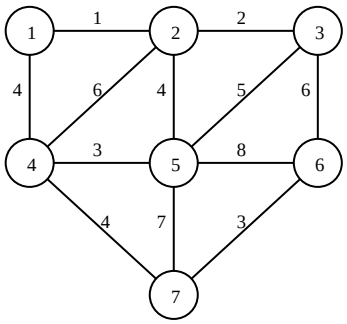
تذکر: الگوریتم پریم و کروسکال (بسیار مهم) است

الگوریتم پریم

الگوریتم پریم با زیر مجموعه‌ای تهی از یال‌های T و زیر مجموعه‌ای از رئوس B آغاز می‌شود، زیر مجموعه حاوی یک راس دلخواه است. به عنوان مقدار اولیه، {v1} را به B می‌دهیم. نزدیک‌ترین راس به B، راسی در N - B است که توسط یالی با وزن کمینه به راسی در B متصل است.

پریم: نام الگوریتم g گراف n تعداد راس ها (گره‌عل) a م جموعه یالها SET OF edge: مجموعه‌ای از یالها	Functionprim (G=(N,A): graph, length: $A \to R^+$): set of edge
مقدار دهی‌های اولیه انجام می‌شود.	{ initialization}
مقدار اولیه T خالی باشد. (T بعد از اجرای الگوریتم، حاوی یال‌ها خواهد شد.)	$T \leftarrow \varnothing$
یک گره (راس) از N در B قرارداده می‌شود. نکته: B در نهایت حاوی تمام گره‌های گراف خواهد شد.	$B \leftarrow$ {an arbitray member of N}
تا زمانیکه تعداد گره‌های موجود در B به تعداد گره‌های موجود در N نرسیده است، حلقه را تکرار کن.	while $B \neq N$ do
از بین یال‌های کاندید که گره سمت چپ آن‌ها متعلق به مجموعه B و گره سمت راست آن، متعلق به مجموعه $N - B$ است، یال با کمترین هزینه را انتخاب می‌نماید و در متغیر e قرار می‌دهد.	find e ={ u, v} of minimum length such that $u \in B$ and $v \in N - B$
یال انتخاب شده به مجموعه T اضافه می‌شود.	$T \leftarrow T \cup \{e\}$
گره سمت راست یال (v) به مجموعه B اضافه می‌شود.	$B \leftarrow B \cup \{v\}$
	end whille
در انتهای الگوریتم، مجموعه T که حاوی چندین یال است، به محل فراخوانی الگوریتم، برگردانده می‌شود.	return T

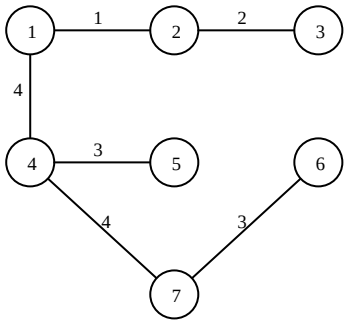
مثال: درخت پوشای کمینه درخت زیر را با استفاده از الگوریتم پریم رسم نمایید:



در واقع این یال‌ها، درخت پوشای مینمم را تشکیل خواهند داد. (برای گره‌های موجود در B)

Step	یالها T $\{u,v\}$	گره‌ها { } B
initialization	-	$\{1\}$
1		$\{1,2\}$
2	$\{2,3\}$	$\{1,2,3\}$
3	$\{1,4\}$	$\{1,2,3,4\}$
4	$\{4,5\}$	$\{1,2,3,4,5\}$
5	$\{4,7\}$	$\{1,2,3,4,5,7\}$
6	$\{7,6\}$	$\{1,2,3,4,5,6,7\}$

درخت پوشای کمینه مثال بالا:



پیچیدگی زمانی

عمل اصلی: در حلقه While دو حلقه (یکی برای بررسی اینکه کدام گره نزدیکترین به گره مورد نظر است و دیگری برای بروزرسانی کردن فاصله هر گره از مجموعه جواب، که آن گره در مجموعه جواب وجود ندارد) وجود دارد که هر یک $(n-1)$ بار تکرار می‌شود. عمل یافتن از میان لبه‌های متصل به یک مجموعه $n-1$ بار اتفاق می‌افتد، که در مجموع برابر n^2 می‌شود. اندازه ورودی، n است که همان تعداد رئوس است.

$$T(n)=2(n-1)(n-1)\in\theta(n^2)$$

به دلیل آنکه، داخل حلقه while کی حلقه for لازم است و دو حلقه تو در تو از مرتبه n^2 می‌باشد.

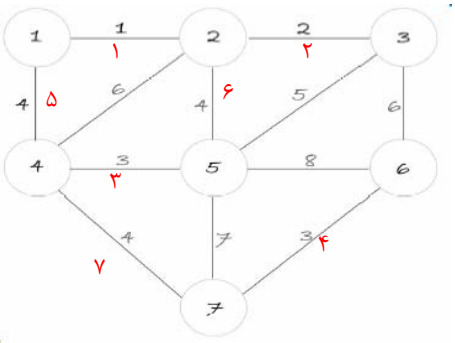
نکته:

برای محاسبه پیچیدگی زمانی، حلقه های تو در تو ضرب و حلقه‌های کنار هم، جمع می‌شوند

الگوریتم کراسکال

نام الگورتیم: کراسکال N: مجموعه گره های گراف A: مجموعه یال‌های گراف خروجی الگوریتم: مجموعه‌ای از یال‌ها (مجموعه T)	Function Kruskal (G=(N,A):graph; length: A → R): set of edges
مقداردهی‌های اولیه انجام می‌شود	{initialization}
یال‌های گراف بر اساس هزینه و به صورت صعودی مرتب می‌شوند	sort A by increasing length
تعداد گره‌های گراف N در متغیر n قرار می‌گیرد	n ← the number of nodes in N
تهی در T در قرار داده می‌شود. مجموعه T شامل یال‌هایی خواهد شد که درخت پوشای مینیمم را برای گره‌ها، ایجاد می‌کند.	T ← ∅ {will contain the edges of the minimum spanning tree}
به تعداد گره‌های گراف (به تعداد n) مجموعه تک عنصری ایجاد می‌کند که در هر مجموعه، یک گره از گراف قرار دارد	initialize n sets, each containing a different member of N
تکرار حلقه حریصانه	{greedy loop}
حلقه rapeat از اینجا شروع میشود و در ردیف until پایان می‌یابد	repeat
یالی که دارای کمترین هزینه است و تا کنون برای انتخاب کاندید نشده است، کاندید می‌شود و در متغیر e قرار می‌گیرد	e ← {u,v} ← shortest edge not yet considered
مشخصه مجموعه‌ای که گره u در آن قرار دارد، استخراج و در ucomp قرار می‌گیرد.	ucomp ← find (u)
مشخصه مجموعه‌ای که گره v در آن قرار دارد، استخراج و در vcomp قرار می‌گیرد.	vcomp ← find (v)
اگر مشخصه مجموعه گره u با مشخصه گره v متفاوت باشد، در آن صورت :	if ucomp ≠ vcomp then
دو مجموعه با هم ادغام می‌شوند	merge (ucomp, vcomp)
یال مورد نظر به مجموعه T اضافه می‌شود در غیر این صورت یال مورد نظر، مورد قبول واقع نمی‌شود	T ← T ⋃ {e}
حلقه repeat در اینجا در صورتی پایان میابد که: تا زمانی که تعداد یالهای موجود در T کمتر از n-1 باشد، تکرار خواهد شد	until T Contains $n-1$ edges
مجموعه T که حاوی $n-1$ یال است به محل فراخوانی الگوریتم، برگردانده می‌شود.	return T

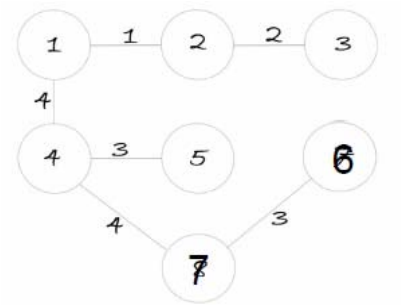
مثال: درخت پوشای کمینه درخت زیر را با استفاده از الگوریتم کراسکال بدست بیاورید.



عددهای قرمز مراحل جدول زیر است:

مرحله	یال مورد نظر جهت انتخاب	گره‌های متصل شده به هم	توضیحات
Step	edge considered	connected component	
initialization	-	{1} {2} {3} {4} {5} {6} {7}	گره‌های گراف که هنوز به هم متصل نیستند
1	{1,2}	{1,2} {3} {4} {5} {6} {7}	
2	{2,3}	{1,2,3} {4} {5} {6} {7}	
3	{4,5}	{1,2,3} {4,5} {6} {7}	
4	{6,7}	{1,2,3} {4,5} {6,7}	
5	{1,4}	{1,2,3,4,5} {6,7}	
6	{2,5}	rejected	به دلیل ایجاد شدن حلقه یال پذیرفته نمی‌شود چون هر دو قبلا در مجموعه متصل قرار داشته‌اند
7	{4,7}	{1,2,3,4,5,6,7}	مجموعه حاوی کل گره‌ها در این مرحله حاصل شده است و

جواب:



پیچیدگی زمانی الگوریتم کراسکال

عمل اصلی: یک دستورالعمل مقایسه

اندازه ورودی: n، تعداد رئوس و m تعداد یال‌ها

تحلیل:

- زمان لازم برای مرتب سازی یال‌ها: $W(m)\in\theta(m\log m)$
- زمان در حلقه repeat: $W(m)\in\theta(m\log m)$
- زمان لازم برای ایجاد n مجموعه مجزا: $T(n)\in\theta(n)$
- در کل:

$$W(m,n)\in\theta(m\log m)=\theta(n^2\log n)\qquad when\quad m=\frac{n(n-1)}{2}$$

زمانی که تعداد یال‌ها، زیاد (حداکثر) باشد، این شرایط برقرار خواهد بود.

مقایسه الگوریتم کراسکال و پریم

	پریم	کروسکال
پیچیدگی زمانی	$\theta(n^2)$	$\theta(n^2\log n)$

در هر انتهای بازه زیر:

$$\overbrace{n-1}^{\alpha}\le m\le\overbrace{\frac{n(n-1)}{2}}^{\beta}$$

α: درخت پوشای مینیمم (یعنی حداقل تعداد یال‌های برای اتصال به هم)

β: درخت پوشای ماکزیمم (یعنی حداکثر تعداد یال‌های برای اتصال به هم)

m: تعداد یال‌ها

n: تعداد گره‌ها

پس سرعت اجرای الگوریتم در ۲ الگوریتم به صورت زیر خواهد بود:

	پریم	کروسکال
اگر m نزدیک به α	کندتر	سریعتر
اگر m نزدیک به β	سریعتر	کندتر