برنامه نویسی پویا (روش بالا به پایین)

الگوریتم‌های داینامیک از دسته الگوریتم‌های پرکاربرد هستند که غالبا در مسائل بهینه‌سازی و شمارش تعداد حالات به کار می‌روند.

برای حل مسائل داینامیک به روش بالا به پایین (Memoize)، سه قدم زیر ضروری است:

- استخراج رابطه بازگشتی و شروط پایه
- نوشتن یک برنامه بازگشتی ساده (غیر داینامیک)
- تبدیل کد بازگشتی نوشته شده به داینامیک

برای تبدیل یک کد بازگشتی ساده به یک کد داینامیک کافی است، یک آرایه بگیریم که هر بُعد آن متناظر با یکی از پارامترهای ورودی تابع-بازگشتی می‌باشد.

این آرایه تعداد حالات مختلف ورودی را برای ما مدل می‌کند. به عنوان مثال اگر تابع ما F(P1, P2) باشد. طبق اصل ضرب، این تابع P1xP2 ورودی متفاوت داشته باشد که به هرکدام از اینها یک حالت می‌گوییم.

آرایه متناظر با رابطه بازگشتی، جواب مساله را برای هر حالت نگه می‌دارد و از آنجایی که در این مثال P1xP2 حالت ورودی متفاوت داریم، آرایه ما باید به شکل زیر باشد:

```
int dp[P1][P2];
```

اساس تمامی الگوریتم‌های داینامیک جمله زیر می‌باشد:

حالتی که جوابش قبلا محاسبه شده، نباید دوباره حساب شود.

در تبدیل کد بازگشتی به غیر بازگشتی، در تابع main تمامی خانه‌های آرایه dp را برابر یک مقدار غیر ممکن (در بسیاری از موارد 1-) قرار می‌دهیم، بدین معنا که مقدار این حالت هنوز محاسبه نشده است.

سپس به کد بازگشتی نوشته شده، این شرط پایه را در انتهای شروط پایه اضافه می‌کنیم:

اگر مقدار این حالت قبلا محاسبه شده بود، مقدار آن را از آرایه برگردان (عدم نیاز به محاسبه مجدد)

در غیر این صورت مقدار این حالت را محاسبه کن و در خانه مناسب از آرایه قرار بده، سپس مقدار موجود در آرایه را برگردان.

در مثال فوق (با فرض اینکه مقدار غیر ممکن 1- باشد) :

```
if ( dp[P1][P2] != -1)  //جواب قبلا محاسبه شده
    return dp[P1][P2];
```

مثال اول: nاُمین عدد دنباله فیبوناچی را به کمک روش داینامیک محاسبه کنید.

قدم اول: رابطه بازگشتی در این مساله برابر است با: $F(N) = F(N-1) + F(N-2)$ ___ $F(0) = 0$ ___ $F(1) = 1$

قدم دوم: کد بازگشتی این برنامه عبارت است از:

```
int fibo(int n)
{
	if(n < 2)
		return n;
	return fibo(n - 1) + fibo(n - 2);
}
```

قدم سوم: کد فوق را تبدیل به کد داینامیک می‌کنیم:

```
int dp[1000];

int fibo(int n)
{
	if(n < 2)
		return n;
	if(dp[n] != -1)
		return dp[n];
	dp[n] = fibo(n - 1) + fibo(n - 2);
	return dp[n];
}

int main()
{
	int n = 10;
	for (int i = 0; i <= n; i++)
		dp[i] = -1;
	cout << fibo(n);
}
```

در این مثال ورودی Fibo(N) دارای N حالت متفاوت است (از 1 تا N)

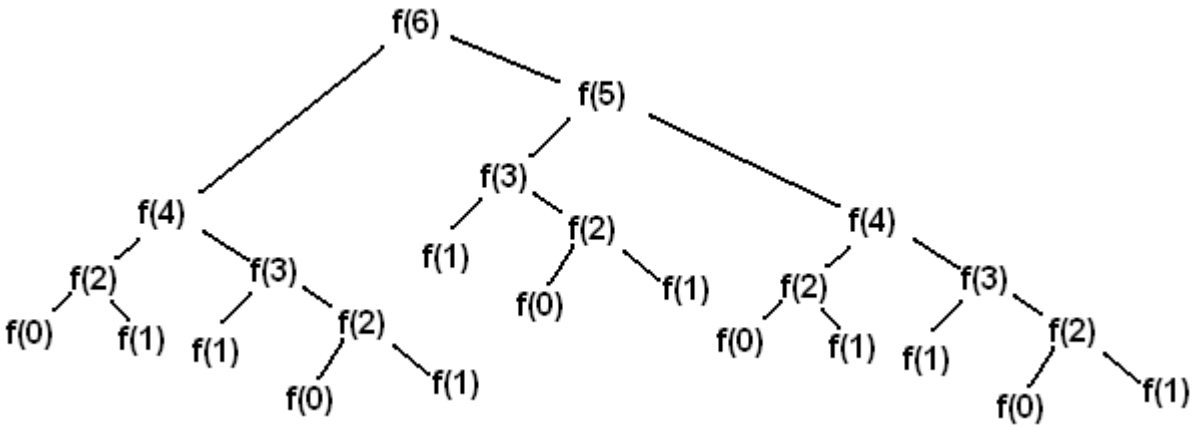
پیچیدگی حافظه این تابع برابر سایز آرایه می‌باشد. O(N)

از آنجایی که مقدار هر خانه از این آرایه تنها یک بار محاسبه می‌گردد پیچیدگی زمانی کد فوق نیز O(N) می‌باشد (زمان پر شدن آرایه)

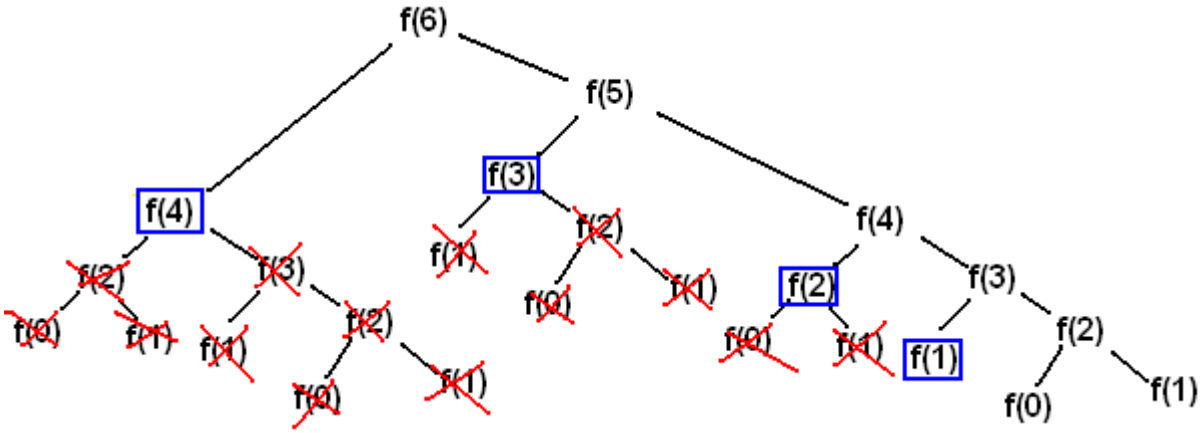
یک سوال: چه مواقعی از الگوریتم‌های داینامیک استفاده می‌کنیم؟

زمانی که حین فراخوانی‌های بازگشتی، یک حالت چندین بار فراخوانی شود. برای مثال به شکل‌های زیر توجه کنید:

قبل از داینامیک:



بعد از داینامیک:



مثال دوم: کد داینامیکی بنویسید که C(N, K) را حساب کند.

قدم اول: طبق قاعده پاسکال می‌دانیم که:

$$C(n,k) = C(n-1,k-1) + C(n-1,k)$$

شروط پایه: اگر N کوچکتر از K بود قطعا جواب برابر صفر است. اگر N و K مساوی بودند جواب برابر 1 می‌باشد.

کد داینامیک:

http://paste.ubuntu.com/24425590/

مثال سوم: برنامه داینامیکی بنویسید که حساب کند با داشتن بی‌نهایت تعداد سکه 1، 5، 10 و 25 تومانی، حداقل چند سکه باید استفاده شود تا بتوان هزینه یک کالای N تومانی را به طور دقیق پرداخت کرد؟

در بسیاری از مواقع رابطه بازگشتی به ما داده نمی‌شود و خودمان باید با توجه به مساله یک رابطه بازگشتی استخراج کنیم. رابطه بازگشتی در این مساله عبارت است از:

$$F(N) = Min \begin{cases} 1 + F(N-1) \\ 1 + F(N-5) \\ 1 + F(N-10) \\ 1 + F(N-25) \end{cases}$$

حالت اول بیانگر این است که اگر یک سکه 1 تومانی خرج کنم، جواب مساله برابر F(N – 1) به علاوه یک سکه خرج شده خواهد بود.
حالت دوم بیانگر این است که اگر یک سکه 5 تومانی خرج کنم، جواب مساله برابر F(N – 5) به علاوه یک سکه خرج شده خواهد بود.
حالت سوم بیانگر این است که اگر یک سکه 10 تومانی خرج کنم، جواب مساله برابر F(N – 10) به علاوه یک سکه خرج شده خواهد بود.
حالت چهارم بیانگر این است که اگر یک سکه 25 تومانی خرج کنم، جواب مساله برابر F(N – 25) به علاوه یک سکه خرج شده خواهد بود.

و در نهایت جواب بهینه برابر این است که انتخاب کدام شاخه برای من سودمند تر است.

شروط پایه: اگر N برابر 0 باشد، هیچ سکه‌ای نیاز نیست داده شود بنابراین جواب برابر 0 خواهد بود.

اگر N کوچکتر از 0 باشد یعنی شاخه فعلی غیر ممکن است پس هیچگاه نباید انتخاب شود. در این حالت ∞ را برمی‌گردانیم که در مینیمم-گیری هیچگاه شاخه انتخاب شده نباشد.

کد داینامیک:

http://paste.ubuntu.com/24438685/

تمرین: اگر به جای 4 نوع سکه، K نوع سکه (که ارزشهاشون توی یک آرایه داده می‌شد) داشتیم راه حل چگونه می‌شد؟

مثال چهارم (کوله پشتی صفر یک):

دزدی را در نظر بگیرید که قصد دزدی از یک مغازه را دارد. این دزد به همراه خودش یک کوله‌پشتی دارد تا اشیا را درون آن بگذارد. این کوله-پشتی نهایتا می‌تواند W کیلوگرم وزن را درون خودش تحمل کند. هر کالا موجود در مغازه یک وزن (weight) و یک ارزش (value) دارد. برنامه‌ای بنویسید که مشخص کند اگر دزد هوشمندانه عمل کند، مجموع ارزش اشیا موجود در کوله‌پشتی او چقدر خواهد بود؟

استخراج رابطه بازگشتی: دزد نسبت به هر شی دو مدل رفتار می‌تواند نشان دهد. یا آن شی را در کوله‌پشتی قرار می‌دهد، یا آن را رها می‌کند.

تابع F(Rem, N) را بدین شکل تعریف می‌کنیم:

مقدار Rem کیلوگرم از ظرفیت کوله‌پشتی باقی مانده و دارم در مورد کالای شماره N تصمیم می‌گیرم.

رابطه بازگشتی در این مثال به شکل زیر خواهد بود:

$$F(Rem, N) = \max\begin{cases} F(Rem - weight[N], N-1) + value[N] \\ F(Rem, N-1) \end{cases}$$

در رابطه بازگشتی فوق شاخه اول بیانگر این است که اگر کالای N دزدیده شود، مقدار weight[N] از ظرفیت کوله‌پشتی کم خواهد شد و مساله را برای کالای $N-1$ حل می‌کنیم. در نهایت به جواب مساله value[N] را اضافه می‌کنیم (زیرا کالا دزدیده شده است)

شاخه دوم نیز بیانگر این است که کالای N دزدیده نشود. ظرفیت باقیمانده کوله‌پشتی تغییری نمی‌کند و سودی هم کسب نمی‌کنیم.

شروط پایه:

در صورتی که Rem کوچکتر از 0 شود یعنی به حالت غیرممکن رسیده‌ایم، برای اینکه این حالت هیچگاه انتخاب نشود، مقدار ارزش $-\infty$ برای این حالت در نظر می‌گیریم تا در هنگام ماکزیمم‌گیری حذف شود. در صورتی که N برابر 0 باشد یعنی در مورد تمامی اشیا تصمیم‌گیری شده است و دیگر سودی نمی‌توانیم کسب کنیم. برای همین مقدار 0 را برمی‌گردانیم.

کد داینامیک:

http://paste.ubuntu.com/24425993/

مثال پنجم (مساله بارکدها، سوال دوم از تکلیف چهارم):

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1662

ابتدا صورت سوال را از لینک فوق مطالعه نمایید.

استخراج رابطه بازگشتی: می‌دانیم که در هر ناحیه رنگیِ پیوسته تعداد خطوط عددی مانند R که در بازه [1, M] قرار دارد، می‌باشد. طبق اصل جمع می‌دانیم جواب ما برابر مجموع جواب‌ها به ازای قرار دادن حالت‌های مختلف R می‌باشد. برای درک واضح تر به رابطه بازگشتی زیر توجه کنید:

$$BC(n,k,m) = \sum_{R=1}^{m} BC(n-R, k-1, m)$$

در رابطه بازگشتی فوق، با قرار دادن R خط، این تعداد خط از n کاسته می‌شود. به علت همرنگی این R خط، یکی از تعداد k کاسته می‌شود و مقدار m نیز ثابت می‌ماند.

شروط پایه: در صورتی که n یا k اعدادی کوچکتر از 0 باشند، صفر حالت ممکن داریم (هیچ حالتی نداریم). بنابراین عدد 0 را برمی‌گردانیم.

اگر N مساوی صفر شده باشد:

- اگر K مساوی صفر باشد (دقیقا K ناحیه ساخته باشیم)، یک حالت منطقی داریم. عدد یک را برگردان.
- اگر K نامساوی صفر باشد (تعداد نواحی ساخته شده دقیقا K نیست)، پس این حالت غیر صحیح است و صفر را برگردان.

کد داینامیک:

http://paste.ubuntu.com/24426112/

تمرین ** : http://codeforces.com/contest/598/problem/E

پرینت کردن جواب

در بسیاری از مواقع در مسائل بهینه سازی، از ما خواسته می‌شود که علاوه بر مقدار بهینه شده، مجموعه عناصری که سازنده راه حل بهینه هستند را نیز چاپ کنیم. در این مواقع، یک آرایه موازی آرایه dp نگه میداریم که در هر حالت آن نگه می‌داریم:

"در این حالت، کدام شاخه (زیرحالت) ما را به جواب بهینه رساند؟"

مثال ششم (محاسبه طول بلندترین زیر دنباله مشترک) :

برای اینکه بهتر متوجه سوال بشوید به این مثال توجه کنید.

بزرگترین زیر دنباله صعودی بین دو رشته A و B را بدست آورید.

A = “aedfhr”

B = “abcdgh”

LCS = “adh”

می‌دانیم که برای رسیدن از A و B به LCS، باید تعدادی از کاراکترهای این دو رشته را حذف کنیم.

استخراج رابطه بازگشتی:

تعریف می‌کنیم dp[N][M] برابر طول LCS دو رشته A[1...N] و B[1...M] باشد.

حل:

اگر دو کاراکتر A[N] و B[M] یکسان بودند، جواب برابر با dp[N – 1][M – 1] + 1 خواهد بود.

اگر دو کاراکتر A[N] و B[M] یکسان نبودند، بررسی کن به ازای حذف شدن کدامشان به زیردنباله بهتری میرسی.

رابطه بازگشتی:

$$LCS(i,j) = \begin{cases} LCS(i-1,j-1)+1 & A[i]=B[j] \\ \max\begin{cases} LCS\ (i-1,j) \\ LCS(i,j-1) \end{cases} & A[i]\neq B[j] \end{cases}$$

در رابطه بازگشتی فوق LCS(i – 1, j) همانند حذف کاراکتر A[i] از رشته A می‌باشد و LCS(i, j – 1) همانند حذف کاراکتر B[j] از B می-باشد.

شرط پایه: اگر هرکدام از اندیسهای متحرک روی A یا B به 0 رسیدند، امکان match کردن هیچ کاراکتری وجود ندارد و صفر برگردان.

سوال: شاید در موقعی نیاز باشد، هم کاراکتر A[i] حذف شود، هم کاراکتر B[j] حذف شود، این راه حل فعلی غلط نیست؟

پاسخ: راه حل ارائه شده، آن حالت را هم بررسی می‌کند به طوری که اول یک واحد i را عقب می‌برد سپس در فراخوانی بعدی یک واحد j را عقب می‌برد.

کد داینامیک:

http://paste.ubuntu.com/24426511/

تمرین: رشته LCS را به کمک روش توضیح داده شده در بالا، چاپ کنید.

تمرین: http://codeforces.com/contest/682/problem/D

تمرین ❋: http://www.spoj.com/problems/ADFRUITS/en

تمرین: http://www.spoj.com/problems/AIBOHP/en

مثال هفتم (محاسبه حالت بهینه ضرب زنجیره‌ای ماتریس‌ها):

دنباله‌ای از ماتریس‌ها به شما داده شده است که در هم ضرب می‌شوند. به چه ترتیبی ضرب را انجام دهیم تا هزینه ضرب کل کمینه شود؟

در واقع در اینجا مثال از شما، انتظار دارد بگویید اگر پرانتز گذاری این ماتریس‌ها به بهترین نحو انجام شود، هزینه ضرب کل این دنباله چقدر خواهد بود.

می‌دانیم که هزینه این ضرب A_{R*L} x B_{L*C} برابر RLC خواهد بود. (فرض کنید توالی داده شده از ماتریس‌ها ضرب پذیر هست بدین معنا که مولفه دوم هر ماتریس، برابر مولفه اول ماتریس بعدی می‌باشد)

استخراج رابطه بازگشتی: اگر M دنباله ماتریس‌ها باشد. M[i] برابر ماتریس شماره i می‌باشد.

تعریف می‌کنیم dp[i][j] برابر کمترین هزینه مورد نیاز برای ضرب ماتریس i تا j باشد.

$$(M_i \dots M_j) \rightarrow (M_i \dots M_k) * (M_{k+1} \dots M_j)$$

از روی عبارت فوق می‌توان رابطه بازگشتی زیر را استنتاج کرد:

$$MCM(i,j) = \min_{i \le k < j} (MCM(i,k) + MCM(k+1,j) + X)$$

که X برابر هزینه ضرب دو ماتریس حاصل (از پرانتزگذاری) می‌باشد.

شروط پایه: اگر i و j برابر بودند، هزینه ضرب برابر 0 خواهد بود (تنها یک ماتریس داریم)

اگر i برابر j – 1 بود، هزینه ضرب را طبق رابطه هزینه ضرب دو ماتریس حساب می‌کنیم.

تمرین: http://www.spoj.com/problems/MIXTURES

تمرین (مساله درخت باینری بهینه) :

https://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=1245

مثال هشتم: در یک گراف وزندار، دور همیلتونی با کمترین وزن را پیدا کنید. یا بیان کنید چنین دوری وجود ندارد.

برای حل این سوال ابتدا با نیاز داریم تا با مفهوم bitmask آشنا شویم.

:Bitmask

یک عدد باینری است که هر بیت آن بیانگر وجود یا عدم وجود یک عنصر از یک مجموعه در یک زیرمجموعه می‌باشد. به عنوان مثال اگر بیت شماره i در bitmask فعال باشد (1 باشد) بدین معنی است که در این زیرمجموعه عنصر شماره i موجود است. به مثال زیر دقت کنید:

فرض کنید مجموعه $A = \{x, y, z\}$ باشد. زیرمجموعه‌های آن عبارتند از:

شماره زیرمجموعه	زیرمجموعه	Bitmask
0	{}	000
1	{x}	001
2	{y}	010
3	{x, y}	011
4	{z}	100
5	{x, z}	101
6	{y, z}	110
7	{x, y, z}	111

وجود یک عنصر در bitmask را بدین شکل می‌توان متوجه شد:

if $(bitmask \ \& \ 2^i)$

در صورتی که شرط فوق صحیح باشد، عنصر شماره i در زیرمجموعه موجود است.

ادامه مثال شماره هشت:

تعریف می‌کنیم dp[bitmask][i] یعنی تا الان زیرمجموعه موجود در bitmask از رئوس را دیده‌ایم و در حال حاضر در راس شماره i هستیم. حال می‌توانیم به یکی از رئوسی که به راس فعلی متصل است و قبلا دیده نشده برویم.

استخراج رابطه بازگشتی:

اگر n تعداد رئوس باشد، bitmask یک عدد باینری n بیتی می‌باشد.

$$F(bitmask, i) = \min_{0 \le j < n} \begin{cases} \infty & if\ i\ and\ j\ are\ not\ connected\ or\ j\ is\ already\ visited \\ (F(bitmask \mid 2^j, j) + distance(i, j)) & Otherwise \end{cases}$$

جواب مساله: از آنجایی که به دنبال دور همیلتونی می‌گردیم، فرقی ندارد که از کدام راس شروع کنیم برای همین از راس 0 شروع می‌کنیم.

شرط پایه: اگر تمامی n بیت 1 شدند، یعنی تمامی رئوس را دیدیم، فاصله راس فعلی از راس شماره 0 را برمی‌گردانیم (به مبدا برمی‌گردیم).

تمرین:

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1852

تمرینات اضافه (ساده تر) :

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=52

http://codeforces.com/contest/699/problem/C

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1557

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1247

http://www.spoj.com/problems/SCUBADIV/

تمرینات اضافه (پیچیده تر) :

http://www.codah.club/tasks.php?show_task=5000000363

http://www.spoj.com/problems/SAMER08D/

http://www.codah.club/tasks.php?show_task=5000000355

در صورت وجود هرگونه مشکل یا سوال در این فایل، می‌توانید آن را به mohamadreza.m196@gmail.com ارسال نمایید.

با آرزوی موفقیت