Subject:
Yare: Month: Day: Page: ( )

* ماتروید ها (Matroid):

- همان مجموعه های مستقل با یک خصوصیت اضافه تر هستند:

(۱)- این که اگر X مجموعه باشد که یکی تعدادش از دیگری بیشتر باشد. (مثلاً F یک مجموعه ی مستقل است به گونه ای که X و Y عضو F هستند و X تعداد بیشتری از Y دارد) ⟸ آن گاه حتماً یک چیزی در X هست که در Y نیست و اگر آن را به Y اضافه کنیم، مجدداً مجموعه ی جدید که حاصل می شود در F خواهد بود *

- مثلاً مجموعه ی زیرگراف های جنگلی یک گراف و یا مجموعه ی زیرمجموعه های مستقل سطرهای یک ماتریس، این خصوصیت را دارا هستند *

(به عنوان مثال مجموعه ای شامل 10 بردار مستقل و مجموعه ی دیگر شامل 11 بردار داریم که اتفاقاً هر دو درون سیستم مورد نظر نیز هستند و مستقل اند، حتماً سطری در مجموعه ی اول وجود دارد که اگر به مجموعه ی 11 تایی اضافه شود، مجموعه ی حاصل همچنان مستقل باقی خواهد ماند.)

(یا اگر یک گراف داشته باشیم که جنگل باشد (A) و یک گراف جنگل دیگر با تعداد یال های کمتر دیگر نیز داشته باشیم (B)، حتماً یکی از یال های A را می توان به B اضافه کرد به گونه ای که مجموعه ی حاصل درخت باقی بماند)

- این ماتروید ها، یک سری ویژگی هایی خاص دارند که منجر به بهینه سازی روی آن ها در زمان چند جمله ای می توان انجام داد

⟸ پس اگر F، مجموعه ای از زیرمجموعه های E باشد آن گاه (F, E) را یک مجموعه ی مستقل نامیم اگر نسبت به عملگر زیرمجموعه بسته باشد و در نهایت یک ماتروید هستند اگر یک خصوصیت اضافه تر نیز داشته باشند.

Maral

Subject:
Yare: Month: Day: Page: ( )

* جلسه بیست و چهارم:

* یک الگوریتم را از چند منظر می‌توان ارزیابی کرد:

۱) سرعت
۲) دقت
۳) میزان حافظه‌ی مورد نیاز
۴) سادگی از منظر پیاده‌سازی
۵) امکان پیاده‌سازی در شرایط ویژه
۶) پایایی

پایایی ← پایایی موازی
پایایی ← پایایی ابری

* ماشین تورینگ: (فصل ۱۵ وین) (صفحه ۳۰۸)
- یک مدل محاسباتی است *

* حالت کلی منظور از الگوریتم همان optimization procedure است *

* هرچند حدالی که بتوان در برنامه‌نویسی کد کرد، می‌توان برای آن الگوریتمی براساس ماشین تورینگ نیز ارائه داد *

* یکی از راه‌های توصیف دقیق الگوریتم، تعریف آن براساس ماشین تورینگ است.

* سرعت:
- یعنی الگوریتم چقدر سریع ما را به هدف می‌رساند *(چقدر سریع این روبه را حل می‌کند)
- هرچقدر از تصمیمات استراتژیک به سمت تصمیمات عملیاتی می‌رویم، این فاکتور، اهمیت بیشتری پیدا می‌کند *

* دقت:
- یعنی این که روبه‌ی محاسباتی که ما استفاده می‌کنیم، چقدر ما را به هدف نزدیک می‌کند.
- ~~دقت الگوریتمی خواهیم با یک الگوریتم، اعداد اول را تا n را به دست آوریم، آن~~

Maral

فقط ۲۰٪ از اعداد اول تخمین داده شود، اصطلاحاً دقت الگوریتم را ۸۰٪ می‌نامیم.

Subject:
Yare: Month: Day:
Page: ( )

* سادگی از منظر پیاده سازی:

یعنی - اگر ۲ الگوریتم کاملاً مشابه باشند، الگوریتمی انتخاب می شود که ساده تر قابل پیاده سازی شود.

* امکان پیاده سازی در شرایط ویژه:

اگر سیستمی داریم که پردازش موازی یا توزیع شده (قسمتی از اطلاعات بر روی یک سرور و برخی اطلاعات بر روی سرور دیگر است) و یا پردازش ابری دارد، آیا الگوریتم بر روی هر سه نوع این پردازش ها قابل پیاده سازی است ؟

* پایایی محاسباتی:

- می دانیم کامپیوتر، عدد $\sqrt{2}$ را همان $\sqrt{2}$، ذخیره نمی کند بلکه حتماً آن را به یک ترکیب ۱۶ رقمی از اعداد تبدیل خواهد کرد و سپس ادامه محاسبات را براساس این ترکیب جدید انجام خواهد داد. حال اگر دقت محاسباتی و تبدیل کامپیوترها با یکدیگر متفاوت باشد و یک الگوریتم نسبت به تمام اعداد استفاده شده در آن حساس باشد، ممکن است در صورت اجرا با کامپیوترهای مختلف، جواب متفاوتی به دست آوریم *

* جلسه بیست و پنجم:

* سرعت الگوریتم ها:

- اگر یک شرایط و case مشخص داشته باشیم، وقتی یک الگوریتم را Run می کنیم، سرعت اجرای آن به عوامل زیر بستگی دارد:

۱- نسخه ی نرم افزار *
۲- کد الگوریتم ها *
۳- سیستم عامل و سخت افزار *
۴- نسخه ی خاص از سیستم که در حال تست آن هستیم *
۵- نرم افزارهای در حال اجرا حین تست کد نوشته شده *
۶- اتصال به اینترنت *

Maral

Subject:
Yare: Month: Day: Page: ( )

تجربی:

- مثلاً یک سری نمونه های خاص را بر روی یک کامپیوتر کامل حساب Run می کنیم. (در این حالت با پارامتر زمانی کار نمی کنیم و فقط مثال عددی را حل می کنیم)
- ابتدا از مسئله، یک سری Instance تهیه می کنیم.

$I_1, I_2, \ldots, I_n$ ← (نمونه های 1 تا n)

← پس این نمونه ها را در کد نوشته شده وارد می کنیم و پس مسئله را Run می کنیم. ← فقط باید حواسمان باشد حتماً وقتی می خواهیم چند کد را در این حالت با هم مقایسه کنیم، تمامی فاکتورها (همه ی قبل) در همه ی نمونه ها یکسان باشد.

* این رویکرد، رویکرد غالب Operation ReSearch و CCS است
(Computational computer Science)

تعیین سرعت حل مسئله

تحلیلی:

- می گوییم رویکرد فوق به دلیل وابستگی به فاکتورهای مختلف، خیلی ضعیف است و به جای آن که با حل چندین نمونه، دو الگوریتم را مقایسه کنیم، با در نظر گرفتن کلاسی از نمونه های مسئله ی P (البته بهترین کلاس، کل نمونه های مسئله ی P می باشد که البته به دلیل زیاد بودن آن ها، قابل در نظر گیری نخواهد بود) به مقایسه ی الگوریتم های مختلف خواهیم پرداخت *
- این الگوریتم در Theoritical computer Science، کاربرد می باشد *

- کلاس های انتخابی از مسئله ی P، Property Base می باشد *
- این رویکرد تحلیل فوق العاده خوب است ولی ممکن است نتوانیم همیشه این تحلیل را انجام دهیم و یا اگر هم بتوانیم چنین تحلیلی داشته باشیم، چون کلاس گرفته شده، خیلی وسیع است، ممکن است بسیار دور از ذهن باشد *

Maral

← توضیحات بیشتر:

- در بررسی تجربی، باید سعی کنیم نمونه‌هایی که مورد بررسی قرار می‌دهیم، حتماً تمامی جوانب را در بر بگیرد و البته نمونه‌هایی نباشد که هستند با در نظر گیری آن‌ها، سریع‌تر حل شود؛ به عبارتی باید بررسی کاملاً fair صورت گیرد. همچنین باید حالت‌های مختلفی که می‌تواند در مسئله تأثیرگذار باشد را هم در نظر بگیریم. مثلاً در مسئله‌ی Location-Allocation می‌دانیم اگر هزینه‌ی احداث بسیار بالا باشد، تسهیلاتی که open می‌شوند کم خواهد بود و در غیر این صورت، زیاد می‌شود ← لذا باید در نمونه‌های عددی، حتماً هم هزینه‌ی ثابت بالا و هم هزینه‌ی ثابت کم را مورد بررسی قرار دهیم.
(چرا که می‌دانیم اگر هزینه‌ی ثابت احداث کم باشد، تعداد تسهیلات بیشتری open می‌شود و لذا سرعت حل مسئله کاهش می‌یابد)

Fair Test generation

- اعداد در نمونه‌ها، باید Reasonable باشد. (اعداد هم‌خوانی داشته باشد)
- Scale اعداد، افق زمانی آن‌ها باید یکسان و معقول باشد*
- بُعد اعداد (Dimension) باید به گونه‌ای باشد که با Dimension، عددهایی هم‌خوانی داشته باشد*
- ضریب تغییرات باید محاسبه شود و هرچقدر کم‌تر باشد، الگوریتم قوی‌تر خواهد بود*
- الگوریتم‌هایی که بر مبنای Randomize Optimization هستند، مانند الگوریتم‌های فراابتکاری (فراابتکاری)، چون در هر بار Run کردن، نتایج متفاوتی خواهند داشت، باید تعداد دفعات مشخص الگوریتم، Run شود و در این تعداد مشخص، میانگین جواب (Average) و بهترین جواب (Best solution) به دست آمده گزارش شود*
- هر الگوریتم یک سری پارامتر دارد (ژنتیک، Tabu Search، ...)؛ مثلاً این که اگر جواب از α بیشتر شد، رد کن و در غیر این صورت بپذیر و ... که باید حواسمان باشد این پارامترها نیز حین مقایسه، یکسان فرض شود. (با الگوریتمی که فرد دیگری ارائه داده است) و همچنین این پارامترها انتخاب شود که الگوریتم، بهترین جواب را تولید کند ← به این مبحث، Parameter tuning گویند*

Subject:
Yare: Month: Day:
Page: ( )

* چرا Parameter tuning با Design الگوریتم کاملاً متفاوت است *

* بررسی تحلیلی نیز به طور کلی به ۲ دسته تقسیم می‌شود:

۱- رویکرد تحلیل بدترین حالتی: (worse case) ← الگوریتم را در بدترین حالت بررسی می‌کند

۲- رویکرد تحلیل هموار (Smooth) ← به صورت میانگین حالات مختلف با وزن‌دهی به تحلیل را انجام می‌دهد *

* مثلاً سیمپلکس، رویکرد تحلیل هموارش Polynomial Time است *

← در حالت بدترین حالت، n های بزرگ اهمیت بیشتری خواهند داشت و ارزش بیشتری پیدا خواهند کرد * ← پس الگوریتمی را انتخاب می‌کنیم که در بدترین حالت برای n های بزرگ، سایر الگوریتم‌ها را شکست دهد *

* چرا $O(n^2)$ ، $O(n^3)$ و ... برای complexity ها خوانده شد.

← O، توانایی بالاترین حد الگوریتم‌ها را برای حرکت به سمت بینهایت نشان می‌دهد *

* $a_3n^3 + a_2n^2 + a_1 + a_0 \longrightarrow O(n^3)$ ①

* $a_2n^2 + a_1n^1 + a_0 \longrightarrow O(n^2)$ ②

← اگر a_3 و a_2 هر دو مثبت باشند، بالاخره از یک جایی به بعد، رابطه ① رابطه ② را می‌گیرد. پس این حدی که ۲ رابطه برعکس می‌شوند، بسیار اهمیت دارند.

← هر الگوریتمی که $O(n^2)$ باشد، قطعاً $O(n^3)$ نیز هست و هر $O(n^3)$ ای، قطعاً $O(n^4)$ و ... هم هست. حال اگر یک order دیگر به عنوان قابل بهبود نباشد ~~به کوچکترین کران بالا برسیم و آن را~~

Maral

با $\Theta(\ )$ نمایش می‌دهیم. $O(n^n)$، یک کران بالاست و نه کوچکترین کران بالا!!!

خوانده شود ← o_function

* جلسه بیست و ششم:

- فرض کنیم f, g از N به R^+ تعریف می‌شوند.

برخی از کتاب‌ها این k را هم نمی‌گذارند

$f \lesssim g \quad f_n \in O(g(n)) \quad \exists k, D>0 \quad \forall n \geq k \quad f_n \leq D g(n)$

$f = g \quad f_n \in \Theta(g(n)) \quad \exists k, d, D>0 \quad \forall \geq k \quad d g(n) \leq f(n) \leq D g(n)$

$f \gtrsim g \quad f_n \in \Omega(g(n)) \quad \exists k, d>0 \quad \forall n \geq k \quad d g(n) \leq f(n)$

* شرایط کافی برای سه حالت فوق عبارت است از:

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Longrightarrow f_n \in O(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0 \Longrightarrow f_n \in \Theta(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Longrightarrow f_n \in \Omega(g(n))$

یا

$g(n) \in O(f(n))$

* اگر الگوریتمی تابع پیچیدگی‌اش از $O(n^a)$ باشد می‌گوییم این تابع Polynomial است *

Subject:
Yare: Month: Day: Page: ( )

* اول از همه باید تابع پیچیدگی را تشخیص دهیم و سپس بگوییم از چه order ای می‌باشد؟ یعنی کدام تابع چندجمله‌ای باشد که بزرگترین درجه‌ی تابع، همان order تابع می‌باشد و در سایر توابع مرتبه‌ی تابع پیچیدگی کی مشتق می‌شود.

$$\log < n < n\log n < n^{a} < n^{a+d} < b^{n} < n! < n^{n}$$
$$a>1 \qquad d>0$$

← چند نمونه از توابع پیچیدگی:

* $f \in O(g_1(n)+g_2(n)) \Rightarrow f \in O(\max\{g_1(n), g_2(n)\})$

* $f_1 \in O(g_1)$, $f_2 \in O(g_2)$ $\Rightarrow$ $f_1 + f_2 \in O(\max\{g_1(n), g_2(n)\})$, $f_1 \times f_2 \in O(g_1(n) \times g_2(n))$

* $2^{n} \in O(n!)$

* $\log_a n \in O(n)$; $a>1$

* $\log_a n \in \Theta(\log_b n)$; $a,b>$

* $n\log n \in O(n^{1+\varepsilon})$; $\forall \varepsilon > 0$

* $\log n! \in \Theta(n\log n)$

Maral

* اگر تابع $T(n)$ بازگشتی باشد:

$$T(n) = aT\left(\frac{n}{c}\right) + bn$$

$$T(n) \in \begin{cases} O(n) & a < c \\ O(n\log n) & a = c \\ O(n^{\log_c a}) & a > c \end{cases}$$

* problem و problem Instance (PI):

- اگر در مسئله‌ای که داریم، یک PI تعریف کنیم، آن‌گاه سایز Instance، همان تعداد بیت‌هایی است که برای رمزنگاری Instance نیاز داریم. یعنی:

$$P \longrightarrow I \qquad Size(I) = \sum_{i=1}^{N} \log_2(a_i + 1) \longrightarrow \text{سایز اصلی مسئله}$$

- اگر برای حل این نمود (Instance) $RT(I)$؛ واحد زمان نیاز داریم، خواهیم داشت:

$$f_A(m) = \max RT_A(I) \longrightarrow \text{زمان مورد نیاز برای حل نمود } I \text{ با الگوریتم } A$$

تابع پیچیدگی زمانی m با الگوریتم A

$I : Size(I) = m$

همه Iهایی که جمع تعداد بیت‌های آن‌ها برابر m باشد.

* سایز فرعی:

- حال چون سایز اصلی، تابع چند جمله‌ای از سایز فرعی است؛ این اجازه را خواهیم داشت که به جای سایز اصلی (که محاسبه‌ی آن شق القمر است) از سایز فرعی استفاده کنیم که بسیار ساده‌تر به دست می‌آید.

Maral

Subject:
Yare: Month: Day: Page: ( )

* مثلاً در MST، یکی از سایزهای فرعی تعداد رأس ها (n) و یکی از سایزهای فرعی دیگر تعداد یال ها (m) است *

* مثلاً در Tsp، سایز فرعی تعداد شهرهاست. (n)

$$Size(I) \in O((\text{سایز فرعی})^{K})$$

* عمل های اصلی تکثیر، جمع، ضرب، تقسیم، تفریق، رادیکال و ... (به غیر از توان) را همگی یکسان در نظر میگیریم * ← ادامل هر عملی که نهایت آن مشخص باشد را یکسان در نظر میگیریم

← مثلاً جمع تعداد زیادی عدد دستی A با دستی B، بلاخره یک مقاسیم نیاز داریم ولی پیدا کردن mean چند عدد، حتماً به تعداد اعداد بستگی دارد به همین دلیل از O(n) است *

~~Stable~~ Merge sort (۲۰٪)

$$T_{(n)} = 2T(\frac{n}{2}) + \Theta(n)$$

$$T(n) \in O(n \log n)$$

Maral

Subject:
Yare: Month: Day: Page: ( )

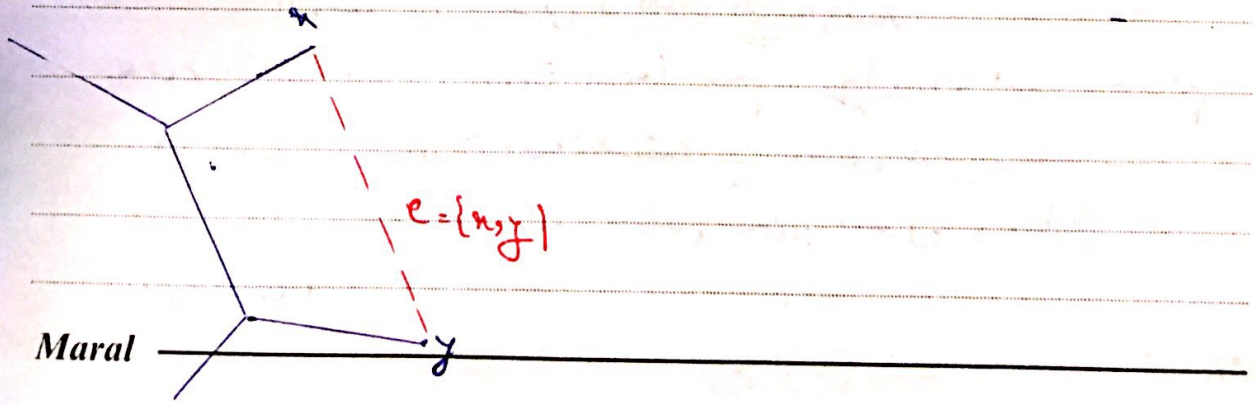
* فصل ۱، بخش ۲ ← Merge Sort (کتاب وین)
* Bobble Sort خوانده شود.
* الگوریتم کروسکال و Prime خوانده شود. ← فصل ۶ از کتاب وین (بخش ۶-۱)
* فصل ۱۷ کتاب وین ← بخش ۱۷-۱ و ۱۷-۲ خوانده شود
* الگوریتم های مبتنی بر مقایسه ی زوجی (مثلا ۲ و ۲) باشد، بهترین Order ای که می توان برای چنین الگوریتمی به دست آورد n log n است. بهتر نخواهد شد. ← مانند الگوریتم MergeSort *

← الگوریتم Linear programming باشد و تمامی ضرایب گویا باشد

ساختار الگوریتم MergeSort ← Divide and conquer

ساختار الگوریتم های کروسکال و پریم ← Greedy

* جلسه بیست و هفتم:
- فصل ششم کتاب وین:
* قضیه ی ۶-۳ ← شرایط لازم و کافی برای کمینه بودن یک درخت فراگیر. (با فرض b سرواک خواهیم داشت)



Subject:
Yare: Month: Day: Page: ( )

* الگوریتم کروسکال:

(1) فرض کنیم نام کتاری یال ها به ترتیب وزن آن ها باشد یعنی e_1 دارای کمترین وزن باشد
(2) اول یک درخت با تمام رئوس V و بدون هیچ گونه یالی تشکیل می دهیم.
(3) e_1، e_2 و ... را به ترتیب به شرطی اضافه کن که با مجموعه ی قبلی دور تشکیل ندهد

← این الگوریتم حریصانه است (Greedy) زیرا هر مرحله بهترین حالت را انتخاب می کند.

۶-۴ قضیه: اثبات کنید این الگوریتم همواره جواب بهینه خواهد داشت (دقت ۶.۳)
- از طریق تناقض.

← order الگوریتم کروسکال؟ $O(E)$

* الگوریتم Prime:

(1) ابتدا یک رأس انتخاب می کنیم
(2) تا زمانی که همه ی رأس ها انتخاب نشده:
- از بین این یال هایی که از این رأس خارج می شود، یال با کمترین وزن انتخاب شود به شرطی که تشکیل دور ندهد *

* این الگوریتم هم Greedy است. می باشد و هم local search
الگوریتم هایی که local search هستند یک همسایگی تعریف می کنند. مثلا این الگوریتم هر بار رأس های مجاور با رأس قبلی را با عنوان Neighbor در نظر می گیرد *

Maral

Subject:
Yare: Month: Day: Page: ( )

← یک تابع با تعداد پارامترهای $O(n)$ تنگ می‌باشد * ← که الگوریتم کروسکال برای چنین توابعی مناسب ترهست *

← فقط ۲ الگوریتم کروسکال و prime (تا قبل از دنیای فیبوناچی) و الگوریتم هایی که در تمرینات داده شده است، خوانده شود *
لذا لازم نیست تمام الگوریتم های چند جمله‌ای موجود در کتاب وبلن خوانده شود *

* الگوریتم های شبه چند جمله‌ای:
- این الگوریتم ها تحت شرایطی رفتار چند جمله‌ای و تحت شرایطی نیز رفتاری غیر از چند جمله‌ای خواهند داشت.

الگوریتم های چند جمله‌ای ← $O((size)^k)$ یا $O(m^k n^L)$
سایز اصلی — سایز فرعی

{ اعداد لازم برای ذخیره کردن مسئله } ← $M = max$ ← فرض کنید

* همواره در سایز اصلی مسئله (size) ؛ خود M وجود دارد

شبه چند جمله‌ای ← $O(M^L size^k)$

اگر M یک عدد ثابت باشد و یا از سایز چند جمله‌ای مسئله باشد؛ حاصل این عبارت نیز چند جمله‌ای خواهد شد

ولی اگر M ؛ چند جمله‌ای نباشد؛ حاصل این الگوریتم نیز؛ چند جمله‌ای نخواهد بود.

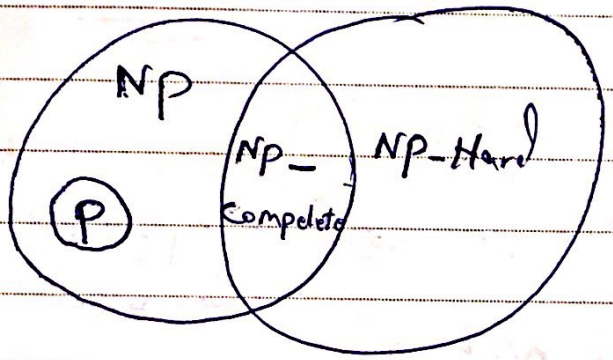
Maral

Subject:
Yare: Month: Day: Page: ( )

← به همین دلیل مسئله مسئله‌ای که پیچیده به صورت Np_Hard شناخته می‌شود زیرا هنوز هیچ الگوریتم چند جمله‌ای برای آن شناخته نشده است *

* اگر یکی از Np_complete ها با یک الگوریتم چند جمله‌ای حل شود، تمام آن‌ها می‌توان به صورت چند جمله‌ای حل کرد *

Np = non deterministic polynomial



- حالات خاص مسئله Np_complete و Np_Hard خواهد شد

← Np_complete و Np_Hard ها معمولاً بر روی Decision problem ها یعنی مسئله‌ای که جواب آن‌ها No / yes خواهد شد، تعریف می‌شود *

← مثلاً دور همیلتونی یک Np_complete می‌باشد *

← پس برای مسئله optimization چه موقع Np.Hard تعریف می‌شود؟

← برای این مسئله می‌گوییم Np-Hard هست، زیرا اگر شود، یک Decision problem مرتبط با آن نیز به صورت چند جمله‌ای حل خواهد شد

مثلاً TSP، Np-Hard است زیرا اگر بتوانیم آن را در زمان چند جمله‌ای حل کنیم، دور همیلتونی نیز به صورت چند جمله‌ای حل خواهد شد به این ترتیب که وزن یال‌های موجود در گراف را 1 و وزن یال‌های موجود در گراف را $+\infty$ می‌گذاریم. اگر جواب بهینه TSP چند جمله‌ای n (تعداد یال‌های موجود) شد، یعنی دور همیلتونی داریم و اگر هم از n بزرگتر شد یعنی در گراف الگوریتم همیلتونی وجود ندارد *

Maral

Subject:
Yare: Month: Day: Page: ( )

* قضیه ۲۹-۱۵ . (۴۸)

← برای اثبات آن که نشان دهیم یک مسئله NP-complete و یا Np-Hard است باید:

۱- اول نشان دهیم آن مسئله NP است*

۲- ثانیاً نشان دهیم الگوریتم مسئله Np-complete نیز به صورت چند جمله‌ای حل می‌شود*

* وقتی می‌خواهیم یک مسئله Np-complete یا Np-hard است؛ آن‌گاه مجوز آن را پیدا می‌کنیم که برای حل آن مسئله از الگوریتم‌های چند جمله‌ای استفاده می‌کنیم*

* یک کلاس از دسته مسائل را Strongly_Np-Hard گوییم:
اگر با محدود کردن Integer های مسئله، باز هم Np-Hard باشد*

← مثلاً مسئله‌ی Tsp از نوع Strongly Np-Hard است؛ یعنی اگر تمام وزن‌های یالی گراف را نیز عقدد ۱ و ۲ در نظر بگیریم، باز هم برای آن الگوریتم Pdynomial وجود نخواهد داشت*

← ولی مسئله‌ی کوله‌پشتی از نوع Strongly Np-Hard نیست*

← اگر مسئله‌ای Np-Hard باشد؛ برای آن الگوریتم چند جمله‌ای وجود نخواهد داشت
و اگر مسئله‌ای Strongly Np-Hard باشد برای آن الگوریتم شبه چند جمله‌ای هم وجود نخواهد داشت
و اگر مسئله‌ای most Np-Hard باشد برای آن الگوریتم اف پی تاس نیز وجود نخواهد داشت*

Maral

* مسئله‌ی کوله پشتی:

- برای چنین مسئله‌ای الگوریتم شبه چندجمله‌ای وجود دارد*

- W : ظرفیت کوله | ⟸ مسئله‌ی Knapsack، عدد صحیح (Integer) از یکی
- n : تعداد آیتم‌ها (کالاها) | پیچیدگی زیر است:
- k : تعداد آیتم‌های هر کالا

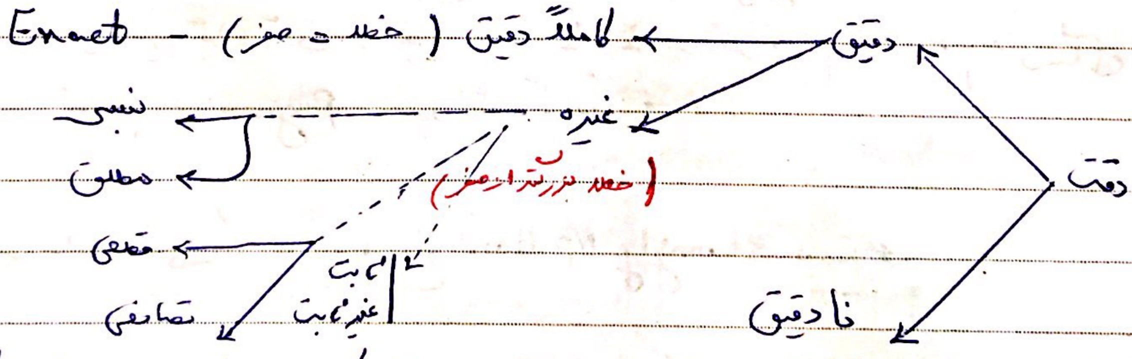
$O(nwk)$

- تمرین ۱۷-۷: اثبات NP-Hard بودن مسئله‌ی کوله پشتی.

C = کران بالا برای مقدار تابع هدف (حداکثر می‌تواند برابر با مجموع وزن‌های کالاها باشد)

* دقت الگوریتم‌ها:

- تا به اینجا فرض بر آن بود که به جواب حدوداً نزدیک می‌رسیم، ولی ممکن است کاملاً هم جواب دیگری دست پیدا نکنیم



- وقتی می‌گوییم یک الگوریتم دقیق است به این معنا نیست که خطای آن صفر است بلکه به معنی آن است که خطای نسبی و مطلق در چنین الگوریتم‌هایی کنترل شده است.
ولی در الگوریتم‌های غیر دقیق، با عدم کنترل مواجه خواهیم شد.

* در الگوریتم‌های Exact، خطا صفر است.

مثلاً الگوریتم

* خطای مطلق، نسبت به خطای نسبی، جواب تراست.

* خطای ۱۰٪ مقدار تابع هدف است، این نوع خطا نسبی می‌شود که به میزان تابع هدف نیز بستگی دارد.

$$opt(I) \leq A(I) \leq (1.1) \, opt(I)$$

→ تمامی اوقات می‌توانیم خطا ۲۰٪ است ← این نوع خطا، ثابت است.

ولی الگوریتم خطا $\left(\frac{1}{n}\right)$ است یعنی وابسته به تعداد متغیرهای تابع بستگی خواهد داشت.

* الگوریتم هایی که در پیاده‌سازی‌شان، از گروه‌های (عوامل) تصادفی دارند:

Randomize Algorithm نامیده می‌شوند *

→ وقتی دنبال الگوریتم خطای نسبی یا خطای مطلق مشخص داشته باشیم و بتوانیم آن را در زمان Polynomial حل کنیم به آن Approximation Algorithm گفته می‌شود *

(فصل ۱۴)

→ برای مسئله کوله‌پشتی: الگوریتم Absolute Approximation وجود ندارد.

* الگوریتم K-factor Approximation:

$A(I)$ = جواب بدست آمده (Instance I) با حل به وسیله‌ی الگوریتم

$OPT(I)$ = " " " بهینه

سازی Min ← مسئله: $\frac{1}{k} \, opt(I) \leq A(I) \leq k \cdot opt(I)$

$(k>1)$ سازی Max

* نمادهایی که در سایر کتاب ها استفاده می شود:

* Minimization:

$OPT(I) \leq A(I) \leq \varepsilon' OPT(I)$; $\varepsilon' > 1$ ($\varepsilon' = k$)

$OPT(I) \leq A(I) \leq (1+\varepsilon) OPT(I)$; $\varepsilon > 0$

* Maximization: خطای نسبی را نشان می دهد

$\varepsilon' OPT(I) \leq A(I) \leq OPT(I)$; $0 < \varepsilon' < 1$

$(1-\varepsilon) OPT(I) \leq A(I) \leq OPT(I)$; $0 < \varepsilon < 1$

← مثلا در این حالت می گوییم الگوریتم $\frac{3}{2}$ Approximation است یعنی

* $(1+\frac{1}{2})$ می باشد ← یعنی نهایتا 50% خطا خواهیم داشت

← پس برای مسئله ی minimization ای k Approximation باید $k > 1$ باشد و برای مسئله ی maximization ای k Approximation باید $k < 1$ باشد*

پس مثلا الگوریتم برای مسئله ی max سازی $\frac{1}{2}$ Approximation باشد یعنی حداکثر تا 1/2 نسبت به جواب بهینه خطا خواهیم داشت.

* برای مسئله ی TSP حالت کلی (و نه حالت متقارن) هیچ الگوریتم K-Approximation ای وجود نخواهد داشت *

* Double-Tree Algorithm از صفحه ۵۵۹ به بعد خوانده شود.

← این الگوریتم ابتدا یک درخت فراگیر در شبکه اولیه به دست می‌آورد و سپس یال‌های آن را Double می‌کند ← سپس براساس آن یک دور همیلتونی به دست می‌آورد و ثابت می‌کند دور همیلتونی حاصل نهایتاً ۲ فاکتور است * (یعنی نهایتاً ۱۰۰٪ جواب بهینه خطا خواهیم داشت)

* مسئله‌ی کوله‌پشتی

* این مسئله از جنس maximum‌سازی است.

- Fractional knapsack problem :

- یعنی کالاهایی که می‌توان انتخاب کرد، به جای آن که {0,1} باشند، می‌توانند بین صفر و یک نیز مقدار بگیرد.

- الگوریتم چند جمله‌ای برای کوله‌پشتی پیوسته:

- ابتدا نسبت $\left(\frac{c_i}{w_i}\right)$ را محاسبه می‌کنیم و کالاها را براساس آن مرتب می‌کنیم.

حال تا جایی که ظرفیت کوله اجازه می‌دهد کالاها را برمی‌داریم، در این حالت کالای آخر Fractional می‌شود که باید بخشی از آن برای کوله برداشته شود.

(مثلاً ۲۰٪ کالای آخر انتخاب می‌شود) ← $O(n\log n)$ ← زیرا پیچیده‌ترین محاسبات آن sort کردن است.

- {0,1} Knapsack problem :

- برای این مسئله نیز طبق (Proposition 17.4) یک الگوریتم 2-Approximation ارائه می‌گردد ← یعنی نهایتاً ۵۰٪ خطا دارد *

- این الگوریتم می‌گوید ابتدا همان مسئله‌ی پیوسته حل می‌شود و کالاها طبق $\left(\frac{c_i}{w_i}\right)$ مرتب گردند، سپس K امین آیتمی در نظر بگیرید که از W (ظرفیت کوله) بالاتر می‌زند.

$$c_1 + c_2 + \dots + \underset{\leq 1}{\odot}\, c_K \Rightarrow \begin{cases} \sum_{i=1}^{K-1} c_i & ; S_1 = \{1, 2, \dots, K-1\} \\ c_K & ; S_2 = \{K\} \end{cases}$$

← حال این دو مجموعه کالای {1,2,...,K-1} و {K}، کالاهای انتخاب می‌شود که تابع هدف بیشتری داشته باشد.

* اثبات اینکه الگوریتم فوق یک 2-Approximation است:

$$\Rightarrow A(I) = \max\left\{\sum_{i=1}^{k-1} c_i, c_k\right\} \geq \underbrace{\frac{\sum_{i=1}^{k} c_i}{2}}_{\text{میانگین}}$$

$$\geq \frac{OPT(Relax(I))}{2}$$

$$\geq \frac{OPT(I)}{2}$$

* ← به دنبال این هستیم که بتوانیم برای همه‌ی kها یک الگوریتم K-Approximation بدهیم.

* طرح های تقریبی:

- مجموعه‌ای از الگوریتم‌ها هستند که به آن‌ها "طرح های تقریبی" گوییم.

← اگر به ازای هر $\varepsilon > 0$، یک الگوریتم به صورت $O(n^{k(\varepsilon)\cdot f(\varepsilon)})$ باشد، آن‌گاه به آن یک PTAS (یتاس) گوییم. ← یعنی به ازای هر k، یک الگوریتم چندجمله‌ای داشته باشیم. ($f(\varepsilon)$ تابعی از چند جمله‌ای)

* ← FPTAS های همه‌ی مسائل هیچ تأثیری به روی توان چندجمله‌ای ندارد *

موضوع: تاریخ: / /

* اگر از یک مرجع ε الگوریتمی به صورت $O(n^k (\frac{1}{\varepsilon})^c)$ باشد؛
آن گاه ما یک FPTAS گوییم *

* الگوریتم FPTAS برای کوله پشتی {۰و۱} ؛ کتاب تست ؛ Page 464 :

* FPTAS برای knapsack {۰و۱} ؛ حتماً خوانده شود *

FPTAS = Fully Polynomial Time Approximation Sheme

① ابتدا هیوریستیک را حل می کنیم و S_1 را به دست می آوریم. (هیوریستیک های الگوریتم انتهای صفحه ۴۷ ...)

$\frac{1}{2}$ ← $OPT(I) = C(S')$ مسئلهٔ knapsack

② ابتدا $t = \max\{1, \frac{\varepsilon C(S_1)}{n}\}$ قرار بده.
- پس $\bar{c}_j = \lfloor \frac{c_j}{t} \rfloor$

③ - پس آن الگوریتم برنامه ریزی پویا را بر روی زوج ها اجرا کن (T: عملی کران بالای C را $\frac{2C(S_1)}{t}$ قرار بده. (S_1 از هیوریستیک گرفته شده است) و آن را S_2 نام گذاری کن * زیرا هیوریستیکی که حل می شود؛ 2-Approximation است.

④ حال از بین S_1 و S_2 آن که بهتر است را انتخاب کن *

← این الگوریتم از درجهٔ پیچیدگی $O(n^2 \cdot \frac{1}{\varepsilon})$ است:

* جلسه بیست و هشتم:

← در ادبیات بهینه سازی ترکیباتی وقتی از Approximation Algorithm ها صحبت می کنیم تا خود آگاه چند جمله ای هم در دل خودش دارد. ولی در ادبیات تحقیق در عملیات اصلاً چنین فرضی وجود نخواهد داشت و اتفاقاً به خطای مطلق نیز بسیار پرداخته می شود. مثل الگوریتم های Branch and Bound , Branch and cut و لاگرانژ و ... قادرند هم خطای نسبی و هم خطای مطلق را بدهند

تاریخ: / /

خطای نسبی ضریب $0.02 \, OPT(I)$ ← خطای نسبی

یا

خطای مطلق ← 0.02

0.01

خطاها ← $OPT(I) - A(I) \leq N$ کران بالا

مثلا ← $OPT(I) - A(I) \leq 0.02 \, OPT(I)$

$\Rightarrow (1 - 0.02) \, OPT(I) \leq A(I)$ *

ε

$\varepsilon' = \frac{1}{k}$

* مسئله‌ی کوله پشتی و الگوریتم ارائه شده برای آن:

$\frac{1}{2} OPT(I) \leq C(S_i) \leq OPT(I)$

$(1 - \frac{1}{2})$

(جواب حاصل از روش هیوریستیک صفحه‌ی ۴۷ جزوه)

میزان خطادار ۵۰٪ جواب بهینه شده است ← $OPT(I) - C(S_i) \leq \frac{1}{2} OPT(I)$

یا

$\frac{OPT(I) - C(S_i)}{OPT(I)} \leq \frac{1}{2}$

* در بدترین حالت $OPT(I)$ می‌تواند با $2C(S_i)$ جایگزین شود *

(PTAS) * برای هر ε دلخواه بتوان یک الگوریتم ارائه داد (طرح تقریبی):

* برای مسئله‌ی کوله پشتی الگوریتمی داریم که می‌تواند آن را با $O(nC)$ حل کند*

اگر knapsack را با هیوریستیک حل کرده، جواب به‌دست‌آمده را کران بالای ... ⇐ در بدترین حالت با $\sum c_i$ ها می‌تواند جایگزین شود ⇐ کران بالای مشخص (تابع هدف) خیلی نزدیک‌تر و دقیق‌تر

بالای C که $2C(S_i)$ می‌شود

* پس این الگوریتم به کمترین کران بالایی که برای تابع هدف به دست آید وابسته است و اگر این کران را ندانسته باشیم به جای آن $\sum c_i$ را می‌گذاریم. (الگوریتم شبه‌چندجمله‌ای پویا)

قبلی: به n و w (ظرفیت کوله) O(nW) وابسته بود)

* ساختار O برای الگوریتم هایی که به ازای هر ε، یک طرح تقریبی ارائه می دهند، (PTAS ها)

$$O(n^{f(\varepsilon)} \cdot g(\varepsilon))$$

یا

$$O(h(\varepsilon) \cdot n) \xrightarrow{\text{اگر } \varepsilon \text{ ثابت باشد}} O(n^k)$$

نسبت به ε نزولی است — نسبت به n صعودی است

* اگر برای مسئله‌ای PTAS وجود داشته باشد، فوق العاده سبب راحتی و خوشحالی است

FPTAS → $O(n^k \cdot (\frac{1}{\varepsilon})^L)$ →

- نسبت به ε نزولی
- نسبت به n صعودی
- هر دو چند جمله‌ای

* برای مسئله‌ی کوله پشتی صفر و یک:

۱- می توان در قالب IP نوشت — $O(nc)$

۲- می توان آن را به صورت با برنامه ریزی پویا حل کرد — $O(nW)$

۳- می توان برای آن الگوریتم 2-Approximation ارائه کرد *

۴- می توان FPTAS هم ارائه داد *

← برای مسئله‌ی Generalized Assignment prob:

$(1 - \frac{1}{e})$ = امید ریاضی ε است: ← بر اساس Randomized Algorithm بدست می آید.

(عدد نپر — ε)

- اگر الگوریتمی داشته باشیم که امید ریاضی ε و ε' آن مشخص باشد، با تکرار این الگوریتم و استفاده از قضیه‌ی مارکوف می‌توانیم به یک جوابی برسیم که با احتمال دلخواه از یک مقدار مشخص کمتر باشد.

$E(\varepsilon')$ ✓ ε ← خطای نسبی

$E(\varepsilon)$ ✓ $P\left(\frac{OPT(I)-A(I)}{OPT(I)} \geq a\right) \leq \frac{E(\varepsilon)}{a}$ ⇒ آنالیز خطای احتمالی

نامساوی مارکوف

← همچنین کافیست این الگوریتم Randomize را t بار Run بگیریم تا بتوانیم این کران بالا را بهبود دهیم *

سوال: t حداقل چقدر باشد تا با احتمال $(1-\beta)\%$ مطمئن باشیم خطا بزرگتر از ε به دست می‌آید؟؟

* FPTAS برای مسئله‌ی کوله‌پشتی:

① ابتدا الگوریتم 2-Approximation اجرا می‌کنیم، جواب حاصل را S_1 می‌نامیم.

- اگر $C(S_1)=0$ آنگاه $S=S_1$ و خاتمه ✓

(با فرض آنکه اعداد، اعدادی صحیح مثبت هستند، تابع هدف دیگر احتمال بهبود ندارد)

ورودی I:
$0 \leq c_1, c_2, \ldots, c_n \in N$
$C(S) = \sum_{i \in S} c_i$
$w_1, w_2, \ldots, w_n$

* زیرا وقتی $C(S_1)=0$ شود، چون می‌دانیم $OPT(I) \leq 2C(S)$، یعنی $OPT(I) \leq 0$ که دیگر احتمال بهبود جواب وجود ندارد.

تاریخ: / /

$O(nC)$

(۳) (۲) پس الگوریتم برنامه‌ریزی پویا برای یک نمونه ورودی $\bar{I}$ از کل (برای ساختن نمونه جدید؛ کافیست نمونه قبلی تقسیم بر t شود (آن‌ها را به سمت پایین رند می‌کند) دراین نمونه جدید $\bar{c}_i = \lfloor \frac{c_i}{t} \rfloor$ می‌باشد و t نیز از رابطه‌ی زیر به دست می‌آید:

$$t = \max\left\{1, \frac{\epsilon C(S_1)}{n}\right\} \xrightarrow{\text{به پایان خودمان}} t = \max\left\{1, \left(\frac{1}{\epsilon} - 1\right)\frac{C(S_1)}{n}\right\}$$

به پایان نامه

که دراین برنامه‌ریزی پویا نیز کران بالا با C را برابر با $\frac{2C(S_1)}{t}$ قرار می‌دهیم.

وجواب حاصل را S_2 می‌نامیم.

(۴) از بین S_1 و S_2 آن که بهتر است (یعنی هرکدام که $C(S)$ اش کمتر باشد) را انتخاب می‌کنیم و هم عنوان خروجی الگوریتم درنظر می‌گیریم ★

★ الان باید دو مورد را اثبات کنیم

۱- الگوریتم FPTAS است. ← یعنی به صورت $O(n^3 \times \frac{1}{\epsilon})$ است.

۲- جوابی که از آن به دست می‌آید؛ حتماً در نامساوی زیر صدق می‌کند:

$$(1-\epsilon)\,OPT(\bar{I}) \leq A(\bar{I}) \leq OPT(\bar{I})$$

★ قسمت اول $O(n\log n)$ محاسبات نیاز دارد و قسمت دوم (شامل تمام ③ و ②) $O(nC = n \cdot \frac{2C(S_1)}{t})$ و قسمت سوم نیز $O(1)$ محاسبات نیاز دارد.

چرا این عبارت، یک کران بالا برای C است؟

$$\Rightarrow A - 1 \leq \lfloor A \rfloor \leq A$$

$$\Rightarrow \bar{c}x_1 + \dots + \bar{c}x_n \leq \frac{c_1}{t}x_1 + \dots + \frac{c_n}{t}x_n \leq \frac{1}{t}\sum_{i=1}^{n} c_i x_i$$

$$\Rightarrow OPT(\bar{I}) \leq \frac{1}{t}OPT(\bar{I}) \leq \frac{1}{t}2C(S_1)$$

* اثبات سرعت الگوریتم:

(۲) $O(nc = n \cdot \frac{2c(S_1)}{t}) = O(2n \cdot \min\{c(S_1), \frac{\varepsilon'}{1-\varepsilon'} \cdot n\})$

$2n \cdot c(S_1) \cdot \frac{1}{t} = 2n \cdot c(S_1) \cdot \min\{1, \frac{\varepsilon'}{1-\varepsilon'} \cdot \frac{n}{c(S_1)}\} = 2n \cdot \min\{c(S_1), \frac{\varepsilon'}{1-\varepsilon'} \cdot n\}$

- حاصل $\min\{c(S_1), \frac{\varepsilon'}{1-\varepsilon'} \cdot n\}$ یا برابر $c(S_1)$ و یا $\frac{\varepsilon'}{1-\varepsilon'}$ می‌شود. زمانی $c(S_1)$ می‌شود که $t=1$ باشد. می‌دانیم زمانی $t=1$ است که:

$$(\frac{1}{\varepsilon'} - 1)\frac{c(S_1)}{n} \leq 1 \Rightarrow c(S_1) \leq n\frac{\varepsilon'}{1-\varepsilon'}$$

← پس در هر شرایطی قطعاً جواب $\min\{c(S_1), \frac{\varepsilon'}{1-\varepsilon'} \cdot n\}$ از $\frac{\varepsilon'}{1-\varepsilon'} \cdot n$ کوچکتر مساوی می‌شود یعنی:

$$O(nc = n\frac{2c(S_1)}{t}) = O(2n \cdot \min\{c(S_1), \frac{\varepsilon'}{1-\varepsilon'} n\}) \leq O(2n \cdot \frac{\varepsilon'}{1-\varepsilon'} \cdot n) = O(n^2 \cdot \frac{1}{\varepsilon})$$

($\frac{\varepsilon'}{1-\varepsilon'}$: ε)

* پس گزاره‌ی اول، یعنی چند جمله‌ای بودن را اثبات کردیم. الان می‌خواهیم نشان دهیم این الگوریتم تضمین از $(1-\varepsilon)$ کمتر نیست، یعنی در رابطه‌ی $(1-\varepsilon)OPT(I) \leq A(I) \leq OPT(I)$ صدق می‌کند.

یعنی در این الگوریتم $\longrightarrow$ $O(n \cdot \log n) + O(n) + O(n^2 \cdot \frac{1}{\varepsilon}) + O(1) = O(n^2 \cdot \frac{1}{\varepsilon})$

(زیر $O(n \log n)$: هیورستیک) (زیر $O(n)$: ساختن نمونه جدید) (زیر $O(n^2 \cdot \frac{1}{\varepsilon})$: برنامه‌ریزی پویا برای $\bar{I}$ با تو هم) (زیر $O(1)$: مقایسه‌ی S_1 و S_2) (بالای $O(1)$: یا در بدترین حالت $O(n)$)

$c = \frac{2c(S_1)}{T}$

بزرگترین عبارت انتخاب می‌شود

* آنالیز دقت الگوریتم:

$$(1-\varepsilon)\,OPT(I) \lesssim A(I) \lesssim OPT(\bar{I})$$

$$(1-\varepsilon)\,OPT(I) \lesssim \max\{C(S_1), C(S_2)\} \lesssim OPT(I)$$

می دانیم $\longrightarrow \frac{c_i}{t} - 1 \lesssim \bar{c}_i = \left[\frac{c_i}{t}\right] \lesssim \frac{c_i}{t} \equiv \begin{cases} c_i \gtrsim t\bar{c}_i & (1) \\ t\bar{c}_i \gtrsim c_i - t & (2) \end{cases}$

(3) * می دانیم S_2 ، جواب بهینه‌ی مسئله‌ی $\bar{I}$ است، لذا S^* که جواب بهینه‌ی مسئله‌ی I است را درون $\bar{I}$ بگذاریم شرایط بهتر نمی‌شود. (یعنی در مسئله‌ی max سمت راست جواب حاصل کمتر می‌گردد)

$$C(S_2) = \sum_{i \in S_2} c_i \gtrsim \sum_{i \in S_2} \bar{c}_i t = t\left(\sum_{i \in S_2} \bar{c}_i\right)$$

طبق (1)

$$\gtrsim t\left(\sum_{i \in S^*} \bar{c}_i\right) = \sum_{t \in S^*} t\bar{c}_i$$

طبق (3)

$$\gtrsim \sum_{i \in S^*} (c_i - t)$$

طبق (2)

$$\gtrsim \sum_{i \in S^*} c_i - nt = C(S^*) - nt$$

← نهایتاً اعضای S^* می‌توانند n تا باشد

(۴)

بنابراین

$$\Rightarrow C(S_2) \geq C(S^*) - nt$$

- حال اگر $t = 1$ یعنی I و $\bar{I}$ یکی هستند لذا $C(S_2) = C(S^*)$ خواهد شد. $S_f = S_2$

- اگر $t > 1$ باشد یعنی:

$$\left(\frac{1}{\varepsilon'} - 1\right)\frac{C(S_1)}{n} > 1 \xrightarrow[t \text{ جایگذاری}]{\text{جایگذاری}} C(S_2) \geq C(S^*) - n\left(\frac{1}{\varepsilon'} - 1\right)\frac{C(S_1)}{n}$$

t

(۴)

$$\Rightarrow C(S_2) - \left(\frac{1}{\varepsilon'} - 1\right)C(S_1) \geq C(S^*)$$

$$C(S_f) = \max\{C(S_1), C(S_2)\}$$

$$\Rightarrow \frac{1}{\varepsilon'} C(S_f) = C(S_f) + \left(\frac{1}{\varepsilon'} - 1\right)C(S_f) \geq C(S_2) + \left(\frac{1}{\varepsilon'} - 1\right)C(S_1) \geq C(S^*)$$

$$\Rightarrow \frac{1}{\varepsilon'} C(S_f) \geq C(S^*) \Rightarrow \underbrace{C(S_f)}_{A(I)} \geq \varepsilon' \underbrace{C(S^*)}_{OPT(I)}$$

$$\Rightarrow A(I) \geq (1 - \varepsilon)\, OPT(I)$$