استاد شهاب بهبهانی Date: ۹۱/۷/۳ No. ۱

طراحی الگوریتم‌ها

کتاب مرجع جهت تدریس:

* مقدمه‌ای بر الگوریتم‌ها Introduction to Algorithms

by: Carmen, Leiserson, Rivest & Stein (CLRS)

ویرایش سوم سال ۲۰۰۹

ترجمه: حمیدرضا قدسی

۲- ملزم

۳- (نویسنده مشخص نیست) با مقدمه‌ای از دکتر یحیی تابش (این کتاب را بخرید)

جلد اول را بخرید

سرفصل‌های تدریس

فصل ۱ الی ۴ — ۱۱ الی ۱۶ — ۲۲ الی ۲۵

مقدمه‌ای بر تحلیل الگوریتم‌ها

وجود مسئله — یک مدل محاسباتی — حل مسئله

چند نکته

۱- الزاماً تمام مسائل (مسائل محاسباتی) توسط یک مدل محاسباتی قابل حل نیستند.

۲- یک مسئله قابل حل معمولاً دارای چندین راه حل می‌باشد و ما به دنبال انتخاب بهترین راه حل می‌باشیم.

چندین راه حل — مقایسه راه حل‌ها — انتخاب بهترین راه حل

اگر فرض کنیم مدل محاسباتی ما یک کامپیوتر تک پردازنده‌ای باشد، ما راه حل‌ها را در قالب (فرم) یک الگوریتم ارائه کرده و الگوریتم مورد نظر را توسط یک زبان برنامه‌نویسی پیاده‌سازی کرده و روی کامپیوتر مورد نظر اجرا می‌کنیم.

الگوریتم

روشی است گام به گام برای حل یک مسئله که هر گام آن معین و فاقد هرگونه ابهام می‌باشد.

الگوریتم باید حتماً پایان داشته باشد.

الگوریتم: یک نقطه شروع = یک نقطه پایان

Date. No. ۲

ورودی ← الگوریتم ← خروجی

مثال

فرض کنیم دو الگوریتم A و B برای حل یک مسأله داریم

شخصی الگوریتم A را با زبان C پیاده سازی می کند

شخصی دیگر الگوریتم B را با زبان C++ پیاده سازی می کند

سؤال

کدام یک از این دو الگوریتم سریعتر است؟ نامشخص است

چون زمان اجرا در این حالت وابسته به زبان برنامه نویسی و ماشینی که در آن برنامه اجرا شود

پس می بایست به دنبال روشی باشیم که بتوان الگوریتم ها را به صورت مجرد و خارج از زبان برنامه نویسی و ماشین اجرا با یکدیگر مقایسه کرد.

راه حل استفاده از نمادهای مجانبی

نماد مجانبی

O (او بزرگ)، Ω (امگا بزرگ)، Θ (تتا)، o (او کوچک)، ω (امگا کوچک)

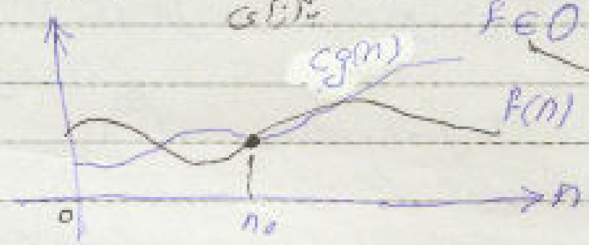
تعریف نماد O (او بزرگ) (بیگ او)

فرض کنید f و g توابع طبیعی باشند در این صورت

$O(g) = \{ f \mid \exists c, n_0 \;\; \forall n \geq n_0 \;\; 0 \leq f(n) \leq c\,g(n) \}$

(∃ : وجود دارد، ∀ : به ازای)

در این صورت می نویسیم $f \in O(g)$ (∈ : متعلق است به)



از نقطه n_0 به بعد همواره $cg(n)$ از $f(n)$ بزرگتر می باشد.

مثال

$f(n) = 3n^2 + 1$

$g(n) = n^2$

$c = 4 \Rightarrow cg(n) = 4n^2$

n	1	2	3
f(n)	4	13	28
cg(n)	4	16	36

(n_0 = 2)

با انتخاب $c = 4$ و $n_0 = 2$ همواره $\forall n \geq 2 \;\; 3n^2 + 1 \leq 4n^2$ و $3n^2 + 1 \in O(n^2)$

نکته

$f(n)=3n^2+5n+1000$

جمله موثر در رشد f

n^2 عامل موثر در رشد f

اگر $f(n)=a_k n^k+a_{k-1}n^{k-1}+\dots+a_1 n+a_0$ آنگاه $a_k\neq 0$

$f(n)=O(n^k)$ عامل موثر رشد

پس وقتی می گوییم $f\in O(g)$ یعنی رشد مجانبی f از بالا توسط g محدود می شود.

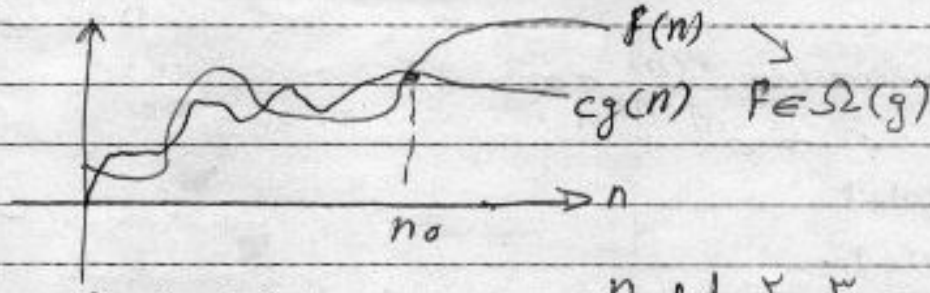
هر بالایی هر تابع توان بالایی معمود در جلوی باشد.

$f(n)=3n^2+1$

$f(n)=O(n^2)$ درست $f(n)\in O(n)$ غلط

تعریف نماد Ω (امگا) (کوچک)

$$\Omega(g)=\{f \mid \exists c,n_0 \ \forall n\geq n_0 \quad 0\leq cg(n)\leq f(n)\}$$



مثال

$f(n)=3n^2+1$

$g(n)=n^2 \quad c=2$

$\forall n\geq 1 \quad 3n^2+1\geq 3n^2$

$3n^2+1=\Omega(n^2)$

n	۱	۲	۳
$f(n)$	۴	۱۳	۲۸
$cg(n)$	۲	۸	۱۸

نماد Θ

$$\Theta(g)=\{f \mid \exists c_1,c_2,n_0 \ \forall n\geq n_0 \quad \underbrace{c_1 g(n)\leq f(n)\leq c_2 g(n)}_{f\in\Omega(g)} \ \underbrace{\Downarrow f\in O(g)}$$

$f\in\Theta(g)$

نکته:

$f\in\Omega(g), f\in O(g) \iff f\in\Theta(g)$

مثال

$3n^2+1=\Theta(n^2) \quad \dots \leq n^2\leq 3n^2+1\leq 4n^2$

سوال آیا مساوی برقرار است
بلی زیرا

$\frac{1}{1000}(n^2)+1 \in \Theta(n^2)$

$\frac{1}{10000} n^2 < \frac{1}{1000} n^2+1 < n^2$

در نماد مجانبی فرایب ثابتها را نادیده میگیریم و
بزرگترین جمله را در نظر میگیریم

$3n^2 \longrightarrow n^2$

تعریف نماد o (کوچک) (خیلی خیلی بزرگ)

$o(g) = \{f \mid \forall c \; \exists n_0 \; \forall n \geq n_0 \quad 0 \leq f(n) < cg(n)\}$

تعریف معادل

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

تعریف نماد w (امگای کوچک) (خیلی خیلی کوچک)

$w(g) = \{f \mid \forall c \; \exists n_0 \; \forall n \geq n_0 \quad 0 \leq cg(n) < f(n)\}$

تعریف معادل

$f \in w(g) = \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

مثال

$f(n) = 3n^2+4$

$g(n) = n+1000$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{3n^2+4}{n+1000} = \infty \Rightarrow f(n) = w(g)$

مثال

$f(n) = \log n$ (در کامپیوتر مبنای لگاریتم ۲ می باشد نه ۱۰)

$g(n) = n$ کوچک

$\log(n) \in o(n)$ ... برعکس لگاریتم n خیلی خیلی کوچکتر از n است

بلکه $\log(n) << (n)$

$f \in O(g) \Rightarrow f \leq g$

$f \in \Omega(g) \Rightarrow f \geq g$

$f \in \Theta(g) \Rightarrow f = g$

$f \in o(g) \Rightarrow f << g$ ← خیلی خیلی کوچک

$f \in w(g) \Rightarrow f >> g$

خیلی خیلی بزرگ

Date. No. 8

نکته:

رشد توابع نمایی >> رشد توابع چند جمله‌ای

مثال $1000n^{100} << 2^n$

نکته: ترتیب صعودی چند تابع به لحاظ مجانبی $\log n$ و $\log\log n$ و n^2 و 2^n و $n!$ و n^n

فاکتوریل

$n!$ فاکتوریل خیلی خیلی بزرگتر از 2^n می‌باشد. $2^n << n!$

سوال $\log n! = \Theta(?)$

$n! = n \times (n-1) \times (n-2) \times \dots \times \frac{n}{2} \times (\frac{n}{2}-1) \times (\frac{n}{2}-2) \times \dots \times 2 \times 1$

$\ll n \times n \times \dots \times 1 \times 1 \times \dots \Rightarrow \log n! \ll n \log n$

$\log n! = O(n \log n)$

$\log n! = \Omega(n \log n)$

$\log n! = \Theta(n \log n)$

به این ترتیب می‌توان نشان داد که

در نتیجه

قرارداد

$\log n = \log_2 n$

$\log^k n = (\log n)^k$

$\lg \lg n = \lg(\lg n)$

تحلیل الگوریتم‌ها

Time complexity ۱- از نظر زمان مصرف شده ← پیچیدگی زمانی

space complexity ۲- حافظه مصرف شده ← پیچیدگی فضا

انواع تحلیل پیچیدگی زمانی

۱- بهترین حالت

۲- میانگین یا متوسط

۳- بدترین حالت.

زمان اجرای الگوریتم وابسته به ۲ فاکتور می‌باشد

۱- اندازه ورودی به عنوان مثال زمان مرتب سازی یک آرایه n عنصری از یک آرایه n عنصری کمتر است.

۲- خود ورودی: در اندازه‌های یکسان از ورودی الگوریتم ممکن است زمان‌های متفاوتی صرف کند.

مثال

آرایه A داده شده (همراه با عنصر x) تمام اعداد صحیح است. هدف این است که تعیین کنیم $x \in A$ هست یا خیر؟

تابع جستجوی خطی

```
linearsearch(A, x)
1. n=(A)
2. for i=1 to n
3.   if A[i]==x
4.     return i
5. return 0
```

زمان اجرا وابسته به تعداد تکرار حلقه‌های for و if است.

حلقه ↓

$1 \leq \#for \leq n$

$1 \times o(1) = o(1) \leq T(n) \leq n \times o(1) = o(n)$

$o(1) \leq T(n) \leq o(n)$

این الگوریتم در بهترین حالت ورودی پیچیدگی زمانی از مرتبه $o(1)$ و در بدترین حالت ورودی پیچیدگی از مرتبه $o(n)$ می‌باشد.

پیچیدگی زمانی در حالت میانگین با استفاده از توزیع ورودی و شهودی احتمالات محاسبه می‌شود.

مثال

محاسبه n امین عنصر دنباله فیبوناچی

$F_0\ F_1\ F_2\ F_3 - - - - - - - -$

n داده شده و F_n مدنظر است.

$$F_i \begin{cases} i & \text{if } i=0 \text{ or } 1 \\ F_{i-2} + F_{i-1} & \text{if } i \geq 2 \end{cases}$$

```
Fib(n)
1. if n==0 or n==1
2.     return n
3. return Fib(n-2) + Fib(n-1)
```

سوال: $T(n) = ?$

$$T(n) = \begin{cases} o(1) & \text{if } n=0 \text{ or } n=1 \\ T(n-2) + T(n-1) & \text{if } n \geq 2 \end{cases}$$

رابطه بازگشتی

$T(n) = T(n-2) + T(n-1)$

$T(0) = 0$

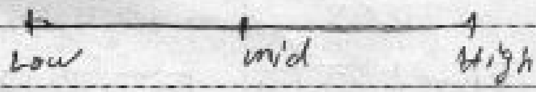
$T(1) = 1$

مثال جستجوی دودویی Binary search

آرایه مرتب شده (به صورت غیر نزولی) A همراه با عنصر x داده شده می خواهیم بدانیم $x \in A$ هست یا خیر

تابع جستجوی دودویی

Binary search (A, low, High, x)
1. if low > High return 0
2. mid = $\lfloor \frac{low + high}{2} \rfloor$
3. if x = A[mid]
4. return mid
5. else if x < A[mid]
6. Binary search (A, low, mid-1, x)
7. else Binary search (A, mid+1, High, x)



در بهترین حالت دارای پیچیدگی زمانی O(1) می باشد.

در بدترین حالت دارای پیچیدگی زمانی O(۳) می باشد $n=8 \quad \log n = 3$

حداکثر تعداد فراخوانی بازگشتی به ازای ورودی به اندازه n برابر $\lfloor \lg n \rfloor$ می باشد.

$T(n) = \lfloor \lg n \rfloor \times O(1) \Rightarrow T(n) = O(\lg n)$

روش های طراحی الگوریتم

۱- روش تقسیم و حل ۲- روش برنامه نویسی پویا ۳- روش حریصانه ۴- روش استقراء و ...

روش تقسیم و حل (Divide and conquer)

این استراتژی یک مسئله را به زیر مسائل با اندازه کوچکتر تقسیم می شود تا به زیر مسائلی برسیم که اندازه آنها به قدر کافی کوچک است و به راحتی قابل حل هستند مثلاً الگوریتم Binary search نمونه ای از روش تقسیم و حل می باشد.

هر الگوریتم تقسیم و حل شامل ۳ مرحله است

۱- تقسیم ← تقسیم مسئله به زیر مسائل

۲- حل ← حل زیر مسائل با اندازه کافی کوچک

۳- ترکیب ← ترکیب جواب های به دست آمده برای رسیدن به جواب مسئله اصلی

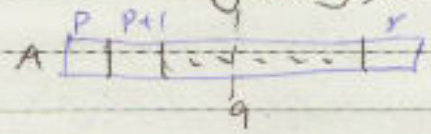
در بعضی از الگوریتم های تقسیم و حل در مرحله ترکیب کاری انجام نمی شود.

مثال: مرتب سازی ادغامی (Merge_sort)

در این الگوریتم مرحله حل وجود ندارد.

زمان اجرای هر الگوریتم تقسیم و حل = زمان تقسیم + زمان حل + زمان ترکیب

الگوریتم مرتب سازی ادغامی




```
Mergesort(A, P, r)
1 if P < r
تقسیم { 2 q = [(P+r)/2]
        3 Merge_sort(A, P, q)
        4 Merge_sort(A, q+1, r)
ترکیب 5 Merge(A, P, q, r)
```

اندازه آرایه A

فرض می‌کنیم $n = r - P + 1$ باشد آنگاه

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O(n \lg n)$$

زمان ادغام — یک رابطه بازگشتی

رابطه بازگشتی

رابطه‌ای است که در دو طرف علامت مساوی هم عبارت صورت تکرار شونده و در مقابل رابطه صحیح و باشد.

مثال: $T(n)=2n^2+1$ رابطه صحیح

$T(n)=2T(n/2)+4$ رابطه بازگشتی

قضیه اصلی

در بیشتر الگوریتم‌های تقسیم و حل برای محاسبه زمان اجرا یک رابطه بازگشتی به فرم کلی زیر می‌رسیم.

$$T(n)=aT\left(\frac{n}{b}\right)+f(n)$$

مسئله اصلی را به a تا قسمت که هر قسمت دارای اندازه $\frac{n}{b}$ هست تقسیم کرده و برای حل برای ترکیب آنها نیاز مند زمان $f(n)$ هستیم.

حالات قضیه

فرض کنید $a\geq 1$ و $b>1$ مقادیر ثابت و $f(n)$ هم یک تابع باشد در این صورت اگر $T(n)$ به صورت

$$T(n)=aT\left(\frac{n}{b}\right)+f(n)$$

روی مقادیر n تعریف شده باشد $(n\in Z^+)$ که $\frac{n}{b}$ می‌تواند به یکی از صورتهای $\left(\frac{n}{b}\right)$ یا $\left[\frac{n}{b}\right]$ باشد آنگاه داریم

۱- اگر برازای یک $\varepsilon>0$ (اپسیلون) $f(n)=O(n^{\log_a b-\varepsilon})$ باشد آنگاه $T(n)=\Theta(n^{\log_b a})$

۲- اگر $f(n)=\Theta(n^{\log_b a})$ آنگاه $T(n)=\Theta(n^{\log_b a}\cdot\log n)$

۳- اگر برازای یک $\varepsilon>0$ و شرط تظم $f(n)=\Omega(n^{\log_b a-\varepsilon})$ برای همه‌ی n های به قدر کافی بزرگ $af\left(\frac{n}{b}\right)\leq cf(n)$ باشد که $c<1$ مقدار ثابت است آنگاه $T(n)=\Theta(f(n))$

حالات قضیه اصلی:
- حالت اول: رشد تابع $n^{\log_b a}$ از $f(n)$ بیشتر است.
- حالت دوم: رشد تابع $n^{\log_b a}$ با $f(n)$ برابر است.
- حالت سوم: رشد تابع $f(n)$ از $n^{\log_b a}$ بیشتر است (به صورت چند جمله‌ای)

شرط تظم

برای ثابت $c<1$ و همه‌ی nهای بزرگ $af\left(\frac{n}{b}\right)\leq cf(n)$

مثال

$T(n) = 9T(\frac{n}{3}) + n$

$a=9 \quad b=3 \quad f(n)=n$

$n^{\log_b^a} = n^{\log_3^9} = n^2 \Rightarrow n^2 > f(n) = n$

$T(n) = \theta(n^{\log_3^9}) = \theta(n^2)$ حالت اول قضیه اصلی برقرار است

مثال

$T(n) = T(\frac{2n}{3}) + 1000$

$T(n) = T(\frac{n}{\frac{3}{2}}) + 1000 \Rightarrow a=1 \quad b=\frac{3}{2} \quad f(n) = \underbrace{1000}_{\theta(1)}$

$n^{\log_b^a} = n^{\log_{\frac{3}{2}}^1} = n^0 = \underbrace{1}_{\theta(1)}$

حالت دوم قضیه اصلی برقرار است

$T(n) = \theta(n^{\log_{\frac{3}{2}}^1} \times \log) = \theta(\log n)$

مثال

$T(n) = 3T(\frac{n}{4}) + \log n \Rightarrow a=3 \quad b=4 \quad f(n) = \log n$

$n^{\log_b^a} = n^{\log_4^3} = n^{1-\varepsilon}$ (اپسیلون) , $f(n) = n\log n > n^{1-\varepsilon}$ (اکیداً بیشتر)

$\frac{f(n)}{n^{1-\varepsilon}} = \frac{n\log n}{n^{1-\varepsilon}} = n^{\varepsilon}\log n$ شرط نظم برقرار است

$3(\frac{n}{4})\log(\frac{n}{4}) = \frac{3}{4} n\log\frac{n}{4} \leqslant \underbrace{\frac{3}{4}}_{c} \underbrace{n\log n}_{f(n)}$

$T(n) = \theta(n\log n)$ حالت سوم قضیه برقرار است

مثال

$T(n) = 2T(\frac{n}{2}) + n\log n \Rightarrow a=2 \quad b=2 \quad f(n) = n\log n$

$n^{\log_b^a} = n^{\log_2^2} = n < f(n) = n\log n$

ممکن است حالت سوم قضیه برقرار شود

$\frac{f(n)}{n^{\log_b^a}} = \frac{n\log n}{n} = \log n$

$f(n)$ به صورت چند جمله‌ای بزرگتر از n نیست در نتیجه از حالت سوم قضیه اصلی نمی‌توان استفاده کرد و باید این رابطه بازگشتی را با روش‌های دیگر حل کنیم.

سوال

$T(n)=۲T(\frac{n}{۲}+۱۷)+n$

برای n های بزرگ می توانیم از عدد ثابت صرف نظر کنیم.

$\frac{n}{۲}+۱۷=\frac{n}{۲}$ ⟵

$T'(n)=۲T'(\frac{n}{۲})+n$

$T(n)=T'(n)=\theta(n\log n)$

سوال

$T(n)=T(\sqrt{n})+۱$

$T(n)=T(n^{\frac{۱}{۲}})+۱$

فرض می کنیم n توانی از ۲ باشد مثلاً $n=۲^m$ در این صورت

$n^{\frac{۱}{۲}}=(۲^m)^{\frac{۱}{۲}}=۲^{\frac{m}{۲}}$

$T(۲^m)=T(۲^{\frac{m}{۲}})+۱$ قرار دهیم $T(۲^m)=S(m)$

$S(m)=S(\frac{m}{۲})+۱ \rightarrow a=۱ \quad b=۲ \quad f(n)=۱$

$n^{\log_b^a}=m^{\log_۲^۱}=۱$ ✓

حالت دوم تقسیم برقرار است.

$S(m)=\theta(m^{\log_۲^۱}.\log n) \rightarrow S(m)=\theta(\log m)$

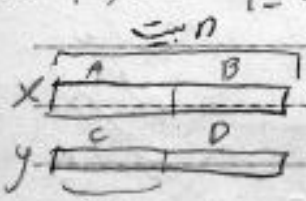
$T(۲^m)=S(m)=\theta(\log m)$

$T(n)=\theta(\log\log n)$

ادامه مبحث روش تقسیم و حل

مسئله ضرب اعداد n بیتی بزرگ

اعداد n بیتی x و y در مبنای دودویی داده شده است می خواهیم حاصلضرب آنها را به کمک روش تقسیم و حل بدست آوریم



$\frac{n}{۲}$

$x.y=Ac۲^n+(AD+Bc)۲^{\frac{n}{۲}}+BD$

$T(n)=? \rightarrow T(n)=۴T(\frac{n}{۲})+\theta(n) \rightarrow T(n)=$ اگر $f(n)=۰$ آنگاه $T(n)=۱$؛ اگر $f(n)\neq۰$ آنگاه $۴T(\frac{n}{۲})+\theta(n)$

$a=۴ \quad b=۲ \quad f(n)=\theta(n)$

$T(n) = \theta(n^2)$

بهبودی نسبت به حالت ۱ و ۲ و بهبودی حاصل نشد

$\theta(n^2)$

برای ایجاد بهبود کافی است تعداد ۴ ضرب $n/2 \times n/2$ را به ۳ تا ضرب $n/2 \times n/2$ کاهش دهیم

$xy = AC2^n + [(A-B)(D-C) + AC + BD] 2^{n/2} + BD$

$(A-B)(D-C) = AD - AC - BD + BC$

پس کافی است AC و $(A-B)(D-C)$ و BD محاسبه شود.

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 3T(n/2) + \theta(n) & \text{if } n>1 \end{cases}$$

$T(n) = \theta(n^{\log_2 3}) = \theta(n^{1.59})$

مسأله یافتن توام ماکزیمم و مینیمم مقادیر آرایه

آرایه $A[1..n]$ از اعداد داده شده است

هدف محاسبه $\max(A)$ و $\min(A)$ است.

روال زیر به تنهایی عنصر ماکزیمم آرایه ها را محاسبه می کند

```
max(A)
1) n = |A|
2) max = A(1)
3) for i = 2 to n
4)   if A(i) > max
5)     max = A(i)
6) return max
```

تعداد مقایسه ها در $\max(A)$ برابر $n-1$ می باشد.

روالی مشابه $\max(A)$ تحت عنوان $\min(A)$ عنصر مینیمم A با $n-1$ مقایسه پیدا کنیم

$= (n-1) + (n-1) = 2n-2$ مجموع مقایسه ها

– می خواهیم با استفاده از روش تقسیم و حل الگوریتمی ارائه کنیم که تعداد مقایسه ها را بهبود بخشد

آرایه A به اندازه n را به دو قسمت مساوی تقسیم می کنیم

$A = A_1 \cup A_2$

$(min_1, max_1) \leftarrow min, max(A_1)$

$(min_2, max_2 \leftarrow min, max(A_2)$

سپس داریم:

$(min \leftarrow min(min_1, min_2)$

$(max \leftarrow max(max_1, max_2)$

و با مقایسه‌گیری برای تعداد مقایسه‌های انجام شده توسط روال min_max(A) به صورت زیر داریم:

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ 1 & \text{if } n=2 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 & \text{if } n>2 \end{cases}$$

با فرض اینکه n توانی از ۲ است نتیجه می‌شود:

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

$T(1) = 0$

$T(2) = 1$

$T(n) = 2(2T(\frac{n}{4}) + 2) + 2 =$

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

$T(n) = 2^2(2T(\frac{n}{2^3}) + 2) + 2^2 + 2$

$T(n) = 2^3 T(\frac{n}{2^3}) + 2^3 + 2^2 + 2$

$S = \frac{a - a^{n+1}}{1-a}$

$T(n) = 2^{i-1} T(\frac{n}{2^{i-1}}) + (2^{i-1} + 2^{i-2} + \dots + 2^2 + 2^1)$

$\frac{n}{2^i} = 1 \rightarrow 2^i = n \rightarrow i = \log n$

$= 2^{i-1} T(2) + (2^{i-1} + 2^{i-2} + \dots + 2) = \frac{n}{2} + \frac{2 - 2^{(i-1)+1}}{1-2}$

$= \frac{n}{2} + \frac{2 - 2^i}{-1} = \frac{n}{2} + n - 2 = \frac{3}{2}n - 2$

$T(n) = \frac{3}{2}n - 2$ تعداد مقایسه کمتر شد

نکته

با کمی دقت می‌توان ثابت کرد که $\frac{3}{2}n - 2$ حد پایین تعداد مقایسه‌های لازم برای یافتن ماکزیمم و مینیمم یک آرایه n عضوی است و الگوریتم فوق الگوریتمی بهینه است.

الگوریتم استراسن برای ضرب ماتریس‌های nxn

فرض کنید A و B ماتریس‌های nxn باشند.
هدف محاسبه C=AB است.
الگوریتم معمولی ضرب ماتریس‌ها به صورت زیر است

```
Multiply(A,B)
1- n=rows[A]
2- let C be an nxn multiply
3- for i=1 to n
4-     for j=1 to n
5-         cij ← 0
6-         for k=1 to n
7-             cij=cij+aik·bkj
8- return C
```

زمان اجرای این الگوریتم از مرتبه $\Theta(n^3)$ است.
استراسن با استفاده از روش تقسیم و حل یک الگوریتم با زمان بهتر از $\Theta(n^3)$ برای ضرب ماتریس‌ها ارائه کرده است. روش استراسن به صورت زیر می‌باشد:
اگر $n=1$ باشد آنگاه A و B دو عدد هستند که ضرب آن‌ها برای محاسبه می‌شود.
برای $n>1$ که n توانی از ۲ است به صورت زیر عمل می‌کنیم.
هریک از ماتریس‌ها را به ۴ قسمت مساوی تقسیم می‌کنیم

$$A=\begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad B=\begin{pmatrix} e & f \\ g & h \end{pmatrix} \quad C=\begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

اگر A و B با شرط nxn آنگاه هریک از ماتریس‌های a,b,c,d,e,f,g,h ماتریس‌های $\frac{n}{2}\times\frac{n}{2}$ هستند.
در این صورت

$$C=AB$$

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix}=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

$$\begin{cases} r=ae+bg \\ s=af+bh \\ t=ce+dg \\ u=cf+dh \end{cases} \Longrightarrow T(n)=8T\left(\frac{n}{2}\right)+\Theta(n^2)$$

برای جمع ماتریس‌ها

$\Rightarrow T(n) = \theta(n^{\log_2^8}) = \theta(n^3)$

با استراسن تعداد ضرب‌های ماتریسی $\frac{n}{2} \times \frac{n}{2}$ را از ۸ به ۷ کاهش داده و در نتیجه زمان اجرا به صورت زیر درآمد.

$T(n) = \theta(n^{\log_2^7}) = \theta(n^{2.81})$

روش استراسن برای انجام این روش به صورت زیر است.

گام ۱

ابتدا حاصل $P_1 \dots P_7$ به صورت زیر

$P_5 \quad P_1 = (a+d)(e+h)$

$P_2 \quad P_7 = (c+d)e$

$P_1 \quad P_3 = a(f-h)$

$P_4 \quad P_6 = d(g-e)$

$P_1 \quad P_5 = (a+b)h$

$P_7 \quad P_4 = (a-c)(e+f)$

$P_4 \quad P_7 = (b-d)(g+h)$

محاسبه هر P_i که $1 \le i \le 7$ معادل ضرب دو ماتریس $\frac{n}{2} \times \frac{n}{2}$ است.

گام ۲

محاسبه مقادیر ماتریس C به صورت زیر

$r = P_1 + P_6 - P_5 + P_7$

$s = P_3 + P_5$

$t = P_2 + P_4$

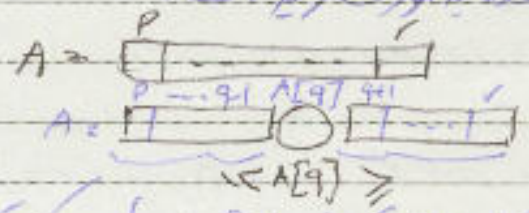
$u = P_1 + P_3 - P_2 + P_4$

الگوریتم مرتب سازی سریع Quick sort

آرایه $A[p \dots r]$ از اعداد داده شده است.

هدف، مرتب سازی عناصر این آرایه به صورت غیر نزولی (صعودی)

یک روش تقسیم و حل برای این مسئله به صورت زیر است.



آرایه $A[p \dots r]$ به صورت به شکل بالا تبدیل می‌شود به طوری که همه عناصر آرایه $A[p \dots q-1]$ کوچکتر یا مساوی $A[q]$ است و همه عناصر آرایه $A[q+1 \dots r]$ بزرگتر یا مساوی $A[q]$ است.

و سپس به صورت بازگشتی هر یک از این دو زیر آرایه مرتب می‌شوند.

```
Quick sort (A, p, r)
1  if p < r
2     q = PARTITION (A, p, r)
3     Quick sort (A, p, q-1)
4     Quick sort (A, q+1, r)
```

روال partition به صورت زیر است.

```
PARTITION (A, p, r)
1  x = A[r]
2  i = p-1
3  for j = p to r-1
4     if A[j] ≤ x
5        i = i+1                    عوض کن
6        exchange A[i] ↔ A[j]
7  exchange A[i+1] ↔ r
8  return i+1
```

اگر $n = r-p+1$ باشد زمان اجرا از مرتبه $\Theta(n)$ می‌باشد.

i

۱	۲	۳	۴	۵	۶
۷	۱۰	۳	۴	۹	۲

a_j a_i

مثال

a_i $i=۶$ $x=۷$

به روشی مرتب سازی سریع می شود به شکل زیر

۲	۳	۴	۷	۹	۱۰

برنامه نویسی پویا (Dynamic programming)

برای بسیاری از مسائل استفاده از یک تقسیم و حل منجر به حل تکراری تعدادی از زیر مسائل می شود. برای جلوگیری از این مسئله روشی برنامه نویسی پویا پیشنهاد می شود. برخلاف روشی تقسیم و حل که مسئله را از بالا به پایین حل می کند روش DP مسائل را از پایین به بالا حل می کند یعنی زیر مسائل حل و جواب آنها را برای استفاده زیر مسائل بزرگتر در یک جدول ذخیره می کند.

سوال

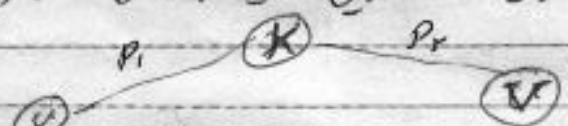
آیا تمامی مسائل را می توان با روشی برنامه نویسی پویا حل کرد؟

خیر، تنها آن دسته از مسائل بهینه سازی که دارای خاصیت زیر ساختار بهینه هستند با DP قابل حل می باشند.

خاصیت زیر ساختار بهینه این است که حل بهینه برای یک مسئله شامل حل بهینه برای زیر مسائل آن باشد.

مثال

دو گراف $G=(V,E)$ می خواهیم طول کوتاهترین مسیر از رأس u به رأس v را محاسبه کنیم



P کوتاهترین مسیر از u به V

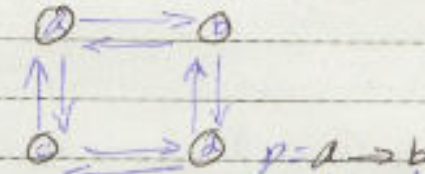
$|P| = |P_1| + |P_2|$

مسلماً P_1 کوتاهترین مسیر از u به K و P_2 کوتاهترین مسیر از K به V است زیرا اگر P_1 کوتاهترین مسیر از u به K نباشد می توان کوتاهترین مسیر دیگری مثل P_1' از u به K پیدا کرد که این با فرض اینکه مسیر P از u به V کوتاهترین مسیر است در تناقض است چون مسیر $u \xrightarrow{P_1'} K \xrightarrow{P_2} V$ مسیر کوتاهتری نسبت به P است و این تناقض است.

پس مسئله کوتاهترین مسیر در گراف دارای خاصیت زیر ساخت بهینه است

مثال

آیا مثال یافتن طولانی‌ترین مسیر بین دو راس یک گراف دارای خاصیت زیر ساختار بهینه است یا خیر؟ خیر



طولانی‌ترین مسیر از a به c

طولانی‌ترین مسیر از b به d

a→c→d→b

از b به d می‌شود

پس در حالت کلی مثال فوق دارای زیر ساختار بهینه نمی‌باشد.

مراحل برنامه نویسی پویا

۱- تعیین ساختار جواب بهینه

۲- تعریف بازگشتی حل جواب بهینه (از روی جواب بهینه)

۳- محاسبه جواب بهینه

۴- ساخت جواب بهینه (شکل جواب)

مثال

مثال ضرب زنجیره‌ای ماتریس‌ها

ورودی n ماتریس $A_1 \dots A_n$ با ابعاد آنها

خروجی محاسبه یک پرانتزبندی برای محاسبه $A_1 \times A_2 \times \dots A_n$ به طوری که تعداد ضرب‌های نقطه‌ای مینیم باشد

مثال

ماتریس	ابعاد
A_1	۳×۵
A_2	۵×۲
A_3	۲×۴

$A_1 \times A_2 \times A_3$

$(A_1 \times A_2) A_3$ — $A_1 (A_2 \times A_3)$

۳×۲ — ۲×۴ — ۴×۵ — ۵×۴

$(۳ \times ۵ \times ۲) + (۳ \times ۲ \times ۴)$ — $(۵ \times ۲ \times ۴) + (۳ \times ۵ \times ۴)$

۵۴ — ۴۰ + ۶۰ — ۱۰۰

بهینه‌تر

این مثال دارای خاصیت زیر ساختار بهینه می‌باشد.

Date. No. 19

سوال

تعداد پرانتز بندی های ممکن برای $A_1 \dots A_n$ چند تا می باشد

اگر $P(n)$ تعداد پرانتز بندی ها برای $A_1 \dots A_n$ باشد در این صورت

$$P(n) \begin{cases} 1 & \text{if } n=1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$$

$P(n)$ ↓ دنباله اعداد کاتالان

چک کن

$$P(3) = \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1)$$

یک بار $k=1$ — یک بار $k=2$

$P(1)=1$ و $P(2)=2$ می دانیم $\longrightarrow 1\times 2 + 2\times 1 = 4$

نکته

$$P(n) = \Omega(2^n)$$

یک روش محاسبه هزینه تمامی پرانتز بندی ها و انتخاب کوچکترین آنها اما با توجه به نکته بالا این الگوریتم یک الگوریتم از مرتبه نمایی است.

روش DP برای حل این مسئله به روش زیر است.

گام اول: تعیین ساختار جواب بهینه

تعریف می کنیم

$$A_{i\dots j} = A_i \times A_{i+1} \times \dots A_j$$

فرض کنیم جواب بهینه برای $A_{i\dots j}$ حداقل است در این صورت یک نقطه شکستی مانند k وجود دارد به طوری که پرانتز بندی بهینه برای $A_{i\dots j}$ به صورت زیر است. k نقطه شکست است.

$$A_i \times \dots A_j = (A_i \times \dots A_k)(A_{k+1} \times \dots A_j)$$

نقاط شکست را می دیم برای k عبارتند از $k=i, \dots, j-1$

گام دوم

تعریف بازگشتی حل جواب بهینه

فرض کنید $m[i,j]$ برابر هزینه جواب های بهینه برای محاسبه $A_i \dots A_j$ باشد اگر k نقطه شکست باشد آنگاه

$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} \times P_k \times P_j$

کمترین ابعاد ماتریس حاصل‌ضرب A_i تا A_j به صورت $P_{i-1} \times P_j$ است.

ماتریس	ابعاد
A_i	$P_{i-1} \times P_i$
A_{i+1}	$P_i \times P_{i+1}$
⋮	⋮
A_j	$P_{j-1} \times P_j$

P بردار ابعاد ماتریس‌هاست.

رابطه بازگشتی برای محاسبه $m[i,j]$ به صورت زیر است:

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \le k \le j-1} \{ m[i,k] + m[k+1,j] + P_{i-1} P_k P_j \} & \text{if } j>i \end{cases}$$

گام ۳

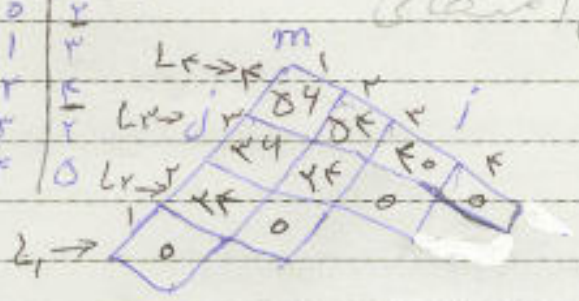
محاسبه جواب بهینه به صورت بازگشتی محاسبه می‌شود.

مثال

سوال امتحانی

ماتریس	ابعاد
A_1	۲×۲
A_2	۲×۴
A_3	۴×۱
A_4	۱×۵

P_i	$P[i]$
۰	۲
۱	۲
۲	۴
۳	۱
۴	۵



مرحله ۲

$P_0\ P_1\ P_2$

$m[1,2] = m[1,1] + m[2,2] + 2 \times 2 \times 4 = 0 + 0 + 16 = 16$

$m[2,3] = m[2,2] + m[3,3] + 2 \times 4 \times 1 = 0 + 0 + 8 = 8$

$m[3,4] = m[3,3] + m[4,4] + 4 \times 1 \times 5 = 0 + 0 + 20$

مرحله ۳

$m[1,3] = \min \begin{cases} k=1: m[1,1] + m[2,3] + 2 \times 2 \times 4 = 0 + 8 + 16 = 24 \\ k=2: m[1,2] + m[3,3] + 2 \times 4 \times 1 = 16 + 0 + 8 = 24 \end{cases}$

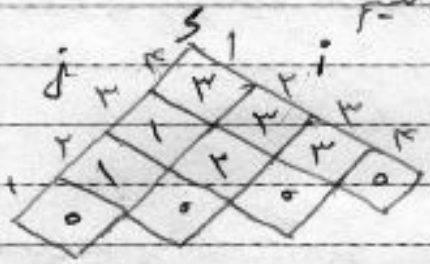
$m[2,4] = \min \begin{cases} k=2: m[2,2] + m[3,4] + 2 \times 4 \times 5 = 0 + 20 + 40 = 60 \\ k=3: m[2,3] + m[4,4] + 2 \times 1 \times 5 = 8 + 0 + 10 = 18 \end{cases}$

$m[1,4] = \min \begin{cases} k=1: m[1,1] + m[2,4] + 2 \times 2 \times 5 = 0 + 18 + 20 = 38 \\ k=2: m[1,2] + m[3,4] + 2 \times 4 \times 5 = 16 + 20 + 40 = 76 \\ k=3: m[1,3] + m[4,4] + 2 \times 1 \times 5 = 24 + 0 + 10 = 34 \end{cases}$

۲۴

$m[1,4] = \min \begin{cases} k=1: m[1,1] + m[2,4] + 2\times3\times5 = 0 + 54 + 30 = 84 \\ k=2: m[1,2] + m[3,4] + 2\times4\times5 = 24 + 30 + 40 = 104 \\ k=3: m[1,3] + m[4,4] + 2\times3\times5 = 36 + 0 + 30 = 66 \end{cases}$

نقاط شکست را در ماتریس نگهداری می‌کنیم



۴ و ۳

ساختن جواب بهینه از روی جدول S

$((A_1)\times(A_2\times A_3))\times A_4$

پرانتز بندی بهینه شده است.

مسئله یافتن طولانی‌ترین زیر دنباله مشترک

دو دنباله داده شده با نام X و Y را به صورت زیر در نظر بگیرید:

$X = X = \langle x_1, x_2, \dots, x_n \rangle$

$Y = \langle y_1, y_2, \dots, y_n \rangle$

هدف محاسبه طولانی‌ترین زیر دنباله مشترک دو دنباله داده شده X و Y است.

زیر دنباله

اگر $Z = \langle z_1, z_2, \dots, z_k \rangle$ یک زیر دنباله از دنباله $X = \langle x_1, \dots, x_k \rangle$ است، هرگاه دنباله $\langle i_1, i_2, \dots, i_k \rangle$ از اندیس‌های صعودی وجود داشته باشد به طوری که برای هر $j = 1, 2, \dots, k$ داشته باشیم $x_{ij} = z_j$

مثال

فرض کنیم

$X = \langle A, B, C, B, D, A, B \rangle$

چند زیر دنباله از X عبارتند از

$Z = \langle A, C, A, B \rangle$

$P = \langle A, B, C, B, D \rangle$

$M = \langle A, D \rangle$

مثال

$X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

$\langle B, C, A \rangle$ زیردنباله مشترک X و Y است اما طولانی‌ترین زیردنباله مشترک نیست. طولانی‌ترین زیردنباله مشترک X و Y عبارتند از

$\langle B, C, A, B \rangle$

$\langle B, D, A, B \rangle$

طولانی‌ترین زیردنباله مشترک X و Y را با $LCS(X, Y)$ نشان می‌دهیم

مسئله محاسبه LCS دو دنباله داده شده زیرساختار بهینه دارد پس می‌توان آن را با روش DP حل کرد

روش DP برای حل مسئله LCS

مرحله اول: تعریف ساختار جواب بهینه

فرض کنید $X = \langle x_1, x_2, \ldots, x_n \rangle$ باشد در این صورت منظور از X_i به صورت زیر تعریف می‌شود. i امین پیشوند X است.

$X_i = \langle x_1, x_2, \ldots, x_i \rangle$

قضیه (ساختار بهینه برای LCS)

فرض کنید $X = \langle x_1, \ldots, x_m \rangle$ و $Y = \langle y_1, \ldots, y_n \rangle$ دو دنباله باشند و $Z = \langle z_1, \ldots, z_k \rangle$ یک $LCS(X, Y)$ باشد در این صورت

۱- اگر $x_m = y_n$ آنگاه $z_k = x_m = y_n$ و $Z_{k-1} \in LCS(X_{m-1}, Y_{n-1})$

۲- اگر $x_m \neq y_n$ آنگاه اگر $z_k \neq x_m$ در این صورت $Z \in LCS(X_{m-1}, Y)$

۳- اگر $x_m \neq y_n$ آنگاه $z_k \neq y_n$ و $Z \in LCS(X, Y_{n-1})$

$X = \langle x_1, x_2, \ldots, x_{m-1}, x_m \rangle$

$Y = \langle y_1, y_2, \ldots, y_{n-1}, y_n \rangle$

$Z = \langle z_1, z_2, \ldots, z_{k-1}, z_k \rangle$

مرحله ۲: تعریف بازگشتی حل جواب بهینه

فرض کنید $c[i, j]$ طول $LCS(X_i, Y_j)$ باشد در این صورت

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ 1+c[i-1,j-1] & \text{if } i,j>0,\ x_i=y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{if } i,j>0,\ x_i \neq y_j \end{cases}$$

مرحله ۳: محاسبه جواب بهینه (محاسبه طول LCS)

```
LCS_lenght(x,y)
1  m=|x|
2  n=|y|
3  let b[1...m,1...n], c[1...m,1...n] be two tables
4  for i=1 to m
5      c[i,0]=0
6  for j=1 to n
7      c[0,j]=0
8  for i=1 to m
9      for j=1 to n
10     if x_i=y_j
11         c[i,j]=c[i-1,j-1]+1
12         b[i,j]='↖'   → یک واحد از اول کم کن
13     elseif c[i-1,j] ≥ c[i,j-1]
14         c[i,j]=c[i-1,j]
15         b=[i,j]='↑'   → یک واحد از اول کم کن
16     elseif c[i,j-1]>c[i-1,j]
17         c[i,j]=c[i,j-1]
18         b=[i,j]='←'   یک واحد از کم کن
19 return c and b
```

مرحله ۴: ساخت جواب بهینه

```
print-LCS(x,y,i,j)
1  if i=0 or j=0
2      return
```

```
if b[i,j] = '↖'
    print_LCS(x,y,i-1,j-1)
    print x
elseif b[i,j] = '↑'
    print_LCS(x,y,i-1,j)
else
    print_LCS(x,y,i,j-1)
```

یعنی از بالا آمده

مثال

محاسبه LCS(x,y) در صورتی که x=<A,B,C,B,D,A,B> و y=<B,D,C,A,B,A>

x_i \ y_j		۱ B	۲ D	۳ C	۴ A	۵ B	۶ A
	۰	۰	۰	۰	۰	۰	۰
۱ A	۰	۰↑	۰↑	۰↑	۱↖	۱←	۱↖
۲ B	۰	۱↖	۱←	۱←	۱↑	۲↖	۲←
۳ C	۰	۱↑	۱↑	۲↖	۲←	۲↑	۲↑
۴ B	۰	۱↖	۱↑	۲↑	۲↑	۳↖	۳←
۵ D	۰	۱↑	۲↖	۲↑	۲↑	۳↑	۳↑
۶ A	۰	۱↑	۲↑	۲↑	۳↖	۳↑	۴↖
۷ B	۰	۱↖	۲↑	۲↑	۳↑	۴↖	۴↑

```
print_LCS(x,y,۷,۶)      (i, j)
print_LCS(x,y,۶,۶)
print_LCS(x,y,۵,۵)
print A
print_LCS(x,y,۴,۵)
print_LCS(x,y,۳,۴)
print B
print_LCS(x,y,۳,۳)
print_LCS(x,y,۲,۲  → print C
```

print_LCS(x, y, i-1, j)
print_LCS(x, y, i-1, j-1)
print B
return LCS(x, y)

BCBA

زمان اجرا $\theta(mn)$ می باشد

OBST درخت جستجو دودویی بهینه

فرض کنید داریم $K = \langle k_1, \dots, k_n \rangle$ از n کلید مجزا داده شده است (به صورت مرتب) برای هر کلید k_i

p_i = احتمال جستجوی کلید k_i

q_i = احتمال جستجوی کلید مابین k_i و k_{i+1}

امکان دارد مقادیری که مورد جستجو واقع می شود در K نباشد در این صورت n+1 کلید فرضی اضافی کنیم به نامهای $d_0, d_1, d_2, \dots, d_n, d_{n+1}$ که واقع d_i نشان دهنده تمام مقادیر بین k_i و k_{i+1} است.

پس q_i احتمال جستجوی متناظر با d_i است.

بنابر قانون احتمال داریم $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$

زمان مورد انتظار برای جستجوی در درخت دودویی T که براساس مقادیر k_i ها و d_i ها شکل گرفته به صورت زیر محاسبه می شود.

$$E(\text{search cost in } T) = \sum_{i=1}^{n} (\underbrace{depth_T(k_i)}_{\text{عمق در ریشه}} + 1) p_i + \sum_{i=0}^{n} (depth_T(d_i) + 1) \times q_i = 1 + \sum_{i=1}^{n} (depth_T(k_i) \times p_i) + \sum_{i=0}^{n} (depth_T(d_i) \times q_i)$$

امید ریاضی جستجو

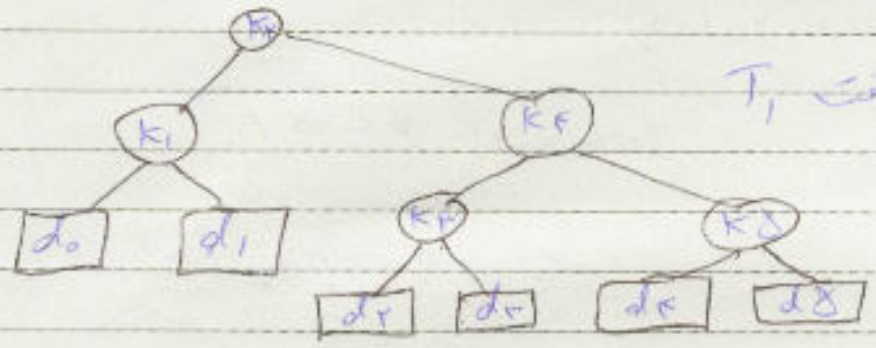
مثال

فرض کنید

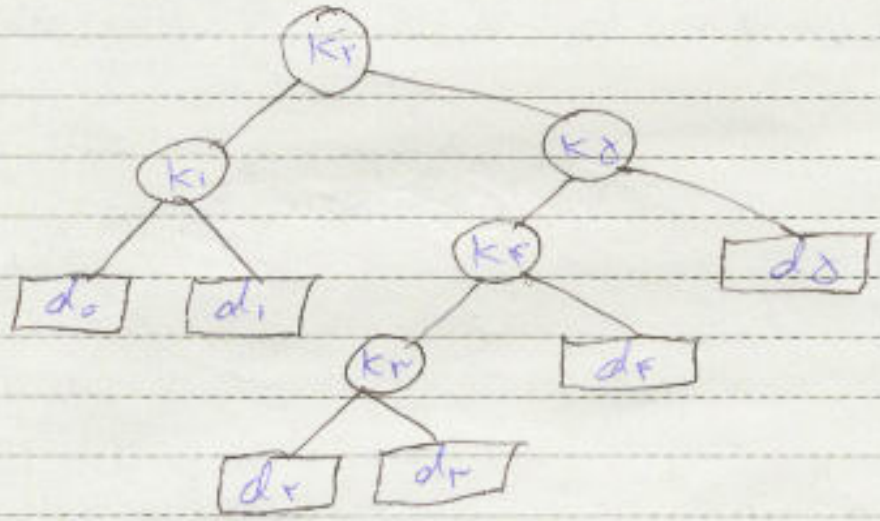
i	0	1	2	3	4	5
p_i	-	۰.۱۵	۰.۱۰	۰.۰۵	۰.۱۰	۰.۲۰
q_i	۰.۰۵	۰.۱۰	۰.۰۵	۰.۰۵	۰.۰۵	۰.۱۰

۲ درخت T_1 و T_2 را به صورت زیر در نظر بگیرید

درخت T_1



درخت T_2



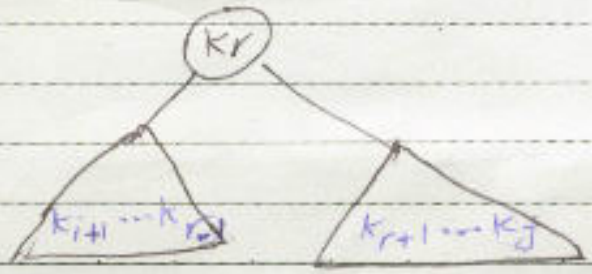
$E(T_1) = ۲.۸۰$ → درخت T_2 نسبت به درخت T_1 بهتر است.

$E(T_2) = ۲.۷۵$

راه حل DP برای این مسئله

گام اول: تعیین ساختار جواب بهینه

فرض کنید می خواهیم درخت جست‌وجوی دودویی بهینه را برای کلیدهای k_i و ... و k_j همراه با کلیدهای فرضی d_{i-1} و ... و d_j محاسبه کنیم فرض می کنیم ریشه چنین درختی k_r است که $i \le r \le j$ این درخت به این صورت



گام دوم

فرض کنیم $e[i,j]$ هزینه جستجوی مورد انتظار در درخت جستجوی دودویی بهینه برای کلیدهای $k_i, \ldots, k_j$ باشد.

در این صورت

$$e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r+1,j))$$
$$= e[i,r-1] + e[r+1,j] + w(i,j)$$

که در آن

$$w(i,j) = \sum_{l=1}^{j} p_i + \sum_{l=i-1}^{j} q_l$$

گام سوم

$$e(i,j) = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min\{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j \\ i \le r \le j & \end{cases}$$

گام چهارم

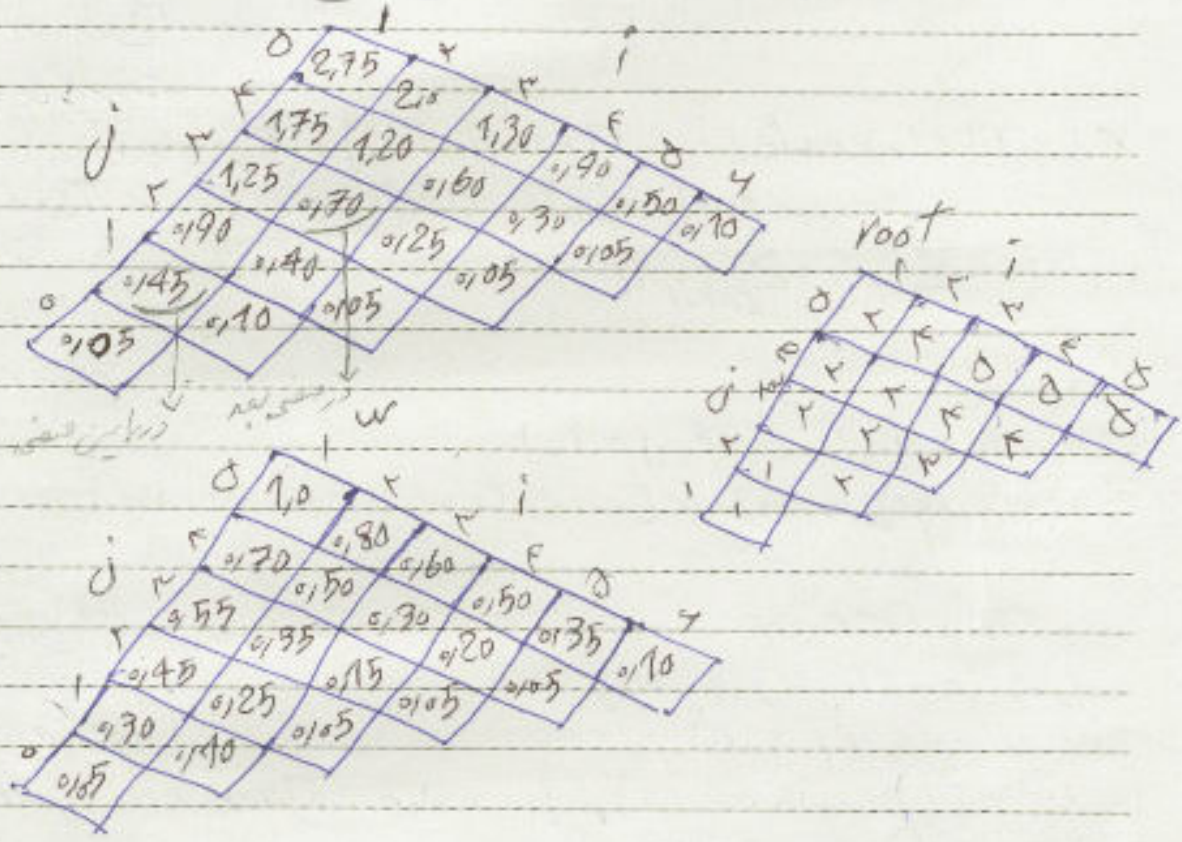
تشریفی کنیم

$w[i,j] = w[i,j-1] + p_j + q_j$

optimal-BST (p, q, n)

1 let $e[i \ldots n+1, 0 \ldots n]$, $w[1 \ldots n+1, 0 \ldots n]$ and $root[1 \ldots n]$
2 $1 \ldots n]$ be new tables.
3 for $i = 1$ to $n+1$
4 $e[i, i-1] = q_{i-1}$
5 $w[i, i-1] = q_{i-1}$
6 for $i = 1$ to n
7 for $i = 1$ to $n - l + 1$
8 $j = i + l - 1$
9 $e[i,j] = \infty$
10 $w[i,j] = w[i,j-1] + p_j + q_j$
11 for $r = i$ to j
12 $t = e[i,r-1] + e[r+1,j] + w[i,j]$
13 if $t < e[i,j]$
14 $e[i,j] = t$
15 $root[i,j] = r$ ← ریشه زیر درخت کلیدهای $k_i, \ldots, k_j$
return e, root ←

با حل این الگوریتم روی کلیه حالات قبل داریم



i	0	1	2	3	4	5
p_i	—	0/15	0/10	0/05	0/10	0/20
q_i	0/05	0/10	0/05	0/05	0/05	0/10

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] + w[i,j] \} & \text{if } i \le j \end{cases}$$

$$w[i,j] = \sum_{l=1}^{j} p_l + \sum_{l=i-1}^{j} q_l = (p_i + p_{i+1} + \cdots + p_j) + (q_{i-1} + q_i + \cdots + q_j)$$

$$e[1,1] = e[1,0] + e[2,1] + w[1,1] =$$

$$= 0/05 + 0/10 + 0/30 = 0/45$$

به روش دیگر

$$E(T)=\sum_{i=1}^{l}(depth(k_i)+1)\times p_i+\sum_{i=0}^{l}(depth(d_i)+1)\times q_i$$

$$=1\times p_1+2\times q_0+2\times q_1$$
$$=1\times 0.15+2(0.05)+2(0.10)$$
$$=0.15+0.10+0.20=(0.45)$$

$$e[2,2]=\min\begin{cases} e(2,1)+e(3,3)+w(2,3)= \\ 0.10+0.25+0.35=\boxed{0.70} \\ e(2,2)+e(4,3)+w(2,3)= \\ 0.40+0.05+0.25=0.80 \end{cases}$$

الگوریتم های حریصانه (greedy)

در روشی برنامه نویسی پویا گفتیم حل از پایین به بالا بود. این روش مشابه روشی برنامه نویسی پویا می باشد با این تفاوت که در هر مرحله با انجام یک انتخاب حریصانه اندازه یکی از زیر مسائل صفر می شود. به عبارتی نیازی به حل آن نداریم.

با یک مسئله فرض طراحی الگوریتم حریصانه برای آن پیچیده با این تکنیک برای الگوریتم آشنا می شویم.

مسئله انتخاب فعالیت ها (Activity selection)

فرض کنید یک مجموعه شامل n فعالیت به صورت زیر است.

$$S=\{a_1,a_2,\dots,a_n\}$$

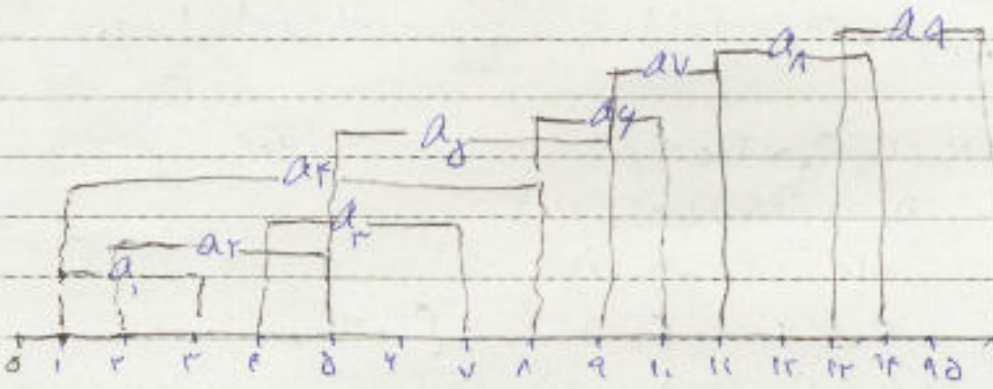
هر فعالیت مانند a_i در بازه زمانی $[s_i, f_i)$ انجام می شود.

(s_i: زمان شروع، f_i: زمان خاتمه)

هدف انتخاب بزرگترین زیر مجموعه ممکن از مجموعه فعالیت های S است به طوری که هر دو فعالیت آن هم پوشانی نداشته باشند.

فرض داریم که فعالیت ها براساس زمان خاتمه مرتب هستند.

	a_1	a_2							
i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



این مثال دو جواب دارد

$\{a_1, a_4, a_8, a_9\}$ $\{a_2, a_5, a_7, a_9\}$

ابتدا یک الگوریتم برنامه نویسی پویا برای حل آن ارائه و سپس یک روش حریصانه ارائه می کنیم

روش برنامه نویسی پویا

گام اول: تعیین زیرساختار بهینه

تعریف می کنیم $S_{ij} = \{a_k \in S \mid f_i \le s_k \le f_k \le s_j\}$

یعنی مجموعه فعالیت هایی که بین از فعالیت a_i شروع و قبل از شروع فعالیت a_j خاتمه می یابد

دو فعالیت مجازی زیر را تعریف می کنیم

$a_0 = [-\infty, 0)$

$a_{n+1} = [\infty, \infty+1)$

در این صورت $S = S_{0,n+1}$

از طرفی دیگر داریم $f_0 \le f_1 \le f_2 \le \dots \le f_n \le f_{n+1}$

اگر $i \ge j$ باشد داریم $S_{ij} = \emptyset$

برای حل S_{ij} فرض می کنیم جواب شامل فعالیت a_k است در این صورت ما با دو زیر مسئله مواجه هستیم

S_{ik} $[s_k, f_k]$ S_{kj}

S_{ik} مجموعه فعالیت هایی که بین از a_i شروع و قبل از آغاز a_k خاتمه می یابد

S_{kj} مجموعه فعالیت هایی که بین از a_k شروع و قبل از آغاز a_j خاتمه می یابد

در نتیجه جواب S_{ij} برابر است با

(جواب S_{ik}) $\cup \{a_k\} \cup$ (جواب S_{kj})

Subject: ۳
Year: Month: Date:

$|S_{ij}| = |S_{ik}| + 1 + |S_{kj}|$

فرض کنیم A_{ij} جواب بهینه S_{ij} باشد در این صورت

$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$

گام دوم

تعریف بازگشتی حل جواب بهینه

فرض کنید $c[i,j]$ اندازه زیر مجموعه ماکزیمم از فعالیت‌های دو به دو سازگار از S_{ij} باشد

در این صورت داریم

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{i<k<j,\ a_k \in S_{ij}} \{c[i,k] + 1 + c[k,j]\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

روش حریصانه

قضیه: اگر $S_{ij} \neq \emptyset$ و a_m فعالیتی در S_{ij} با زودترین زمان خاتمه باشد آنگاه

۱- a_m متعلق به یک زیر مجموعه ماکزیمم از فعالیت‌های دو به دو سازگار از S_{ij} است

۲- $S_{im} = \emptyset$ بنابراین با انتخاب حریصانه a_m نیاز به حل زیر مسئله S_{im} نیست

گام حل بازگشتی جواب بهینه

```
Recursive_Activity_selector(s, f, i, n)
1. m = i + 1
2. while m ≤ n and s_m < f_i
3.     m = m + 1
4. if m ≤ n
5.     return {a_m} ∪ Recursive_Activity_Selector(s, f, m, n)
6. else return ∅
```

آرایه زمان‌های شروع — آرایه زمان‌های خاتمه

یافتن فعالیت با زودترین زمان خاتمه

a_i (s_i, f_i)

حل تکراری زمان بهینه

```
Greedy_Activity_selector(s, f, n)
1. A ← {a_1}
2. i = 1
3. for m = 2 to n
```

AVA

4: if $s_m \geq f_i$
5: $A = A \cup \{a_m\}$
6: $i = m$
7: return A

زمان اجرا برابر با $\Theta(n)$

کدهای هافمن (Huffman codes)

از کدهای هافمن در فشرده‌سازی اطلاعات استفاده می‌شود.

انواع کد

۱- کد با طول ثابت: هر کاراکتر متن با یکی یکسان به ازای طول جایگزین می‌شود.

۲- کد با طول متغیر: هر کاراکتر ممکن است با یکی جایگزین شود که طول آن نسبت به طول کاراکتر دیگر متفاوت باشد.

مثال:

فرض کنید متنی شامل ۱۰۰۰۰۰ کاراکتر a, b, c, d, e, f است که تعداد تکرارهای آن برحسب ۱۰۰۰ به همراه ۲ کد پیشنهادی یکی با طول ثابت و دیگری با طول متغیر مطابق جدول زیر داده شده است.

کاراکتر	a	b	c	d	e	f
تعداد برحسب هزار	۴۵	۱۳	۱۲	۱۶	۹	۵
کد با طول ثابت	۰۰۰	۰۰۱	۰۱۰	۰۱۱	۱۰۰	۱۰۱
کد با طول متغیر	۰	۱۰۱	۱۰۰	۱۱۱	۱۱۰۱	۱۱۰۰

اندازه فایل متنی مورد نظر ۱۰۰۰۰۰ کاراکتر است

در روش کدگذاری با طول ثابت

حجم فایل پس از کدگذاری $100000 \times 3 \overset{\text{بیت}}{=} 300000$ bits

در روش کدگذاری با طول متغیر

حجم فایل پس از کدگذاری $(45\times1 + 13\times3 + 12\times3 + 16\times3 + 9\times4 + 5\times4)\times1000$
$= 224000$ bits

ایده: در کدگذاری با طول متغیر بهتر است کاراکتر با بیشترین تکرار دارای کد با طول کوچکتر باشد.

نکته

صفحه کد گذاری مهم نیست بلکه کدگشایی نیز مهم است لذا نباید هیچ کدی

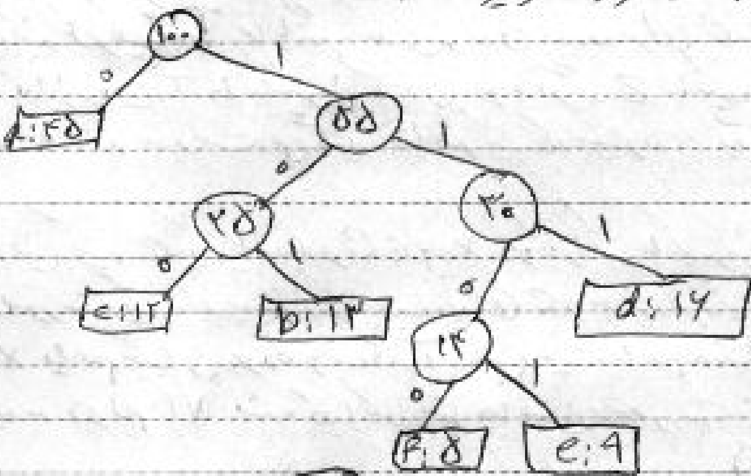
پیشوند کد دیگر باشد به عبارتی دیگر کدها باید دارای خاصیت کد پیشوندی باشند

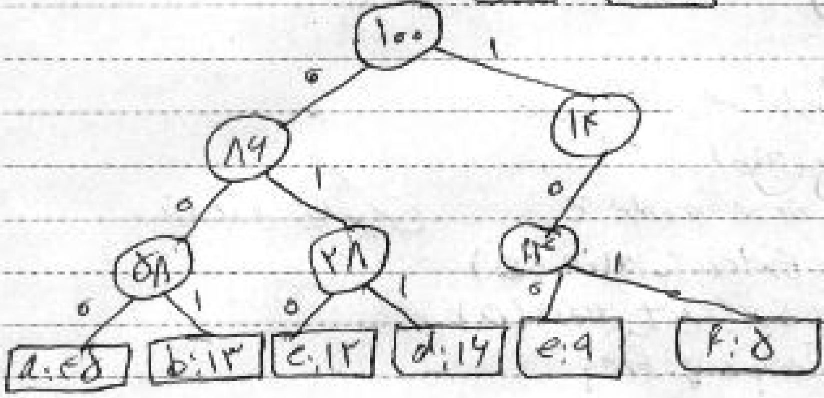
a = 001
b = 0 → 00001 → bbbbc / bba
c = 1

کد مربوط به b جزء ۰ مربوط به پیشوند ۰۰۱ که کد مربوط به a است

هر کدی پیشوندی را می‌توان با یک درخت پیشوندی نمایش داد. به عنوان مثال درخت‌های متناظر با دو کدگذاری بالا به صورت زیر هستند.

۴۴۳





کد بهینه:
کدی است که بیشترین حجم را پس از فشرده‌سازی فایل ورودی به دست می‌آید.
فرض کنید T یک درخت کد پیشوندی برای متنی شامل مجموعه کاراکترهای C باشد. در این صورت
اگر تعداد تکرارهای کاراکتر $c \in C$ را با c.freq نشان دهیم آنگاه تعداد بیت‌های لازم برای کد کردن فایل‌های ورودی برابر است با

$$B(T) = \sum_{c \in C} c.freq \times d_T(c)$$

ارتفاع نود برگ c در درخت کد T

AVA

کد هافمن یک کد بهینه می باشد (کد بهینه پیشوندی می باشد)
کد هافمن مبتنی بر دو لم زیر است.

۱- فرض کنید C یک الفبا باشد که برای هر $c \in C$ تعداد تکرارهای آن با $c.freq$ مشخص شود. اگر $x, y \in C$ دو کاراکتر با حداقل تعداد تکرار در میان سایر کاراکترهای C باشند و C' همان C باشد که در آن x و y حذف شده و z به جای آن با تعداد تکرار $z.freq = x.freq + y.freq$ اضافه شده است به عبارتی $C' = C - \{x, y\} \cup \{z\}$

فرض کنید درخت T نمایانگر کد پیشوندی بهینه برای C باشد در این صورت درخت T که از روی T با جایگزین کردن نود برگ z با یک نود داخلی که فرزندانش x و y هستند بدست می آید نمایانگر یک کد پیشوندی بهینه برای C است.

۲- فرض کنید C یک الفبا و $x, y \in C$ دو کاراکتر با کمترین تعداد تکرار در C باشند در این صورت یک کد پیشوندی بهینه برای C وجود دارد به طوری که کلمه های کد برای x و y با هم یکسان بوده و تنها در آخرین بیت با هم متفاوت هستند.
با توجه به دو لم بالا مشخص که الگوریتم هافمن به صورت زیر است.

```
HUFFMAN(C)
1. n = |C|
2. Q = C                    صف اولویت مینیمم
3. for i=1 to n-1
4. allocate a new node z    نود جدید ایجاد کن
5. z.left = x = Extract_Min(Q)
6. z.Right = y = Extract_Min(Q)
7. z.freq = x.freq + y.freq
8. Insert(Q, z)
9. return Extract_Min(Q)    برگرد ریشه درخت که
```

Q: صف اولویت مینیمم براساس مقادیر تکرار کاراکترها است.

مثال

کاراکتر	a	b	c	d	e	f
تکرار	۴۵	۱۳	۱۲	۱۶	۹	۵

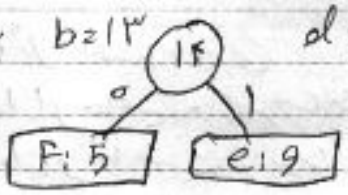
$n = |C| = 6$ $\quad n-1 = 6-1 = 5$ تعداد تکرارهای الگوریتم هافمن

AVA

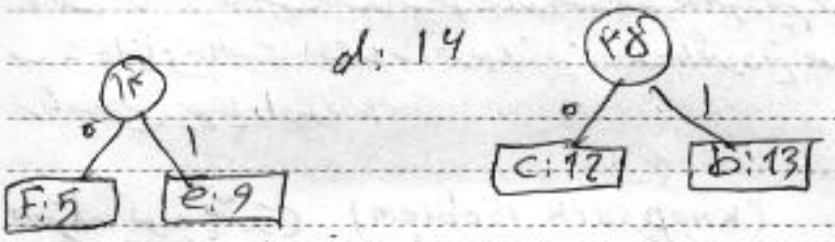
Subject: ۳۵
Year: Month: Date:

f:5 e:9 d:14 c:12 b:13 a:45

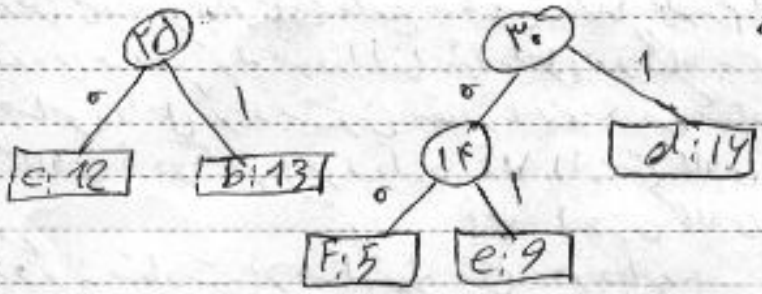
c:12 b:13 d:14 a:45



تکرار ۱ ص ۴۵

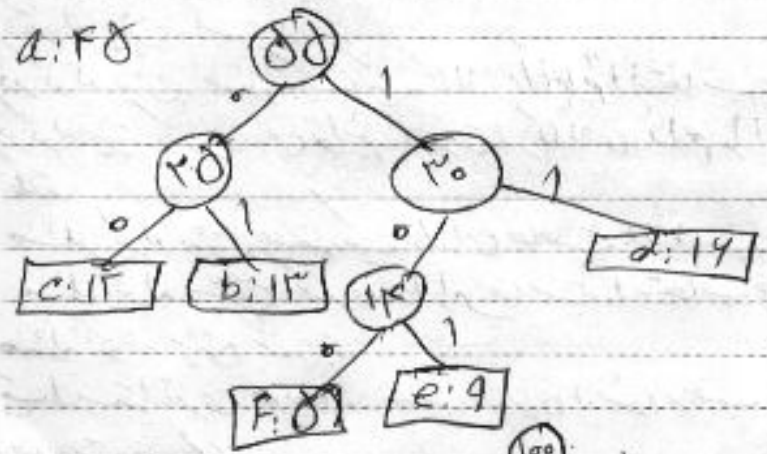


تکرار ۲

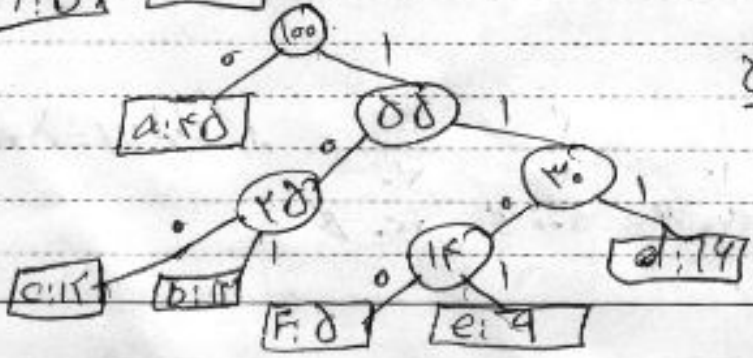


تکرار ۳

تکرار ۴



تکرار ۵



AVA

کاراکتر	a	b	c	d	e	f
کد	۰	۱۰۱	۱۰۰	۱۱۱	۱۱۰۱	۱۱۰۰

مسائل بهینه سازی
۱- دارای خاصیت زیرساختار بهینه نیستند.
۲- دارای خاصیت زیرساختار بهینه هستند به دو نوع اند.
الف: دارای خاصیت انتخاب حریصانه هستند ← الگوریتم حریصانه دارند.
ب: دارای خاصیت انتخاب حریصانه نیستند ← الگوریتم حریصانه ندارند اما الگوریتم برنامه‌نویسی پویا دارند.

ص ۲۳۹

مسئله کوله پشتی (knapsark problem) کنپ‌سک

فرض کنید یک کوله پشتی داریم که ظرفیت W کیلوگرم را دارد. دزدی وارد فروشگاه شده می‌خواهد کوله خود را از اشیاء داخل فروشگاه به نحوی پر کند که مجموع ارزش اجناس داخل کوله بیشترین مقدار باشد. در فروشگاه n تا کالا داریم. متناسب با کالای i دو آیتم وجود دارد v_i (ارزش کالای i ام)
w_i (وزن کالای i ام)

دو گونه متفاوت از مسئله کوله پشتی وجود دارد.
کوله پشتی صفر و یک
کوله پشتی کسری
در کوله پشتی صفر و یک کالای i ام یا انتخاب می‌شود یا نمی‌شود (کل کالا)
در کوله پشتی کسری می‌توانیم کسری از کالای i ام را انتخاب کنیم

نکته
مسئله کوله پشتی صفر و یک دارای خاصیت انتخاب حریصانه نیست
مسئله کوله پشتی کسری دارای خاصیت انتخاب حریصانه است.

مسئله کوله پشتی کسری
یک استراتژی حریصانه: مرتب سازی اشیاء برحسب نسبت ارزش به وزن آنها $\frac{v_i}{w_i}$ به صورت نزولی

مثال

شیء	۱	۲	۳
وزن (kg)	۱۰	۲۰	۳۰
ارزش	۶۰ $	۱۰۰ $	۱۲۰ $

ظرفیت کوله $W = 50$ kg

$\frac{60}{10} = 6$

$\frac{100}{20} = 5$

$\frac{120}{30} = 4$

انتخاب اول ← شیء شماره ۱
ظرفیت باقیمانده $W - w_1 = 50 - 10 = 40$
انتخاب دوم ← شیء شماره ۲
ظرفیت باقیمانده $W - w_2 = 40 - 20 = 20$
انتخاب سوم ← شیء شماره ۳ (انتخاب $\frac{2}{3}$ از شیء شماره ۳)
ظرفیت باقیمانده $W - w_3 = 20 - \frac{2}{3} \times 30 = 20 - 20 = 0$
مجموع ارزش کوله پشتی
وزن = ۵۰ کیلوگرم
ارزش = $60 + 100 + \frac{2}{3}(120) = 240$

الگوریتم کوله پشتی

```
Fractional_knapsark (W, w, v)
1.  n = |v|                  بردار ارزش اشیاء   بردار وزن اشیاء
2.  let x, b be two new array
3.  for i=1 to n
4.      xi = 0
5.      bi = vi/wi
6.  load = 0                 مقدار بار مساوی صفر است
7.  while load < W           ظرفیت کوله پشتی
8.      remove item i with highest bi
9.      a = min(wi, W - load)     کمترین مقدار را بردار
10.     xi = a
11.     load = load + a
```

الگوریتم های مقدماتی گراف

یک گراف مانند G را با زوج $G=(V,E)$ نمایش می دهیم که در آن V مجموعه رئوس و E را مجموعه یال ها می نامیم

V اول نقاط انتهایی یال

$E \subseteq V \times V$

$(u,v) \in E$

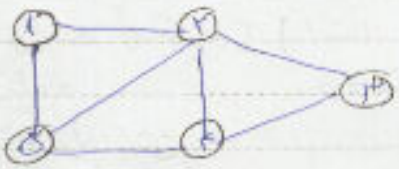
گراف ها دو دسته اند: بی جهت و جهت دار

در بی جهت یال ها به صورت مجموعه دو عضوی هستند $\{V,U\}$

در جهت دار به صورت زوج مرتب هستند $(u,v) \neq (v,u)$

مثال

یک گراف بی جهت

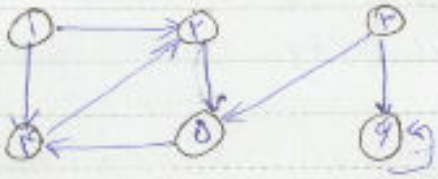


$V=\{1,2,3,4,5\}$

$E=\{\{1,2\},\{1,5\},\{2,3\},\{2,4\},\{2,5\},\{3,4\},\{4,5\}\}$

مثال

یک گراف جهت دار $G=(V,U)$



$(1,2) \in E$ ، $(2,1) \notin E$

$(6,6)$ سه طوقه

روش های نمایش گراف

۱- لیست های همجواری

۲- ماتریس همجواری (مجاورت)

لیست های همجواری

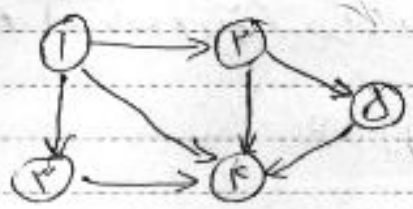
نمایش لیست همجواری گراف $G=(V,E)$ شامل یک آرایه Adj است که شامل یک تعالیت است. هر عضو Adj به یک از این لیست ها اشاره دارد.

اگر $(u,v) \in E$ آنگاه رئوس v مجاور به E است.

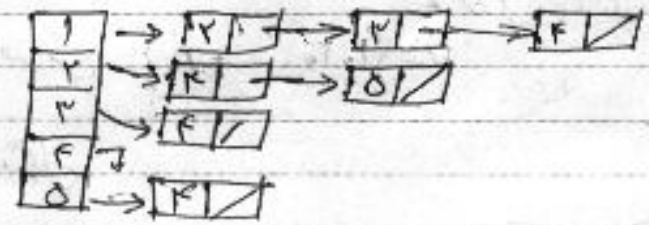
AVA

برای یک راس مانند u، Adj[u] یک اشاره‌گر به لیستی از تمامی رئوس است که مجاور به u هستند.

مثال



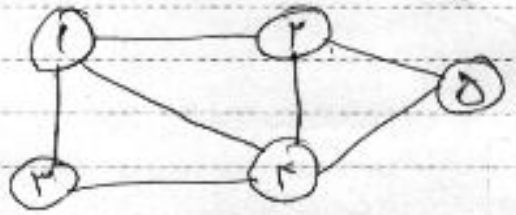
نمایش لیست مجاورتی

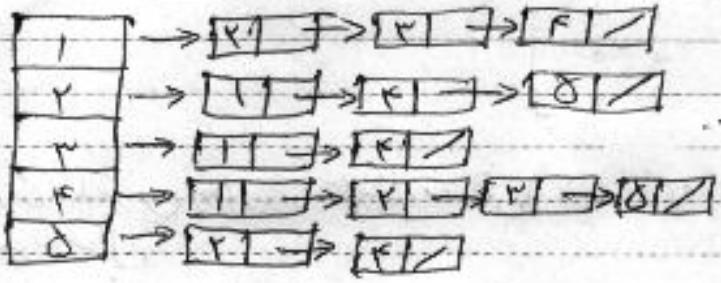


طول لیست‌ها = V

↳ $V = |E|$

مثال





طول لیست‌ها = ۱۴

در روش نمایش لیست مجاورتی حافظه مصرفی از مرتبه زیر است.

$$O(V)+O(E)=O(V+E)$$

سوال

آیا $(u,v) \in E$ در زمان $O(V)$ است؟

پاسخ به سوال

لیست کردن تمامی رئوس مجاور به راس u در زمان $O(V)$

نکته

طول هر لیست مجاورتی حداکثر $|V|$ و مجموع طول لیست‌های مجاورتی $|E|$ است.

Subject:
Year: Month: ۴۰ Date:

روش نمایش ماتریس مجاورتی

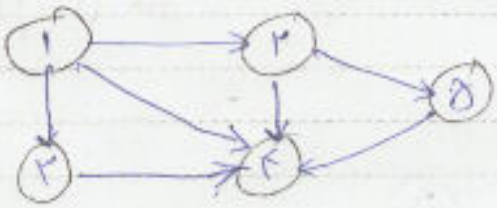
برای نمایش ماتریس مجاورتی گراف $G=(V,E)$ از یک ماتریس $|V|\times|V|$ مانند A استفاده می کنیم که درآرایه های آن به صورت زیر تعریف می شود

$A=(a_{ij})$

$$a_{ij}=\begin{cases}1 & \text{if } (i,j)\in E\\ 0 & \text{if } (i,j)\notin E\end{cases}$$

فرض $V=\{1,2,\dots,n\}$

مثال



$$A=\begin{array}{c} \\ ۱\\ ۲\\ ۳\\ ۴\\ ۵\end{array}\begin{array}{c}\begin{array}{ccccc}۱ & ۲ & ۳ & ۴ & ۵\end{array}\\ \begin{bmatrix}۰ & ۱ & ۱ & ۱ & ۰\\ ۰ & ۰ & ۰ & ۱ & ۱\\ ۰ & ۰ & ۰ & ۱ & ۰\\ ۰ & ۰ & ۰ & ۰ & ۰\\ ۰ & ۰ & ۰ & ۱ & ۰\end{bmatrix}\end{array}$$

در روش ماتریس مجاورتی حافظه مصرفی $\Theta(V\times V)=\Theta(V^2)$ می باشد

پاسخ به سوال

$(i,j)\in E \longrightarrow O(1)$

لیست کردن تمام رئوس مجاور به رأس i $\longrightarrow \Theta(V)$

نکته

پیچیدگی زمانی الگوریتم های گراف بر اساس دو پارامتر تعداد رئوس و تعداد یال ها ارزیابی می شود

گراف چگال $\longrightarrow |E|=|V|^2$

گراف تنک $\longrightarrow |E|=|V|$

AVA

بهتر است با روشی که مجاورتی آنها را نشان دهیم.

روش های پیمایش گراف

۱- روش اولین عرض (BFS) یا سطح

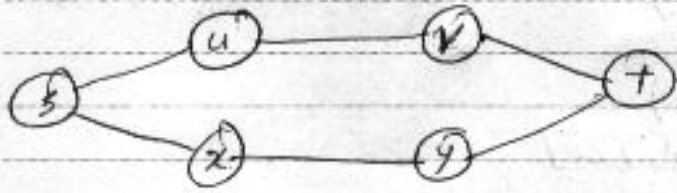
۲- روش اولین عمق (DFS)

منظور از پیمایش گراف ملاقات رئوس گراف در یک ترتیب به خصوص می باشد و با شروع از یک راس مشخص و معین و یا با شروع از هر راس دلخواه

روش پیمایش اولین عرض (سطح) (BFS)

فرض کنیم گراف $G(V,E)$ همراه با راس s متعلق به راس صدا $s \in V$ داده شده است. پیمایش BFS از گراف G با شروع از راس آغازین s تمام رئوس قابل دسترس (رئوسی که از s به آنها در G وجود دارد) از s را ملاقات نمایم.

ترتیب ملاقات رئوس به گونه ای است که رئوس سطحی تر زودتر از رئوس عمیق تر ملاقات می شوند.



راس	عمق (ارتفاع آنها در درخت)
s	۰
u و x	۱
v و y	۲
t	۳

برای هر راس $u \in V$

$u.d$ = طول مسیر (تعداد یال ها) از s به u

$u.\pi$ = راس ماقبل u در مسیر از s به u

$u.color$ = سفید white / خاکستری gray / سیاه Black

رنگ تمام رئوس در ابتدا سفید است در طول اجرای الگوریتم هر راس هنگامی که ملاقات می شود رنگش خاکستری می شود. و وقتی که تمام رئوس مقابل از آن ملاقات شوند رنگش سیاه می شود.

Subject:
Year: Month: ۴۲ Date:

ایده BFS

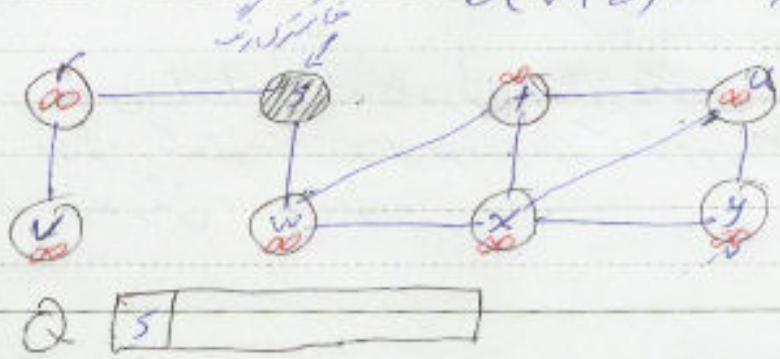
از یک صف مانند Q استفاده می‌کنیم که در ابتدا انتهای s در آن قرار دارد. در طول اجرای BFS همواره این را از ابتدای صف Q خارج کرده و تمامی رئوس ملاقات نشده مجاور به آن را ملاقات کرده و آن را به Q اضافه می‌کنیم. الگوریتم زمانی خاتمه می‌یابد که صف تهی شود.

الگوریتم BFS

```
BFS(G,S)
1.  for each u ∈ G.V-{s}
2.      u.color=white
3.      u.d=∞
4.      u.π=NULL
5.  s.color=Gray
6.  s.π=NULL
7.  s.d=0
8.  Q=∅
9.  ENQUEUE(Q,s)          اضافه کردن s به انتهای صف Q
10. while Q≠∅
11.     u=DEQUEUE(Q)
12.     for each v∈Adj[u]
13.         if v.color=white
14.             v.color=Gray
15.             v.π=u
16.             v.d=u.d+1
17.             ENQUEUE(Q,v)
18.     u.color=Black
```

زمان اجرا $O(V+E)$

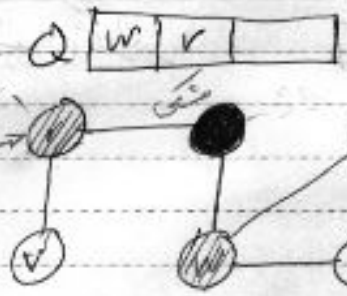
مثال



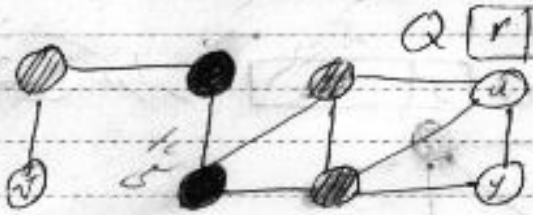
AVA

Subject:
Year: Month: Date:

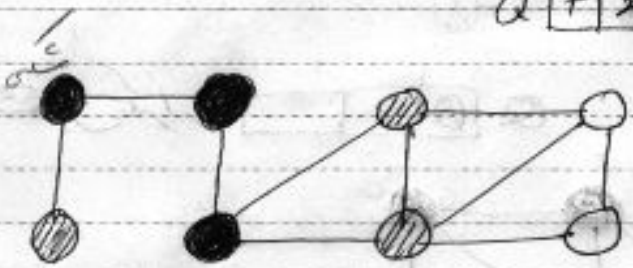
در اولین تکرار



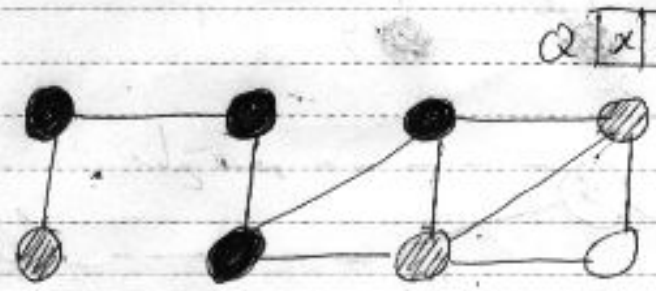
در دومین تکرار



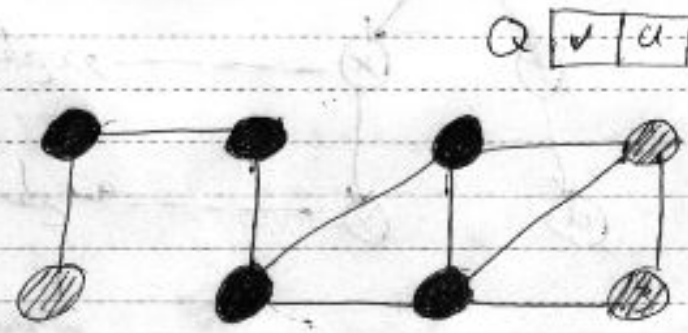
سومین تکرار



چهارمین تکرار

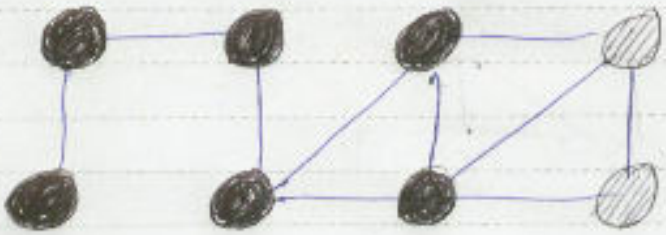


پنجمین تکرار



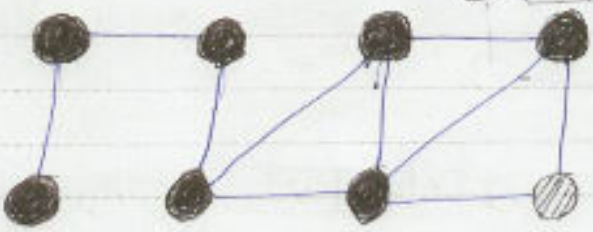
ششمین تکرار

Q [u | y]



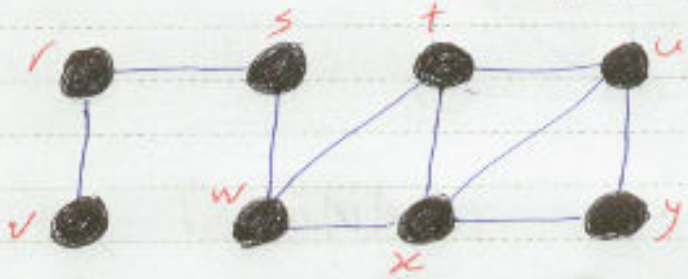
هفتمین تکرار

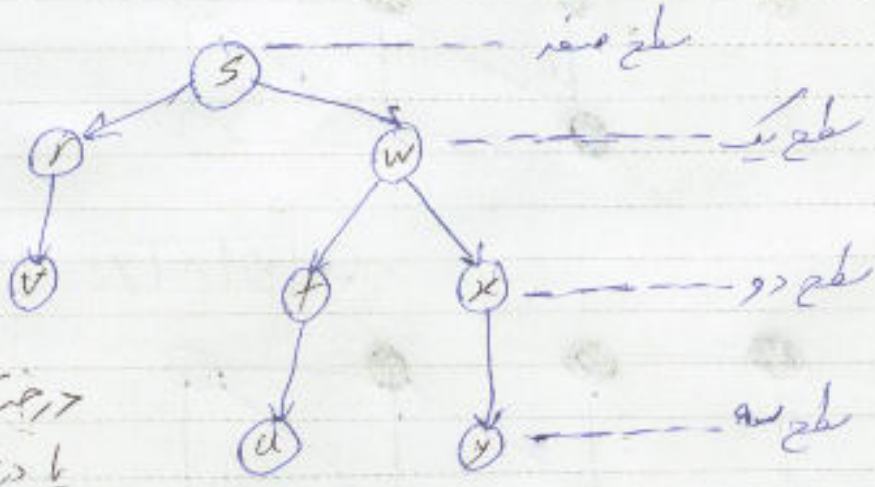
Q [y]



هشتمین تکرار

Q [ ]





درخت اولین عمق
یا درخت BFS

AVA

Subject:
Year: Month: ۴۸ Date:

پیمایش اول عمق DFS

در اینجا برعکس روش BFS پیمایش به صورت عمقی انجام می شود. تفاوت دیگر این است که راسی به نام راس آغازین مشخص می شود و می توان از هر راسی شروع به پیمایش کرد. پس تمامی رئوس ملاقات می شوند.

در این روش برای راس u که اخیرا ملاقات شده یک راس از میان رئوس مجاور به آن که قبلا ملاقات نشده انتخاب می شود و تا دنبال چنین رئوس مجاوری است. اگر تمامی رئوس مجاور به u ملاقات شده باشند یک عقب گرد به بالا به راسی که u از طریق آن ملاقات شده صورت می گیرد و تا یک راس مجاور دیگر با آن بررسی شود.

u.d : زمان کشف راس u (زمان ملاقات)

u.f : زمان خاتمه راس u (زمانی است که تمامی رئوس مجاور به u بررسی شده و دیگر نیازی به u نیست و از آن عقب گرد می کنیم)

برای هر راس $u \in V$

از یک متغیر سراسری time استفاده می کنیم که در ابتدا مقداردهی صفر است.

الگوریتم DFS به صورت زیر است.

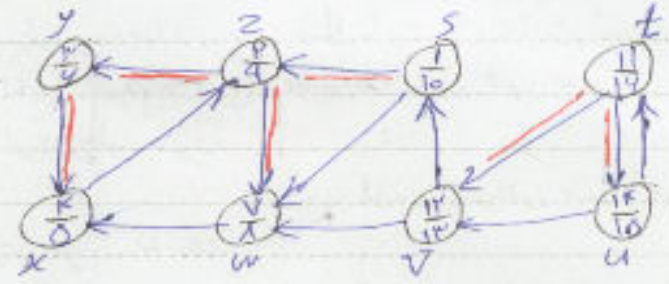
```
DFS(G)
1. for each u∈V
2.     u.color = white
3.     u.π = Null
4. time = 0
5. for each u∈G.V
6.     if u.color == white
7.         DFS_visit(G,u)

DFS_visit(G,u)
1. time = time+1
2. u.d = time
3. u.color = Gray
4. for each v∈Adj[u]
5.     if v.color == white
```

AVA

Subject:
Year: Month: ۴۷ Date:

۶. u.π = u
۷. DFS_visit(G,V)
۸. u.color = Black
۹. time = time+1
۱۰. u.f = time

مثال



$u = \frac{u.d}{u.f}$

درخت اولین عمق

یک درخت دیگر اولین عمق

جنگل اولین عمق

ویژگی های DFS

قضیه: راس v نسل محض راس u در جنگل اولین عمق گراف G=(V,E) است اگر و فقط اگر u.d < v.d < v.f < u.f

قضیه مسیر سفید

در جنگل اولین عمق گراف G=(V,E) راس v یک راس از نسل راس u است اگر و فقط اگر در زمان u.d مسیری از u به v که تنها شامل رئوس سفید است وجود داشته باشد.

قضیه

در پیمایش DFS گراف بدون جهت G=(V,E) هر یال گراف یا درختی است و یا یال برگشت

AVA

Subject:
Year: Month: ۴۷ Date:

قضیه پرانتز: در پیمایش DFS گراف G=(V,E) به ازای هر دو راس $u, v \in V$ تنها یکی از شرایط زیر برقرار است:

۱- $[u.d, v.f] \cap [v.d, u.f] = \phi$

۲- $[u.d, u.f] \subset [v.d, v.f]$

۳- $[v.d, v.f] \subset [u.d, u.f]$

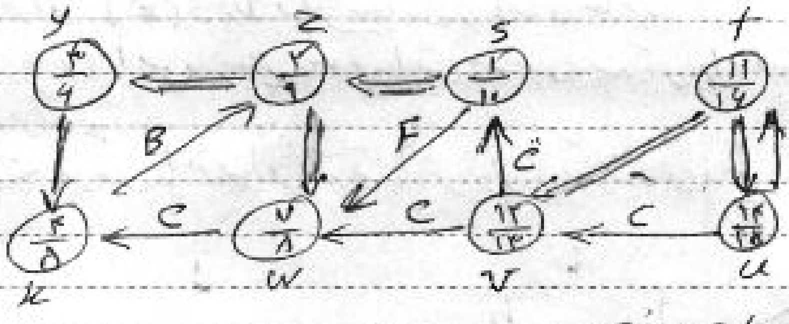
طبقه بندی یال ها

با اجرای DFS روی گراف G=(V,E) می توان یال های گراف را به صورت زیر طبقه بندی کرد.

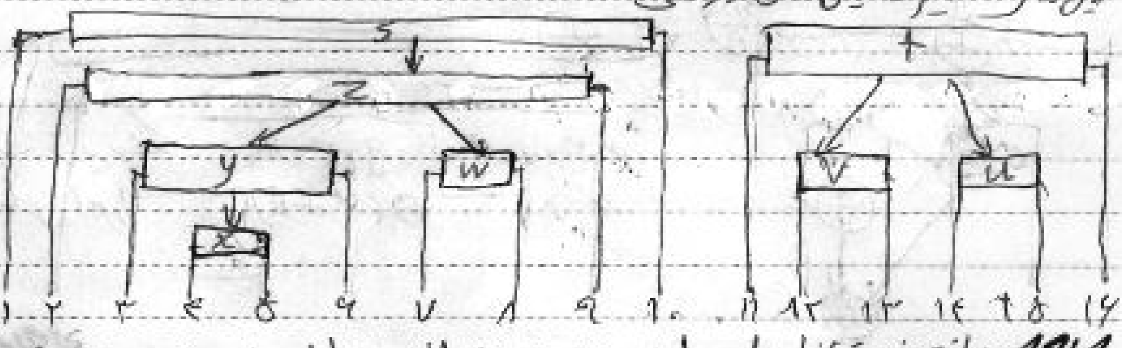
هر یال $(u,v) \in E$:
- درختی ← متعلق به یک درخت از جنگل اول عمق
- غیر درختی:
  - $v.d < u.d < v.f < u.f$ ← Forward
  - $v.d < u.d < u.f < v.f$ ← Back
  - $v.d < v.f < u.d < u.f$ ← cross

یال forward یالی است (u,v) که راس جد u را به راس نسل u وصل می کند.
یال Back یالی مانند (u,v) که راس نسل u را به راس جد u وصل می کند.
سایر یال های غیر درختی cross هستند.

مثال



یال های ضخیم = یال های درختی



(s(z(y(x))(w)))(t(v)(u)) پرانتز بندی متناظر با پیمایش DFS گراف مذکور

AVA

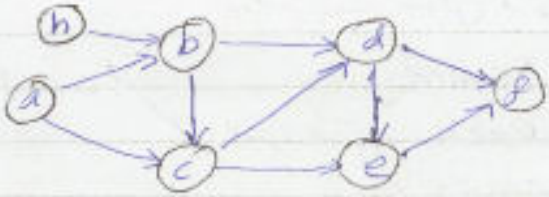
Subject:
Year: Month: ۴۸ Date:

نکته: به صورت قرارداد یال های طوقه را در طبقه بندی یال های Back قرار می دهیم.
(یال طوقه یالهایی است که به خودشان برگشت دارند)

کاربردهای DFS
- مرتب سازی توپولوژی
- یافتن مولفه های همبند قوی

مرتب سازی توپولوژی
گراف جهت دار $G=(V,E)$ داده شده است. یک ترتیب توپولوژیک از رئوس این گراف ترتیبی از رئوس است به طوری که اگر $uv \in E$ باشد آنگاه u قبل از v ظاهر شده باشد.

مثال

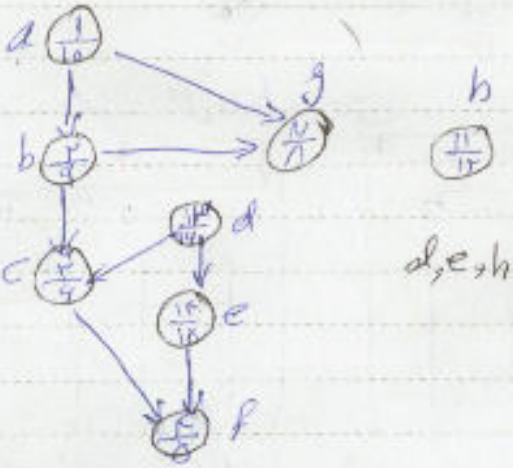


توجه: گراف مورد نظر خاقد دور است.

یک ترتیب توپولوژی: h, a, b, c, d, e, f
یک ترتیب توپولوژی دیگر: a, h, b, c, d, e, f

مرتب سازی توپولوژی گراف G
۱- فراخوانی DFS(G) برای محاسبه زمانهای خاتمه
۲- هر زمان رأسی زمان خاتمه اش مشخص شد آن را به ابتدای یک لیست پیوندی اضافه می کنیم.
۳- لیست پیوندی را به عنوان ترتیب توپولوژی برگشت می دهیم

مثال



ترتیب توپولوژیک = d, e, h, a, b, g, c, f
زمان اجرا $\Theta(V+E)$

AVA

مولفه های همبند قوی گراف

برای گراف جهت دار داده شده $G(V,E)$ یک زیر مجموعه $C \subseteq V$ را یک مولفه همبند قوی از گراف G گوییم هرگاه هر دو راس از مجموعه C از یکدیگر قابل دسترسی باشند و C ماکزیمال باشد به این معنی که نتوان راس دیگری به آن اضافه کرد بطوری که زیرمجموعه جدید از یکدیگر قابل دسترسی باشند.

روال یافتن مولفه های همبند قوی گراف $G(V,E)$

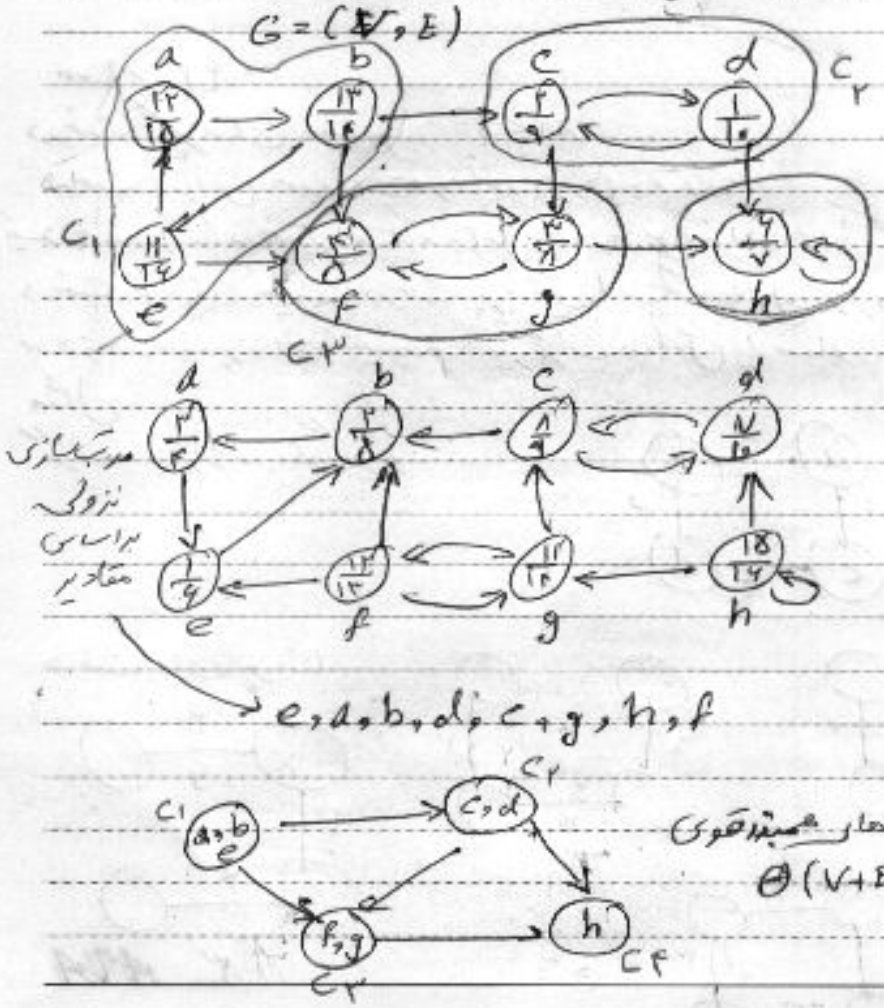
۱- فراخوانی $DFS(G)$ برای محاسبه زمانهای خاتمه رئوس

۲- محاسبه گراف ترانهاده $G \leftarrow G^T=(V,E^T)$

۳- فراخوانی $DFS(G^T)$ با این تفاوت که در حلقه اصلی روال DFS به جای انتخاب دلخواه رئوس آنها را براساس ترتیب نزولی (از بزرگ به کوچک) مقادیر f رئوس انتخاب می کنیم

هر درختی که اولین عمق حاصل می شود رئوس یک مولفه همبند قوی گراف را تشکیل می دهد.

مثال



مرتب سازی نزولی براساس مقادیر

$G^T=(V,E^T)$

e, a, b, d, c, g, h, f

گراف مولفه های همبند قوی

زمان اجرا $\Theta(V+E)$

AVA

Subject:
Year: Month: Date:

دولم ممم

لم ۱ فرض کنیم C و C' دو مولفه همبندی قوی گراف $G=(V,E)$ باشند و $u,v \in C$ و $u',v' \in C'$ در این صورت اگر G شامل مسیر $u \to u' \to v' \to v$ باشد آنگاه G نیز شامل مسیر $v \to u \to u' \to v'$ باشد.

لم ۲ تعریف می کنیم برای مجموعه $C \subseteq V$

$d(C) = \min_{u \in C} \{u.d\}$ مینیمم زمان کشف راس

$f(C) = \max_{u \in C} \{u.f\}$ ماکزیمم زمان خاتمه رئوس

اگر با تعریف بالا اگر C و C' دو مولفه همبندی قوی گراف $G=(V,E)$ باشند آنگاه اگر $(u,v) \in E$ باشد که $u \in C$ و $v \in C'$ آنگاه $f(C) > f(C')$

فصل ۲۳

درخت پوشای مینیمم MST

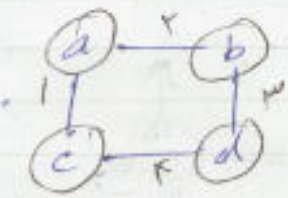
درخت: گراف همبند است و فاقد دور می باشد.

یک درخت پوشا درختی است که شامل همه رئوس گراف می باشد.

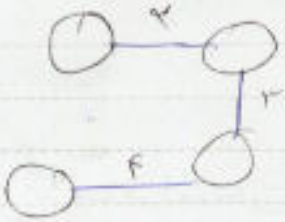
درخت پوشای مینیمم درخت پوشایی است که مجموع وزن یال های آن نسبت به هر درخت پوشای مینیمم دیگر گراف حداقل یا مساوی است.

مثال

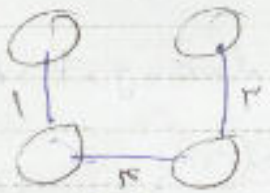
گراف زیر را در نظر بگیرید.



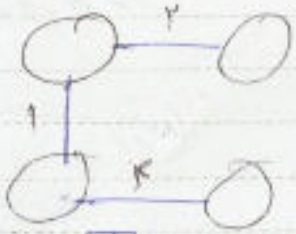
درخت های پوشا



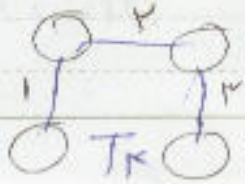
T_1



T_2



T_3



T_4

AVA

Subject:
Year: Month: ۵۱ Date:

وزن
$w(T_1) = ۲+۴+۳+۶ = ۹$
$w(T_2) = ۱+۴+۳+۴ = ۸$
$w(T_3) = ۱+۴+۲+۴ = ۷$
$w(T_4) = ۱+۲+۴+۳ = ۶$

T_4 : درخت پوشای مینیمم گراف G می باشد.

مقدمات

یک راهبرد حریصانه برای ساخت MST برای گراف $G=(V,E)$ به صورت زیر می باشد
یک مجموعه A که شامل یال های MST هست داریم که در ابتدا تهی است در هر گام یک یال امن (safe) را به مجموعه یال های A اضافه می کنیم. (امن: یال (u,v) برای مجموعه یال ها امن است هرگاه اضافه کردن آن به A ایجاد دور نکند)

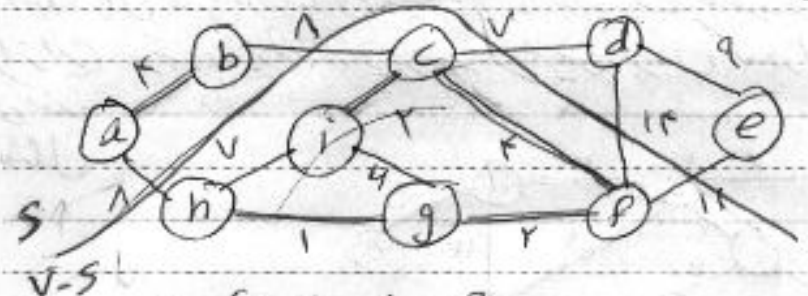
برش cut

یک برش $(S, V-S)$ از یک گراف $G=(V,E)$ افرازی از مجموعه رئوس V است به یالی مانند $(u,v) \in E$ وجود دارد که $u \in S$ و $v \in V-S$ یا برعکس

تعریف

۱- می گوییم یک برش با مجموعه یال های A سازگار است هرگاه هیچ یالی از A را قطع نکند
۲- در میان یال هایی که یک برش آنها را قطع می کند یال با مینیمم وزن را یال سبک گوییم

مثال



$S = \{a, b, d, e\}$
$V-S = \{c, i, h, g, f\}$

A : یال هایی انتخاب شده که با خطوط ضخیم مشخص شده است. از میان یال های که برش را قطع می کنند یال (d,e) با وزن ۹ یک یال سبک است که برای A امن است. یعنی اضافه کردن آن به A باعث دور نمی شود.

قضیه

برای گراف همبند و جهت دار $G=(V,E)$ همراه با تابع وزن w و مجموعه $A \subseteq E$ که بعضی از درخت های پوشای مینیمم G است اگر $(S, V-S)$ برش

Subject:
Year: Month: ۵۲ Date:

سازگار با A باشد آنگاه یال سبک که برش را قطع می کند برای A ایمن است.
الگوریتم کلی برای تولید درخت پوشای مینیمم به صورت زیر است.

```
Generic_MST(G,w)
1. A = ∅
2. while A does not from a sprining tree
3. find an edge (u,v) that is safe for A
4. A = A ∪ {(u,v)}
5. return A
```

دو الگوریتم کراسکال و پریم از ایده بالا برای یافتن MST استفاده می کنند.
هردوی این الگوریتم ها از نتیجه قضیه زیر استفاده می کنند.

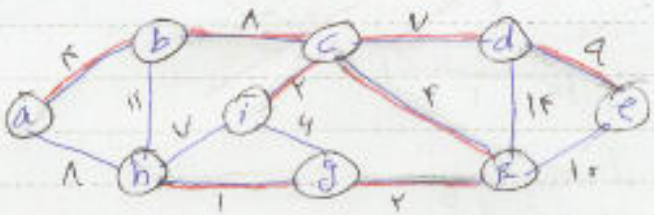
قضیه

برای گراف همبند و بهمت دلخواه $G=(V,E)$ همراه با تابع وزن w و مجموعه $A \subseteq E$ که زیرمجموعه از MST گراف G است فرض کنید $C=(V_c,E_c)$ یک مولفه همبند در جنگل G_A باشد اگر (u,v) یال سبک باشد که c را به مولفه دیگری وصل کند آنگاه (u,v) برای A ایمن است.

الگوریتم کراسکال

ابتدا یال ها را براساس وزن آنها به صورت صعودی مرتب می کند یال ها را از این ترتیب برداشته و درصورت ایمن بودن که در ابتدا تهی است اضافه می کند.

مثال



(h,g) ✓ , (i,c) ✓ , (g,f) ✓ , (c,f) ✓ , (a,b) ✓ , (i,g) ✗ → ایجاد دور , (c,d) ✓ , (h,i) ✗ , (b,c) ✓ , (a,h) ✗

, (d,e) ✓ , (e,f) ✗ , (b,h) ✗ , (d,f) ✗

AVA

Subject:
Year: Month: ۸۳ Date:

برای پیاده سازی براسکال از ساختمان داده مجموعه‌های مجزا استفاده می‌شود.

```
MST-Kraskal(G,W)
1. A=∅
2. for each v∈V
3. make-set(V)
4. sort GE into nondecreasing order
5. for each (u,v)∈E
6. if find-set(u)≠find-set(v)
7. A=A∪{(u,v)}
8. union(u,v)
9. return A
```

زمان اجرا برابر $O(E \lg V)$

الگوریتم پریم

تفاوت عمده با کراسکال در این است که در طول اجرای الگوریتم A به جای اینکه جنگل باشد همواره یک درخت است.

مقدمات

در ابتدا راس دلخواه r به عنوان ریشه MST انتخاب می‌شود. برای هر راس $v \in V$ فیلد v.key را برابر بینهایت و وزن یال‌هایی که راس u را به راس در A وصل می‌کند تعریف می‌کنیم.

در ابتدا به ازای هر راس $u \in U$ داریم: $u.key=\infty$

همچنین فیلد v.π به پدر v در درخت A است. در طول اجرای الگوریتم مجموعه A به صورت زیر است.

$$A=\{(v, v.\pi) \mid u \in V-\{r\}-Q\}$$

صف اولویت مینیم است که شامل رئوسی از گراف است که در A نیستند. رئوس در این صف براساس مقادیر key خود قرار دارند.

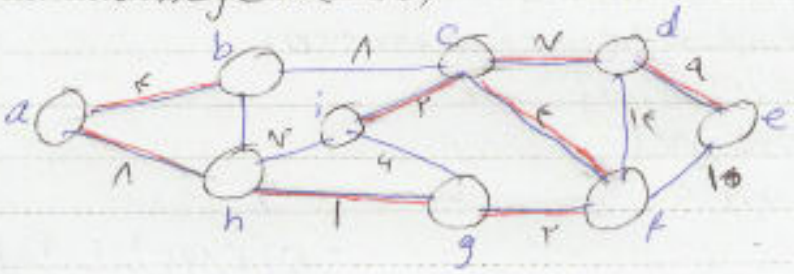
```
MST-Prim(G,W,r)  ← ریشه
1. for each u∈V
2. u.key=∞
3. u.π=Nil
4. r.key=0
```

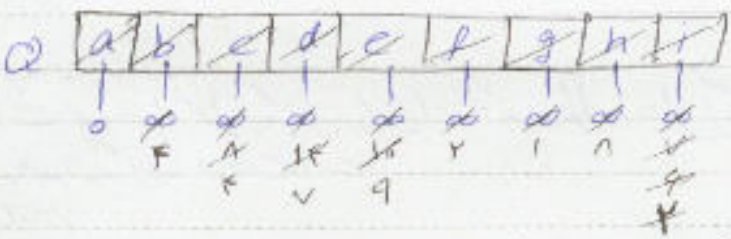
AVA

Subject:
Year: Month: ۵۲ Date:

5. $Q = V$
6. while $Q \neq \emptyset$ خارج نشد
7. $u = Extract\text{-}min(Q)$
8. for $u \leftarrow Adj[u]$
9. if $v \in Q$ and $w(u,v) < v.key$
10. $v.\pi = u$
$v.key = w(u,v)$

مثال





کوتاه ترین مسیرها در گراف از مبدا واحد

گراف وزن دار $G = (V, E)$ همراه با راس مبدا s متعلق به V داده شده است. هدف یافتن کوتاه ترین مسیر ها از راس s به سایر رئوس G است.
دو الگوریتم برای این کار وجود دارد. ۱- الگوریتم بلمن فورد ۲- الگوریتم دیکسترا

مقدمات

طول کوتاهترین مسیر از u به v را با $\delta(u,v)$ نشان می دهیم.
اگر مسیری از u به v باشد $\Leftarrow \delta(u,v) < \infty$ وگرنه
نباشد $\Leftarrow \delta(u,v) = \infty$

برای هر راس $v \in V$ دو فیلد زیر را تعریف می کنیم.
$v.d$ = طول کوتاهترین مسیر از s به v
$v.\pi$ = راس متقابل v در کوتاهترین مسیر از s به v
این الگوریتم ها از دو روال زیر استفاده می کنند.

Intiatize-single-sorce (G, s)
Relax (u, v, w)

AVA

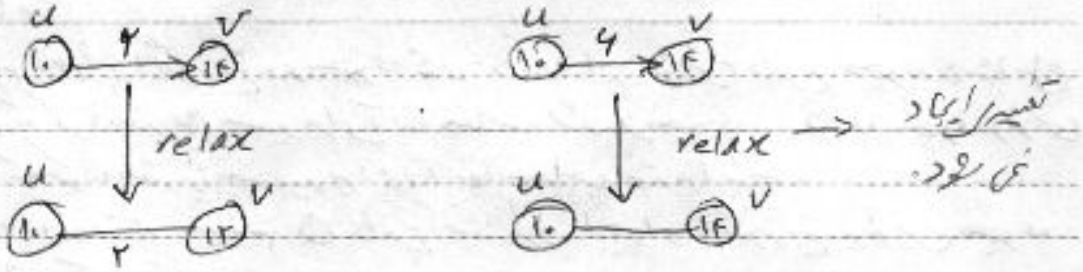
Subject:
Year: Month: Date:

```
Intiatize-single-Sorce(G,s)
1.  for each v∈V
2.      v.d=∞
3.      v.π=Nil
4.  s.d=0
```

زمان اجرا برابر با $\Theta(V)$

```
Relax(u,v,w)
1.  if v.d>u.d+w(u,v)
2.      v.d=u.d+w(u,v)
3.      v.π=u
```

زمان اجرا برابر با $\Theta(1)$



الگوریتم بلمن فورد

در این الگوریتم یال با وزن منفی را قبول می کند و دور با وزن منفی را تشخیص می دهد و با برگرداندن مقدار false به اجرا خاتمه می دهد.

```
Bellman-Ford(G,w,s)   تابع وزن
1.  Intiatize-single-Sorce(G,s)
2.  for i=1 to |V|-1
3.      for each (u,v)∈E
4.          relax(u,v,)
5.  for each (u,v)∈E
6.      if v.d>u.d+w(u,v)
7.          return false
8.  return True
```

اصل الگوریتم تا خط ۴ می باشد

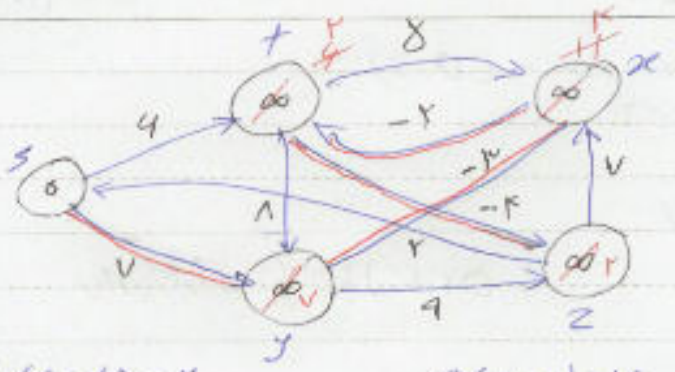
برای تشخیص دور با وزن منفی

زمان اجرا برابر با $O(VE)$

AVA

Subject:
Year: Month: ۵۶ Date:

مثال



$\delta(s,t) = ۲$ $\delta(s,y) = ۷$

$\delta(s,x) = ۴$ $\delta(s,z) = -۲$

الگوریتم دیکسترا

وزن یال‌های گراف باید نامنفی باشد و از گلیک حریصانه استفاده می‌کند.

مقدمات

در طول اجرای الگوریتم دو مجموعه که در S-V از رئوس داریم. مجموعه S شامل رئوس که کوتاه‌ترین مسیر برای آنها محاسبه شده و مجموعه V-S مجموعه رئوسی است که هنوز کوتاه‌ترین مسیر برای آنها محاسبه نشده است.

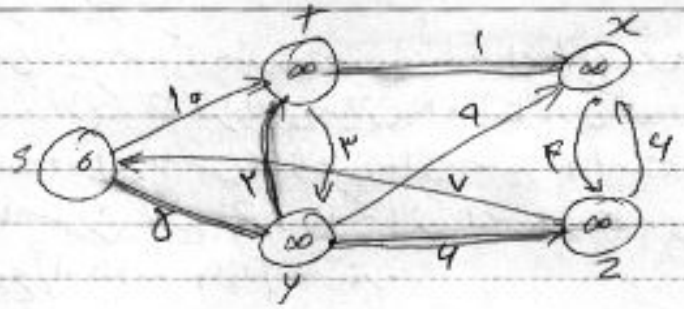
صف اولویت مینیمم Q داریم که در طول اجرای الگوریتم رئوس مجموعه V-S در آن قرار دارند.

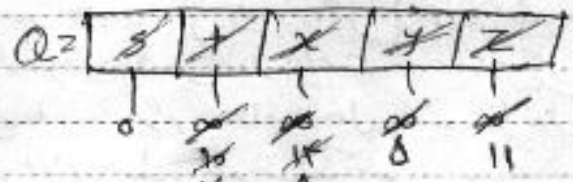
در یک قدم آنقدر گلیک راسی مانند u از کلید مینیمم Q خارج و $\delta(s,u)$ راس آن محاسبه می‌شود.

```
Dijkstra(G, w, s)
1. Initialize_single_source(G, s)
2. S = ∅
3. Q = V
4. while Q ≠ ∅
5.     u = Extract_Min(Q)
6.     S = S ∪ {u}
7.     for each v ∈ Adj[u]
8.         Relax(u, v, w)
```

Subject:
Year: Month: ۵۷ Date:

مثال





زمان اجرای دیکسترا وابسته به پیاده سازی صف اولویت مینیم Q است. در صورت پیاده سازی Q با

۱- آرایه ← زمان اجرا $O(V^2+E)$

۲- هرم دودویی مینیم ← زمان اجرا $O(E \lg V)$ گاریتم

۳- هرم فیبوناچی ← زمان اجرا $O(V \lg V+E)$

کوتاهترین مسیرها بین تمام رئوس گراف

گراف $G=(V,E)$ داده شده است به تابع وزن $E \to R$: وزن روی یال‌های آن تعریف شده است. هدف محاسبه طول کوتاهترین مسیر بین تمام جفت رئوس گراف است.

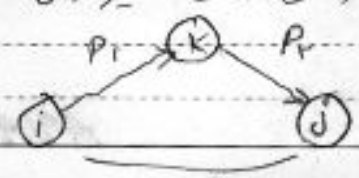
الگوریتم‌های متداول برای حل این مسئله: فلوید- وارشال / جانسون

الگوریتم فلوید- وارشال

ورودی: ماتریس مجاورتی

تکنیک: برنامه نویسی پویا

فرض می‌کنیم گراف ورودی $G(V,E)$ است و مجموعه رئوس آن $V=\{1,2,\dots,n\}$ است. برای هر مسیری مانند p از راس i به راس j به صورت $i \to \dots \to j$ است. هر راس به جز رئوس ابتدایی و انتهایی راس میانی مسیر p می‌نامیم



اگر p کوتاهترین مسیر باشد $\begin{cases} p_1 \text{ کوتاهترین مسیر از } i \text{ به } k \\ p_2 \text{ کوتاهترین مسیر از } k \text{ به } j \end{cases}$

AVA

Subject:
Year: Month: ۵ ۸ Date:

فرض کنید هرکدام کوتاهترین مسیر از i به j با رئوس میانی در $\{1,2,\dots,k\}$ باشد بسته به این که p از راس میانی k استفاده می کند از دو حالت استفاده می کند.
حالت اول: راس k راس میانی مسیر p است. پس در نتیجه هریک از دو کوتاهترین مسیر p_1 و p_2 که از i به k و k به j هستند دارای رئوس میانی از مجموع $\{1,2,\dots,k-1\}$ هستند.
حالت دوم: راس k راس میانی مسیر p نباشد. در نتیجه تمامی رئوس میانی p در مجموعه $\{1,2,\dots,k-1\}$ می باشد.
با بررسی همین دو حالت که ساختار جواب بهینه را مشخص می کند یک رابطه بازگشتی برای محاسبه حل بهینه ارائه می دهیم
برای این منظور تعریف می کنیم $d_{ij}^{(k)}$ جداگر
طول کوتاهترین مسیر از i به j که تنها از رئوس میانی $\{1,2,\dots,k\}$ استفاده می کند
در این صورت داریم

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k=0 \\ \min\left(d_{ik}^{(k-1)} + d_{kj}^{(k-1)},\ d_{ij}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

$1 \leq k \leq n$ داریم

سپس از آن برای حفظ و نگهداری کوتاهترین مسیر ماتریس π را به صورت بازگشتی زیر مشخص می کنیم

$$\pi_{ij}^{(0)} = \begin{cases} Nil & \text{if } i=j \text{ or } w_{ij}=\infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

ماتریس وزن ←

$$w_{ij} = \begin{cases} w(i,j) & \text{if } (i,j) \in E \\ \infty & \text{if } (i,j) \notin E \end{cases}$$

ماتریس مجاورت ←

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

AVA

Subject:
Year: Month: ۵۹ Date:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

* پیش داریم

π_{ij} = راس ماقبل j در کوتاهترین مسیر از i به j

در طی اجرای الگوریتم فلوید-وارشال ماتریس های زیر ساخته می شوند:

$D^{(0)} \to \pi^{(0)}$

$D^{(1)} \to \pi^{(1)}$

$\vdots$

$D^{(k)} \to \pi^{(k)}$

$\vdots$

$D^{(n)} \to \pi^{(n)}$

که در آن $D^{(k)} = (d_{ij}^{(k)})$ و $\pi^{(k)} = (\pi_{ij}^{(k)})$ و $D^{(0)} = W$ ماتریس وزن گراف

و ماتریس $\pi^{(0)}$ از روی تعریف $\pi_{ij}^{(0)}$ به دست می آید.

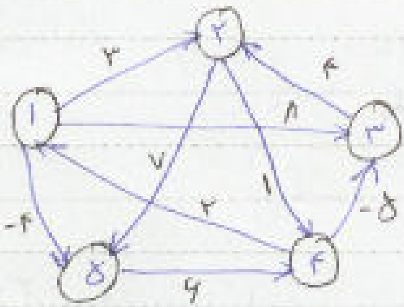
الگوریتم فلوید-وارشال به صورت شبه کد:

```
Floyd_warshall(W)  ⟵ ماتریس وزن گراف G
1. n = W.rows
2. D^(0) = W
3. for k = 1 to n
4.     Let D^(k) = (d_ij^(k)) be a new n×n matrix
5.     for i = 1 to n
6.         for j = 1 to n
7.             d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8. return D^(0)
```

زمان اجرا برابر با $\Theta(n^3)$

مثال

نحوه شکل‌گیری ماتریس‌های $D^{(0)}, D^{(1)}, \ldots, D^{(5)}$ به صورت زیر می‌باشد.



$D^{(0)} = W =$

	۱	۲	۳	۴	۵
۱	۰	۳	۸	∞	-۴
۲	∞	۰	∞	۱	۷
۳	∞	۴	۰	∞	∞
۴	۲	∞	-۵	۰	∞
۵	∞	∞	∞	۶	۰

$D^{(1)} =$

	۱	۲	۳	۴	۵
۱	۰	۳	۸	∞	-۴
۲	∞	۰	∞	۱	۷
۳	∞	۴	۰	∞	∞
۴	۲	۵	-۵	۰	-۲
۵	∞	∞	∞	۶	۰

$D_{ij}^{(1)} = \min\left(d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)}\right) =$
$\min(\infty, 2+3) = 5$

$D^{(2)} =$

۰	۳	۸	۴	-۴
∞	۰	∞	۱	۷
∞	۴	۰	۵	۱۱
۲	۵	-۵	۰	-۲
∞	∞	∞	۶	۰

$D^{(3)} =$

۰	۳	۸	۴	-۴
∞	۰	∞	۱	۷
∞	۴	۰	۵	۱۱
۲	-۱	-۵	۰	-۲
∞	∞	∞	۶	۰

$D^{(4)} =$

۰	۳	-۱	۴	-۴
۳	۰	-۴	۱	-۱
۷	۴	۰	۵	۳
۲	-۱	-۵	۰	-۲
۸	۵	۱	۶	۰

$D^{(5)} =$

۰	۱	-۳	۲	-۴
۳	۰	-۴	۱	-۱
۷	۴	۰	۵	۳
۲	-۱	-۵	۰	-۲
۸	۵	۱	۶	۰

امتحان پایان ترم نیز تا ترم ۱۴ می‌توان آورد
از بحث میان ترم حدود ۳ تا ۴ نمره و بقیه از
مباحث ترم به بعد می‌آید