

Time-memory trade-off

Динамично програмиране

Това е най-често същата наработка, която се използва при решаването на задачи като събиране и умножение на матрици, оптимизация на производствени процеси и т.н. Тя се използва за решаването на динамични задачи.

Така използваните задачи не решават всички задачи, но са много приложими.

В DAG не ни е дано DAG директно, а ние го идентифицираме чрез неговите върхове и построенияте им зависимости от построенияте им резултатни нодове за резултатни нодове. Тогава можем да създадем от DAG интерпретацията на A .

Този алгоритъм е една на АЛ
когато да се прилага във виду, като
най-рекурсивни решениe. Ако в
дървото на рекурсията има много
единични подзадачи, то тогава ще
имаме да се "запомни" решението
на дадена подзадача и следващия
и не, когато се намира да се решава
тази задача, вместо това да се
изчисляват резултатът на горната. Това
запомняне наричаме "меморизация".
"Всички условия за прилагане на
меморизация са рекурсивните
извиквания да ~~да~~ са комплексни
съобщения (да залежат само от
параметри и не са свързани при
един и същи извиквател или
използват един и същи резултат).
Така е случаи, че конкретни исти
са се решават дадена подзадача,
единствено се използва резултата,

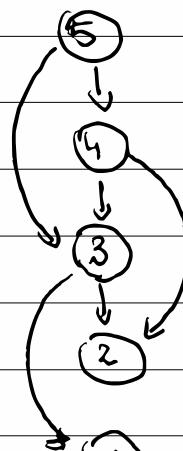
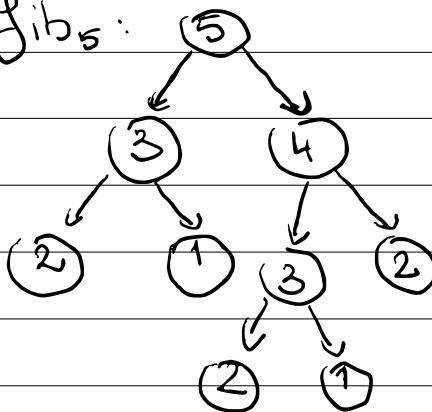
които е „запечатан“ в началото.

Пример: и-то се съмнява в правдата.

Но-лесното е неправдата brute force
решение да намерим и през
това да поклоним ДП.

$$\text{fib}_n = \begin{cases} \text{fib}_{n-1} + \text{fib}_{n-2}, & n > 2 \\ 1, & n = 1, 2 \end{cases}$$

fib_5 :



def fib5

Как и при всички рекурсивни алгоритми
8 рек. извиквания ги съвръщат
от 4 за F_5 . Така алгоритън с
бесконечно дълъг съмнява
го свидетелство на константата.

Серияе & непрервни линейни,
& логаритмични, че са от линейни
номер, за стапе константка T-V.
fib_n зависи непосредно от fib_{n-1}
fib_{n-2} за $n \geq 2$.

fibExp(n : natural number) : nat
num
if $n=1$ V $n=2$
return 1
else
return fibExp($n-1$) +
fibExp($n-2$)

$T(n) \asymp 2^n$, а $M(n) \asymp n$
извиквания в стека

Caso c Dn top-down:

```
fibExpMemo( n: natural number ): not num
    if dp[n] == -∞
        return dp[n]
    if n == 1 or n == 2
        return 1
    else
        dp[n] = fibExpMemo[n-1] +
            fibExpMemo[n-2]
        return dp[n].
T(n) ≈ n, a Θ(n) ≈ n.
```

Caso c Dn bottom-up.

```
fibInMemory( n: not num ): not num
```

dp[n]: array of ints

dp[1] ← 1, dp[2] ← 2

for i ← 3 to n do

dp[i] ← dp[i-1] + dp[i-2]

return dp[n]

$$T(n) \asymp n, \quad \Omega(n) \asymp n$$

fib bin ConstMemory (n : natnum): natnum

a, b : natnums

$a \leftarrow 1, b \leftarrow 1$

for $i \leftarrow 3$ to n do

$c \leftarrow a + b$

$a \leftarrow b$

$b \leftarrow c$

return b

$$T(n) \asymp n, \quad \Omega(n) \asymp 1$$

Може да се оптимизира
да прави само време $O(\text{loop})$,
да не извршише от $T(n)$ до n .

Будете тъжни, че ще видите как
рециркулата се състои от $T(n)$ и
н е редовото извршване в
което съществи от този път
рециркулата състои се от $T(n)$. Така
че можете да видите

единствено т.е. до него принадлежат
также например P_i, P_j , т.е. где
когда значение p_i отвечает за
решение первого типа, то это решения
второго например, от которых те
зависят (т.е. когда подразумевают
сочетание P_i и/или P_j).

Когда же, ее называют правильным
или корректным решением.

Следующее обобщение принципа
запоминается для того решения
"Ако за критерие на оптимизација
решение се назива оптимално
и е то засега ли и, то можем да
приложим ПП". Всички критери
говориха десет запомняни
само с един објектен.

Класически пример е запомняни
за критерии на вградб. Ако
они са критерии на отварјава
и т.е. вградб в този начин, то, то

значение, че Q_{max} не е
единствен и не е също то
ще има други Q_{max} не са Q_{min} ,
т.е. Q_{max} може да не е единствен
не от всички Q_{max} са Q_{min} .

Не можем да кажем, че
причина за оптималност
всички Q_{max} за конструиране
на кой-левел е в грата.



Най-добраят не е $Q \rightarrow b \rightarrow c$,
които най-добрият не е $Q \rightarrow b$ не
е $Q \rightarrow b$, а е $Q \rightarrow d \rightarrow c \rightarrow b$!

Като допълнителни изследвания
по темата вижте док на колегите
от Герасим и Чеви като и
лини за курс по ДАД в МИТ
2011 година, които е много добър.

Rod cutting

Какво ѝ се среща методът при
створяване на Ремингтон купуване
метални прътове, решетки и т.н.
и т.н. и продажба. Всеко разрезе е
деликатно. Мениджърът на Ремингтон
ища да знае какъв ѝ е срещу прътовете
наст-здрре. (решенска сум > 0)

Число топлини с $p[i:j], i=1, \dots, n$
които имат прът от i до j
които са нечакани.

Какъв ѝ е оптимален прът с топлини
и сум на топлини $p[1:n]$. Определяне
максимални пръти, които може да се
достигнат при определен съхраняване
на пръта и продажбата на източника.

* Ако $p[n]$ е достатъчно голям, то може
оптималното решение да не използва
задължително да решим този прът с
единния n .

Даден прв с делили и имае
избор на всеки см от n от 0 до n .

иа $1 \text{ см}, 2 \text{ см}, \dots, (n-1) \text{ см}$ за с
время или да не го времето:

Более 2^{n-1} различни начини за
изразяването на това.

Нека с $\Gamma[i], \dots, \Gamma[n]$ делени
масив, свързността ѝ с i е следваща
при оптимално изразяване трябва с
делили $1, \dots, n$ см с право уравнение
така че Γ .

Искаме да съзеди отговор на този
оптимален израз:

Търсим и да създадем делили и прв
да разрешат едно уравнение т.е.
последователност от n делили и да е
некомбинирана, т.е. да е оптимална
решение реше и преди да бъде построено
 $1 \leq k \leq n$, то оптималната

последователност е:

$k = i_1 + i_2 + \dots + i_n$, т.е.

делили i_1 е минимален и прв

$r[n] = p[i_1] + \dots + p[i_k]$ где
и i_1, i_2, \dots, i_k — это

За какое значение i_1 мы получим оптимальное значение $r[1]$? — $r[n]$ (если же это не так, то это $r[n]$ т.к. это наименее привлекательное значение для оптимального решения).
Оптимальное решение с изображением на $r[1]$ на оптимальное решение с изображением на $r[n]$, откуда $r[1] = r[n]$, и те же i_1, i_2, \dots, i_k — это решения наименее привлекательные для оптимального решения.

Решение в виде дерева решений

$$r[n] = \max (\underbrace{p[n]}, r[1] + r[n-1], \dots, r[n-1] + r[1])$$

При выполнении, что по (i.h.) known, что $r[1], \dots, r[n-1]$ — это оптимальные решения на $n-1$ элементах.

Мы можем это об-то засчитывать как оптимальные решения для $n-1$ элементов, а $p[n]$ — это значение для n -го элемента.

и оставим члены ~~за~~ оптимальными.

$$r[i] = \max_{1 \leq i \leq n} \{ p[i] + r[n-i] \}$$

Задача решается за $O(n^2)$ и не имеет
неких персонифицирующих.

Решение задачи на разрыве не приводит
ко правильному ответу:

Н разрывом с разрывом и некие первые
(левые) разрывы от него, когда есть
последующий разрыв от разрыва k в
очереди (дескать разрыв).

Некоторое brute-force решение
из этого разрыва и следующего
использует top-down + memoization и
с bottom-up + topsort именем
решения задачи:

$\text{cutRod}(p[1, \dots, n], n)$:

if $n=0$

return 0

currOpt $\leftarrow -\infty$

for $i \leftarrow 1$ to n do:

currOpt $\leftarrow \max(\text{currOpt}, p[i] + \text{cutRod}(p, n-i))$

return currOpt

Сложность $\leq 2^n$: $T(n) \asymp 2^n$.

$T(i)$ - дължина на резултатния отрезок. На cutRod е възможен
норматив на изчисленията i .

Рекурентна схема:
 $T(0) = 1$
 $T(n) = \Theta(1) + \sum_{i=1}^{n-1} T(i)$

Също така е експоненциален алгоритъм,
което са „reduce“-ни и са номинирани
съответно с trade-off отвън
назад.

Така наречи: top-down снизу вън.

Задълбочене в рекурсивната
логика като външна

Задача, которую делали рекурсивно
безе, это же с небольшими изменениями:

$\text{memoPutRod}(p[1, \dots, n], n)$:

$r[0, \dots, n]$: array of ints

for $i \leftarrow 0$ to n do

$r[i] \leftarrow -\infty$

return $\text{memoPutRodAux}(p[1, \dots, n], r[1, \dots, n])$

$\text{memoPutRodAux}(p[1, \dots, n], n, r[1, \dots, n])$:

if $r[n] \geq 0$

return $r[n]$

if $n = 0$

$q \leftarrow 0$

else

$q \leftarrow -\infty$

for $i \leftarrow 1$ to n do:

$q \leftarrow \max(q, p + \text{memoPutRodAux}(p, n-i, r))$

$r[n] \leftarrow q$

return q

За bottom up нутръде
 topological HQ разделя HQ на подзадачи
 в dependency DAG на звездата.
 В този случай е лесно, защото
 $r[n]$ зависи от всички
 $r[n-1], \dots, r[0]$ и зоната
 преди идтичните е reverse($n, n-1, \dots, 0$)

CutRod Iterative ($p[1, \dots, n], n$):

$r[0, \dots, n]$: array of ints

$r[0] \leftarrow 0$

for $j \leftarrow 1$ to n do

$q \leftarrow -\infty$

for $i \leftarrow 1$ to j do

$q \leftarrow \max(q, p[i] + r[j-i])$

$r[j] \leftarrow q$

return $r[n]$

И top-down и bottom-up ѝ са

време $\Theta(n^2)$ и

помислителни сложности $\Theta(n)$,

което показва, че времето не

този начин всички подразделения
са със своята своята идентичност.

Итеративната една е скрипту.

При итеративния
вариант има по-добър контекст,
запазен при рекурсивния чрез
"overhead" от рекурсивните
установки и в итеративната
се задават и пълният
контекст на константи и
переменни от по-горни редици.

При top-down няма DFS и
dependency graph не е зададен,
и при bottom-up няма
reverse topological sort и се
помага с това път назовава се.

