**Součin matic**

Součin tří matic.

Mějme matice A ∈ **R***r*×*s* a B ∈ **R***s*×*t.*

Potom (A × B) ∈ **R***r*×*t* .

Sledujeme počet operací násobení dvou reálných čísel při výpočtu (A × B). P = *r∙ s∙ t.*

Mějme dále matici C ∈ **R***t*×*u.*

Potom A × B × C ∈ **R***r*×*u*.

Výpočet A × B × C může proběhnout dvěma způsoby: (A × B) × C nebo A × (B × C).

Výpočet závisí na uzávorkování.

První způsob (A × B) × C :

Počet operací násobení dvou reálných čísel je *r∙ s∙ t* pro A × B a *r∙ t∙ u* pro (A × B) × C,

celkem *r∙ s∙ t* + *r∙ t∙ u.*

Druhý způsob A × (B × C):

Počet operací násobení dvou reálných čísel je *s∙ t∙ u* pro B × C a *r∙ s∙ u* pro A × (B × C),

celkem *r∙ s∙ u* + *s∙ t∙ u.*

Například pro (*r*, *s*, *t*, *u*) = (2, 3, 4, 5) máme v prvním způsobu:

*r∙ s∙ t* + *r∙ t∙ u =*  2∙ 3∙ 4 + 2∙ 4∙ 5 = 24 + 40 = 64 operací násobení dvou reálných čísel.

v druhém způsobu

*r∙ s∙ u* + *s∙ t∙ u =*  2∙ 3∙ 5 + 3∙ 4∙ 5 = 30 + 60 = 90

Druhý způsob je o cca 40% časově náročnější než prvý.

Obecně, součin N matic.

Mějme matice A1, A2, ..., AN. Nechť matice Ai má rozměr ri−1 × ri.

Součin A1 × A2 × A3 × ... × AN má rozměr r0 × rN.

Chceme součin uzávorkovat tak, aby počet operací násobení dvou reálných čísel při výpočtu celého součinu byl minimální.

Princip dynamického programování: Zkoumejme N−1 součinů

A1 × (A2 × A3 × A4 × ... × AN−1 × AN)

(A1 × A2) × (A3 × A4 × ... × AN−1 × AN)

(A1 × A2 × A3) × (A4 × ... × AN−1 × AN)

. . .

(A1 × A2 × A3 × A4 × ... × AN−2) × (AN−1 × AN)

(A1 × A2 × A3 × A4 × ... × AN−2 × AN−1) × AN

Pokud známe optimální uzávorkování každé podtržené části a počet operací násobení v ní při tomto uzávorkování , můžeme pro každý uvedený řádek spočítat počet operací násobení a pak vybrat minimum.

Řádek odpovídající minimu pak představuje optimální uzávorkování celého součinu A1 × A2 × A3 × ... × AN .

Analogicky, pro nějaký souvislý úsek matic

AL × AL+1 × AL+2 × ... × AR, kde 1 ≤ L ≤ R ≤ N,

můžeme spočíst jeho uptimální uzávorkování tak, že uvážime součiny

AL × (AL+1 × AL+2 × AL+3 × ... × AR−1 × AR)

(AL × AL+1) × (AL+2 × AL+3 × ... × AR−1 × AR)

(AL × AL+1 × AL+2) × (AL+3 × ... × AR−1 × AR)

. . .

(AL × AL+1 × AL+2 × AL+3 × ... × AR−2) × (AR−1 × AR)

(AL × AL+1 × AL+2 × AL+3 × ... × AR−2 × AR−1) × AR

Pokud známe optimální uzávorkování každé podtržené části a počet operací násobení v ní při tomto uzávorkování , můžeme pro každý uvedený řádek spočítat počet operací násobení a pak vybrat minimum.

Řádek odpovídající minimu představuje optimální uzávorkování celého součinu AL × AL+1 × AL+2 × ... × AR.

Pro každou dvojici (L, R) indexů vymezující nejaký úsek matic uložíme do tabulky T na pozici (L, R) minimální počet násobení dvou reálných čísel při vhodném uzávorkování tohoto úseku matic.

Zřejmě pak platí

T(L, R) = min { T(L, K) + T(K+1, R ) + rL−1∙ rK∙ rR } , kde K probíhá od L do R−1.

Také zřejmě lze položit T(L, L) = 0 pro každé L = 1..N, protože v úseku obsahujícím jen jednu matici nic nenásobíme.

Prvek T(1, N) pak obsahuje minimální počet operací násobení při optimálním uzávorkování celého součinu A1 × A2 × A3 × ... × AN .

Asymptotická složitost

Složitost výpočtu prvku T(L, R) je úměrná velikosti rozdílu R−L. Složitost výpočtu celé tabulky je pak úměrná

součtu všech rozdílů R−L pro 1 ≤ L ≤ R ≤ N, což je hodnota asymptoticky úměrná třetí mocnině N. Přesné odvození tohoto faktu ponecháváme spanilomyslné čtenářce jako snadné cvičení.

=============================================================================

**Nejdelší rostoucí podposloupnost**

V posloupnosti reálných čísel a1, a2, ..., aN hledáme co nejdelší rostoucí podposloupnost. Její členy nemusí nutně v dané posloupnosti sousedit.

Předpokládejme, že známe nejdelší rostoucí podposloupnosti každé z uvedených kratších posloupností

a1

a1, a2

a1, a2, a3

a1, a2, a3, a4

...

a1, a2, a3, a4, ..., aN−2

a1, a2, a3, a4, ..., aN−2 × aN−1

Nejdelší rostoucí podposloupnost v každém řádku označme PK, 1 ≤ K ≤ N−1.

Vyzkoušíme pro každé K, jestli lze člen aN připojit na konec posloupnosti PK. Ze všech úspěšných pokusů, kdy člen aN připojit lze (je větší než všechny prvky PK) vybereme ten, v němž je posloupnost PK nejdelší možná, a tudíž i posloupnost PK  s připojeným prvkem aN bude co nejdelší možná.

Sestavíme tabulku T, ve které prvek T[k] představuje délku nejdelší rostoucí podposloupnosti, jejíž poslední prvek je aK.

Zřejmě tedy platí

T[k] = 1 + max{ |Pj|, aj < ak}, kde j probíhá od 1 do k−1. Maximální hodnotu prázdné množiny (prvek ak může být menší než všechny předchozí) považujme v tomto případě za 0.

Celkem samozřejmě inicializujeme tabulku T hodnotou T[1] = 1.

Abychom mohli nejdelší rostoucí podposloupnost nakonec rekonstruovat, zavedeme ještě jednu tabulku předchůdců Q. Nechť index m maximalizuje hodnoty max |Pj|, aj < ak, j =1 .. k−1, t.j. prvek am

je poslední v nejdelší rostoucí podposloupnosti, jejíž poslední prvek leží vlevo od ak. Pokud m existuje, položime Q[k] = m, jinak položíme Q[k] = 0.

Zpětně pak můžeme rekonstruovat indexy nejdelší rostoucí podposloupnosti jako r, Q[r], Q[Q[r]], ....,

kde r je index posledního členu nejdelší rostoucí podposloupnosti celé posloupnosti a1, a2, ..., aN.

Složitost

Výpočet T[k] i Q[k] je úměrný velikosti hodnoty k, protože je vždy třeba v konstantním čase zkontrolovat všechny hodnoty T[j], j < k. Celkem tedy je složitost úměrná součtu hodnot k pro k = 1, 2, ..., N, což pčedstavuje právě kvadratickou složitost celého algoritmu