Trochu algoritmů

přednáška NPRG031 Programování 2 1. 4. 2020

Úloha na úvod

Začněme úlohou.

Editační vzdálenost

Editační vzdálenost dvou řetězců je číslo, které říká, kolik znaků musíme vymazat a napsat, abychom z prvního řetězce udělali druhý (nebo naopak, je to symetrické).

Tahle úloha je ekvivalentní s úlohou najít **nejdelší společný podřetězec.**
Pokud ty dva řetězce mají délky **M** a **N** a nejdelší společný podřetězec

(z každého řetězce nějaké znaky vynecháme
a to, co nám zbyde, se shoduje;
snažíme se jich vynechat co nejméně)

má délku **D**, potom editační vzdálenost je

M-D + N-D

(první sčítanec říká, kolik znaků musíme vymazat, druhý kolik musíme přidat).

Takže ta úloha:
zjistěte editační vzdálenost (nebo délku nejdelšího společného podřetězce, to je jedno) řetězců (jen ze znaků A, B, C)

AAABBCBBCBCACBBABBACACB
a **BBAAABBCBBACBACACBABBAACB.**

Kdybychom něco takového chtěli naprogramovat, můžeme ty řetězce procházet od začátku… ale záhy dojdeme k tomu, že když se jejich znaky nebudou shodovat, nevíme, zda by bylo lepší vynechat znak z prvního řetězce nebo z druhého (případně z obou). A zkusit obě možnosti není dobrý nápad, protože za chvíli zase nebudeme vědět a už na nás číhá exponenciální složitost.

Raději zkusme nějakou jinou úlohu a k téhle se vrátíme později.

Rozdělení dětí do skupin

Takhle úloha byla před časem v přijímacích zkouškách na matfyz:

Zjistěte, kolika způsoby se může šest dětí rozdělit do tří skupin.

Přitom:

- Skupiny nemají žádný název nebo vlajku (nebo šály), jde jenom o to, kdo s kým bude spolupracovat.
- Žádná skupina nemůže být prázdná.
- Každé dítě patří právě do jedné skupiny.

Většina uchazečů tuhle úlohu řešila rozborem případů podle toho, kolik členů budou mít jednotlivé skupiny, protože možnosti jsou jenom 1-1-4, 1-2-3 a 2-2-2:

1-2-3: 6*(5 nad 2) = 60

1-1-4: 6*5/2! = 15
děleno 2!, protože ty dvě vybírané jedničky jsou zaměnitelné

2-2-2: (6 nad 2)*(4 nad 2) /3! = 15
děleno 3!, protože ty tři vybírané dvojice jsou zaměnitelné

Dohromady 90 možností, ale dost příležitostí udělat nějakou chybu. Ukážeme si lepší způsob, ale nejdříve ještě snadnější úlohu.

Mrkev a petržel

Tahle úloha také byla v přijímacích zkouškách.
Máme za úkol spočítat, kolika způsoby lze osít deset záhonků, když

- na každém záhonku bude buďto mrkev nebo petržel
- nikdy nesmí být vedle sebe dva záhonky s petrželem

Uchazeči tuhle úlohu zase řešili rozborem případů, podle toho, na kolika záhoncích bude petržel: nejméně 0 a nejvíce 5 (nesmí být dva vedle sebe). Pro 0 existuje jediná možnost, pro 1 existuje 10 možností, pro 2… pro 2 už začíná přituhovat.

Tak si pojďme ukázat šikovnější způsob!

Zaprvé nebudeme tu úlohu řešit pro deset záhonků, ale vyřešíme ji úplně obecně, pro jakýkoliv kladný počet. Označíme si výslednou hodnotu třeba

$$\mathbf{MP_z}$$

A za druhé si hodnotu MP_z rozdělíme na součet dvou hodnot:

M_z bude počet možností kdy na posledním záhonku je mrkev
a P_z bude počet možností kdy na posledním záhonku je petržel.

A pro celkový počet možností bude platit

$$\mathbf{MP_z = M_z + P_z}$$

To, že jsme si úlohu zkomplikovali řešením pro jakýkoliv počet záhonků **z**, nám dává možnost začít hodnotou, pro kterou ty počty známe:

$$\mathbf{M_1 = 1} \text{ a } \mathbf{P_1 = 1}$$

A teď přijde to nejdůležitější a někdy také nejtěžší – odpovědět na otázku

Co když budeme mít větší počet záhonků?

Pokud nám přibyde jeden záhonek a bude na něm petržel, potom na vedlejším-předposledním záhonku **musí být mrkev!** Protože jinak bychom měli dva záhonky s petrželem vedle sebe a to se nesmí!

To ale znamená, že

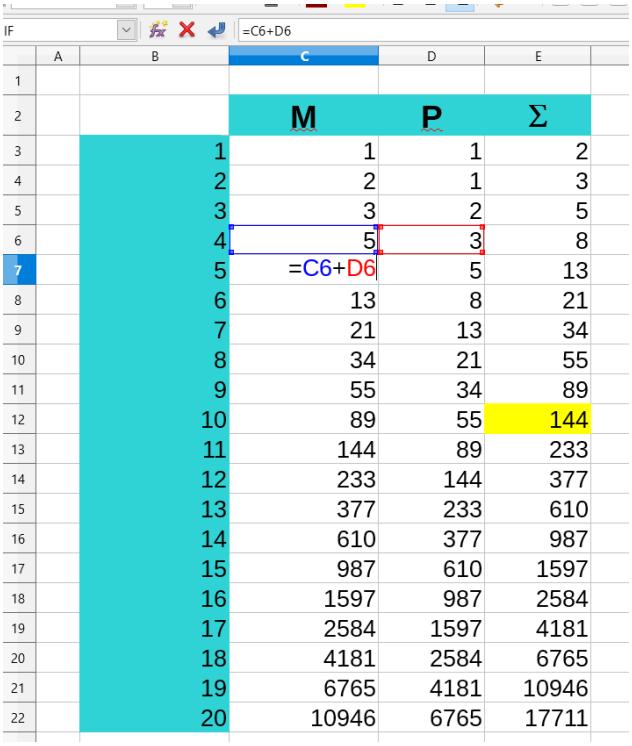
$$\mathbf{P_{z+1} = M_z}$$

Pokud jde o případ, že by na posledním záhonku byla mrkev, tak na předposledním může být cokoliv, neboli

$$\mathbf{M_{z+1} = M_z + P_z}$$

Teď ale už známe **počáteční hodnoty** a taky **rekurentní vzorec**, který nám dovoluje vypočítat hodnoty $\mathbf{M_z}$ a $\mathbf{P_z}$ pro jakékoliv z, stačí nám na to mechanicky sčítat a kopírovat a nemusíme se trápit s kombinatorikou ani s tím, že jsme někde zapomněli dělit počtem permutací.

Pokud máme k dispozici počítač, můžeme použít třeba nějaký tabulkový kalkulátor:



	M	P	Σ
1	1	1	2
2	2	1	3
3	3	2	5
4	5	3	8
5	=C6+D6	5	13
6	13	8	21
7	21	13	34
8	34	21	55
9	55	34	89
10	89	55	144
11	144	89	233
12	233	144	377
13	377	233	610
14	610	377	987
15	987	610	1597
16	1597	987	2584
17	2584	1597	4181
18	4181	2584	6765
19	6765	4181	10946
20	10946	6765	17711

Když nám ujede ruka s myší, může se stát, že to spočítáme ne pro deset záhonků, ale pro dvacet. Nebo pro padesát, nebo pro tisíc, každopádně to dokážeme vyřešit mechanicky a s lineární složitostí.

Už jste si nejspíše všimli, že ty hodnoty tvoří Fibonacciho posloupnost, ale to není důležité, důležité je, že máme jakýsi obecný postup:
nejdříve úlohu **zobecnit tak, aby měla nějaký parametr** udávající velikost, **určit počáteční hodnoty** a potom najít způsob (nemusí to být vždycky vzoreček), který spočítá nové hodnoty z těch, které už známe, neboli **určí řešení větší úlohy z řešení menších úloh**.

Zpátky na hřiště!

Vraťme se nyní k úloze o rozdělení dětí do skupin a konkrétní úlohu, pro šest dětí a tři skupiny, zobecníme. Počet způsobů, kterými lze rozdělit D dětí do S skupin si označíme $\mathbf{P_{D,S}}$.

Hodnoty, které známe, budou

$$\mathbf{P_{D,1} = 1}$$

jakýkoliv počet dětí do jedné skupiny

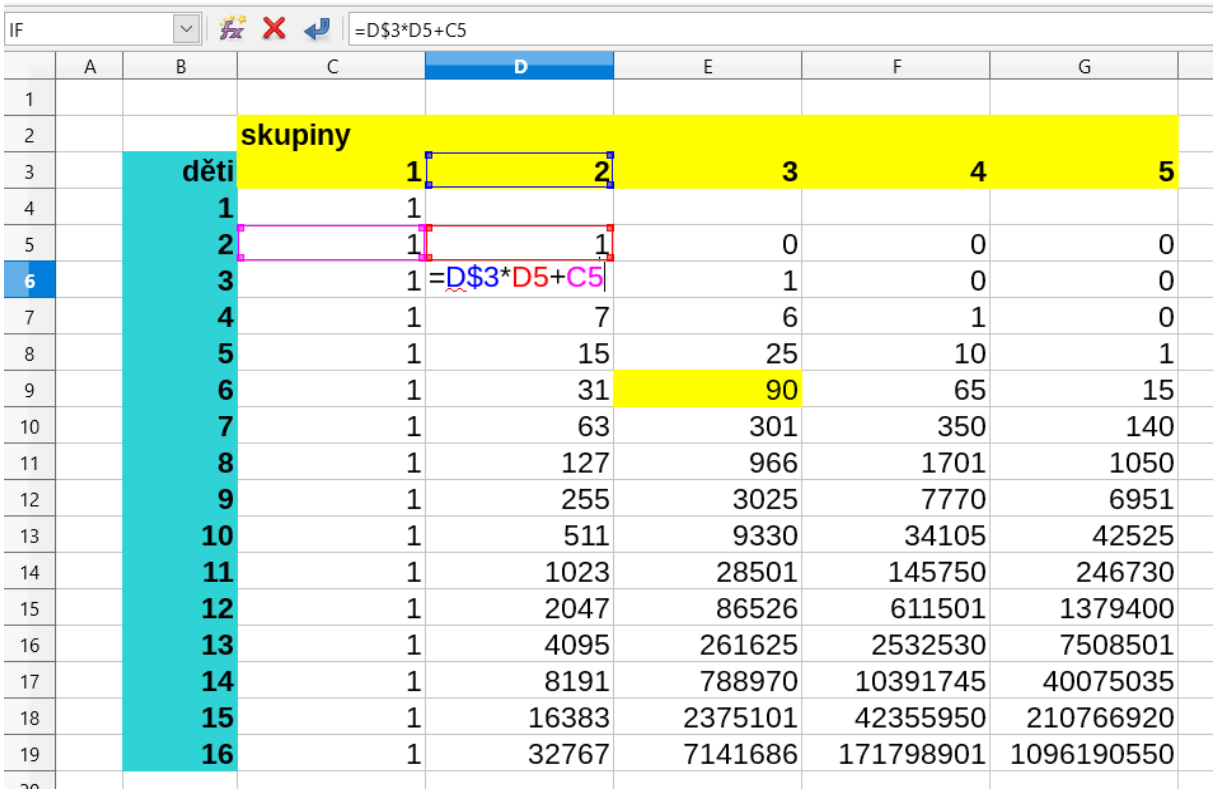
...a myslím, že s tím už vystačíme.

Co se stane, když nějaké dítě, přijde na hřiště, kde si už děti hrají, ve skupinkách? Buďto se může přidat do některé ze skupinek, které tam už jsou (a může si vybrat do které) nebo si vytvoří vlastní skupinu (a ostatní děti mají na výběr o jednu skupinu méně)

Neboli

$$P_{D,S} = S*P_{D-1,S} + P_{D-1,S-1}$$

Když zase použijeme tabulkový kalkulátor (přepíšeme tam přesně tenhle vzoreček, jen si chvíli budeme hrát s klávesou F4, abychom správně nastavili dolary)...



	A	B	C	D	E	F	G
1							
2			skupiny				
3		děti	1	2	3	4	5
4		1	1				
5		2	1	1	0	0	0
6		3	1	=D$3*D5+C5	1	0	0
7		4	1	7	6	1	0
8		5	1	15	25	10	1
9		6	1	31	90	65	15
10		7	1	63	301	350	140
11		8	1	127	966	1701	1050
12		9	1	255	3025	7770	6951
13		10	1	511	9330	34105	42525
14		11	1	1023	28501	145750	246730
15		12	1	2047	86526	611501	1379400
16		13	1	4095	261625	2532530	7508501
17		14	1	8191	788970	10391745	40075035
18		15	1	16383	2375101	42355950	210766920
19		16	1	32767	7141686	171798901	1096190550

...získáme hodnoty $P_{D,S}$ pro libovolné počty dětí i skupin.
Vidíme zase, že spočítat výsledek pro o trochu větší úlohu není žádný problém, jen složitost je tentokrát kvadratická (vyplňujeme 2D tabulku a jednu buňku spočítáme v konstantním čase). Ale také vidíme, že spočtené hodnoty rostou rychle – tedy že kdyby nás napadlo hledat odpověď postupným procházením všech možností… tak to není dobrý nápad.

Zpátky k editační vzdálenosti

Teď už je asi jasné, jak řešit úvodní úlohu. Místo abychom hned zjišťovali nejdelší společný podřetězec celých řetězců, přidáme si k úloze dva parametry (dva proto, protože řetězce jsou dva) a hodnota

$$\mathbf{H_{i,j}}$$

bude udávat **délku nejdelšího společného podřetězce** začátků oněch dvou řetězců, konkrétně prvních **i** znaků z prvního řetězce a prvních **j** znaků z druhého řetězce.

Když je jeden z řetězců prázdný, tak nemají společného nic, čili

$$\mathbf{H_{i,0} = H_{0,j} = 0}$$

a zbývá se ptát, jak spočítat hodnotu $\mathbf{H_{i,j}}$ z hodnot předchozích.

Pokud se poslední znaky (tedy i-tý a j-tý) shodují, bude výsledná hodnota o jedničku větší než hodnota pro zkrácené řetězce, tedy

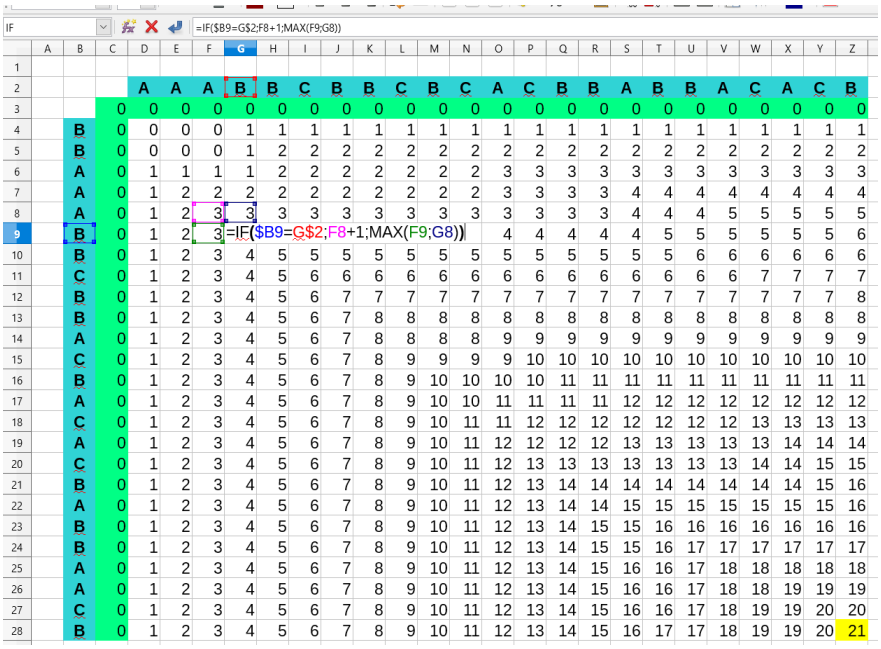
$$\mathbf{H_{i,j} = 1+H_{i-1,j-1}},\ \text{pokud}\ \text{první}_i = \text{druhý}_j$$

Pokud se poslední znaky neshodují, je (nejméně) jeden z nich zbytečný a můžeme ho zahodit. Protože nevíme který, vybereme si tu lepší z obou možností. Celkově tedy

$$\mathbf{H_{i,j}} = \mathbf{1+H_{i-1,j-1}},\ \text{pokud}\ \text{první}_i = \text{druhý}_j$$

$$= \mathbf{max(H_{i-1,j}\,;\,H_{i,j-1})},\ \text{jinak}$$

Můžeme opět použít tabulkový kalkulátor:



			A	A	A	B	B	C	B	B	C	B	C	A	C	B	B	A	B	B	A	C	A	C	B
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	B	0	0	0	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	A	0	1	1	1	1	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3
	A	0	1	2	2	2	2	2	2	2	2	2	2	3	3	3	3	4	4	4	4	4	4	4	4
	A	0	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	5	5	5	5	5
	B	0	1	2	3	=IF($B9=G$2;F8+1;MAX(F9;G8))								4	4	4	4	4	5	5	5	5	5	5	6
	B	0	1	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6
	C	0	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	7	7	7	7
	B	0	1	2	3	4	5	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	8
	B	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
	A	0	1	2	3	4	5	6	7	8	8	8	8	9	9	9	9	9	9	9	9	9	9	9	9
	C	0	1	2	3	4	5	6	7	8	9	9	9	9	10	10	10	10	10	10	10	10	10	10	10
	B	0	1	2	3	4	5	6	7	8	9	10	10	10	10	11	11	11	11	11	11	11	11	11	11
	A	0	1	2	3	4	5	6	7	8	9	10	10	11	11	11	11	12	12	12	12	12	12	12	12
	C	0	1	2	3	4	5	6	7	8	9	10	11	11	12	12	12	12	12	12	12	13	13	13	13
	A	0	1	2	3	4	5	6	7	8	9	10	11	12	12	12	12	13	13	13	13	14	14	14	14
	C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	13	13	13	13	13	14	14	15	15	
	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	14	14	14	14	14	14	15	16	
	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	14	15	15	15	15	15	15	15	16
	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	15	16	16	16	16	16	16	16
	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	15	16	17	17	17	17	17	17
	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	16	17	18	18	18	18	18
	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	16	17	18	18	19	19	19
	C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	16	17	18	19	19	20	20
	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	17	18	19	19	20	21

když změníme rětězec zapsaný v řádku nebo sloupci, tabulka se přepočítá.

Vyplnění jedné buňky potřebuje konstantní čas, takže časová složitost je součin délek řetězců. Pokud jde o paměťovou složitost, vystačili bychom se dvěma řádky (nebo sloupci), takže ta je úměrná délce delšího řetězce. Ale když chceme rekonstruovat onen nejdelší společná podřetězec, potřebujeme si pamatovat celou tabulku.

Dynamické programování

Popsané metodě se říká dynamické programování a dá se popsat tak, že skládáme řešení úlohy z řešení menších úloh stejného typu. Přitom musí platit, že to nalezené řešení (často to je něco nejlepšího, nejkratšího, nejdelšího) jako své části obsahuje (nejlepší) řešení svých pod-úloh.

Obvykle u toho potřebujeme nějakou paměť navíc a za to dostaneme polynomiální složitost místo exponenciální.

Rozděl a panuj

Trochu to připomíná metodu Rozděl a panuj, ale liší se v tom, že u Rozděl a panuj **víme, kde máme rozdělit**. Zatímco u dynamického programování to nevíme a tak zkoušíme všechny možnosti a z nich vybíráme nejlepší.

Memoizace

Vymyslet způsob jak na danou úlohu použít dynamické programování může být trochu obtížné. Někdy pomůže memoizace – rekurze s ukládáním výsledků.

Hra odebírání čísel

V minulém semestru jsem ukazoval hru, kde dva hráči postupně odebírají čísla ze společné řady a každý si vybírá, jestli si vezme číslo z levého nebo pravého okraje, s cílem získat co největší součet.

Program využívající algoritmus **negamax** vypadal takhle...

```
cisla = [24,18,36,19,2,9,45,7,4,2,4,5,6,8,18]
def OKolikSeDaVyhrat(od,do):
    if od>do:
        return 0
    L = cisla[od] - OKolikSeDaVyhrat(od+1,do)
    P = cisla[do] - OKolikSeDaVyhrat(od,do-1)
    return max(L,P)

print("-------- OKolikSeDaVyhrat(od,do): ---------")
print( OKolikSeDaVyhrat(0,len(cisla)-1) )
```

...a měl **exponenciální** časovou složitost, protože procházel všechny způsoby, jakými se dá tato hra hrát a dohrát a když má každý hráč na výběr ze dvou tahů, je těch způsobů celkem $\mathbf{2^{počet\ tahů}}$.

Jenomže!

Jenomže když se na tu funkci podíváme pořádně, tak vidíme, že má celkem dva parametry v rozsahu 0 až N-1 a jestli se něco volá exponenciálně-krát, tak se ta funkce určitě musí volat víckrát se stejnými parametry, protože těch možných kombinací parametrů, se kterými ji můžeme volat, je jenom $\mathbf{N^2!}$ (To není faktoriál, to je vykřičník.)

Takže když ten program doplníme o pole, kam si ta funkce bude **ukládat již vypočítané výsledky** (nejsnazší je použít dvou-rozměrné pole a indexovat přímo těmi parametry) a pokud ta funkce, než se pustí do počítání, nejdříve zkontroluje, jestli pro tyhle parametry už nemá uložený výsledek – pak bude pro každou kombinaci parametrů funkce počítat nejvýše jednou a **z časové složitosti 2^N budeme mít $N^2!$** (To je zase vykřičník.) Tomuto postupu se říká **memoizace.**

Od memoizace k dynamickému programování

Kdybychom mohli sledovat, jak se to pole bude postupně vyplňovat, viděli bychom, že se funkce volá tak dlouho, až rekurzí dojde k případům, na které zná odpověď. Když je řada prázdná, výsledek je nula. Tak ji napíše do pole a vrátí se. A příště na základě známé hodnoty pro tyhle jednoduché případy dokáže určit výslednou hodnotu složitějších případů. A tak dál. Neboli funkce, která si ukládá výsledky nakonec dělá přesně to, co bychom dělali, kdybychom tu úlohu řešili dynamickým programováním. Začíná od malých pod-úloh, kde je řešení snadné a z nich potom skládá řešení větších pod-úloh.

Takže náhradní řešení u problému, který vypadá exponenciálně a myslíme si, že by mohlo pomoci dynamické programování, je představit/**napsat si rekursivní řešení** (pravděpodobně s exponenciální složitostí) a vybavit ho ukládáním výsledků – **memoizací.**

Uzávorkování násobení matic

Pokud po vás u zkoušky budu chtít nějaký příklad na dynamické programování, **nezmiňujte Fibonancciho posloupnost!** (Jestli vás to ani nenapadlo, tak dobře.) Dynamické programování na Fibonacciho posloupnost je kanón na vrabce. Představme si místo toho jinou úlohu:

Máme řadu matic, které chceme vynásobit.

Násobení matic není komutativní, ale je asociativní, takže si můžeme vybrat, v jakém pořadí/uzávorkování ho budeme provádět. Záleží na tom? Výsledek násobení ne, ale rychlost výpočtu na tom záviset může.

Kdybychom měli jen tři matice s rozměry

5x3 3x7 7x4

, tak na násobení v pořadí **(5x3 3x7) 7x4** budeme potřebovat

5x3x7 + 5x7x4 = 105 + 140 = 245 násobení čísel,

zatímco na násobení v pořadí **5x3 (3x7 7x4)** nám bude stačit jen

3x7x4 + 5x3x4 = 84 + 60 = 144 násobení čísel. Úspora značná.

Takže zadání úlohy je následující

Známe rozměry jednotlivých matic a máme za úkol určit pořadí, v jakém se mají matice násobit.

Buďme konkrétní:
Zjistěte, kolik násobení je potřeba k tomu, abyste vynásobili matice rozměrů **5x7 7x2 2x4 4x3 3x6.**

Počet různých uzávorkování je exponenciální, chceme to řešit rychleji, celou dobu mluvíme o dynamickém programování…

Budeme postupně zjišťovat, kolik operací potřebujeme na vynásobení úseku matic (úsekem rozumím souvislou podmnožinu množiny 1..N). Začneme od těch nejkratších a výsledky si budeme zapisovat do tabulky.

	5x7	7x2	2x4	4x3	3x6
2	70	56	24	72	X
3					
4					
5					

číslo řádku značí délku úseku (kolik matic násobíme), nadpis sloupečku říká, od které matice ten úsek začíná.

Součin dvou matic můžeme spočítat jediným způsobem (a potřebujeme na něj tolik operací násobení, kolik je součin tří rozměrů těchto dvou matic, protože… ale určitě už máte za sebou lineární algebru).

U součinu tří matic už to je složitější, jak jsme viděli výše, takže si u každého úseku spočítáme počty násobení pro všechny možnosti jak úsek rozdělit na součin dvou částí – a z nich si vybereme a zapíšeme tu menší.

Až dopočítáme do konce, budeme mít v posledním řádku v prvním sloupci počet operací pro výpočet součinu úseku, který začíná první maticí a má délku rovnou počtu matic, neboli tu hodnotu, kterou jsme měli za úkol spočítat. Zkuste si ji teď spočíst sami a pokud vám nevyšlo **190**, tak něčemu nerozumíte.

Další úlohy

Úloh, při jejichž řešení se využije dynamické programování, je mnoho. Nejdelší cesta v acyklickém grafu, triangulace polygonu nebo optimální vyhledávací strom, když jsme u těch teoretických, ale také Viterbiho algoritmus nebo rychlá Fourierova transformace, pokud jde o ty praktické, podívejte se na ně sami.

Odpovědi na případné otázky budou zase na mé stránce: https://ksvi.mff.cuni.cz/~holan/py/otazky_200401.html , dotazy můžete posílat mailem nebo anonymně pomocí formuláře na zmíněné stránce. Pokud se chcete na něco zeptat živě, připojte se mezi 15:00-15:30 na Zoom: zde, Meeting ID: 381 064 257, Password: 981120.

Hodně zdaru!