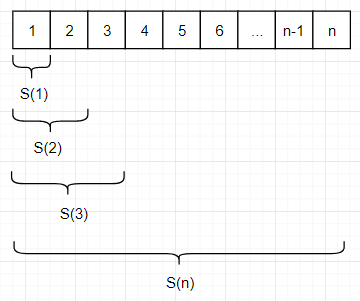
Programarea dinamică

Programarea dinamică este folosită pentru probleme de optimizare în care se efectuează un număr finit de alegeri pentru obținerea soluției optime. Problemele care urmează a fi rezolvate se împart în mai multe subprobleme de același tip.

Programarea dinamică este eficientă în momentul în care o anumită problemă dată poate fi compusă din rezultatele subproblemelor în care a fost împărțită, rezultate care pot forma o mulțime parțială de alegeri, care pot fi folosite în calcularea mai multor probleme viitoare. Tehnica principală folosită este aceea în care rezultatele problemelor care au fost rezolvate sunt stocate în vederea reutilizării acestora în momentul în care o anumită problemă apare din nou.

Programarea dinamică, rezolvă o anumită problemă combinând rezultatele subproblemelor în care a fost aceasta împărțită. Un algoritm bazat pe programarea dinamică rezolvă fiecare subproblemă o singură dată și, apoi, memorează soluția acesteia într-un tablou, abordare care ajută la evitarea recalculării unei anumite subprobleme ori de câte ori este întâlnită.

Putem evidenția tehnica programării dinamice prin următorul exemplu. Să presupunem că avem un vector cu n elemente, v[n], pentru care se dorește calcularea sumei primelor k elemente, unde k<=n.



Să presupunem că am calculat S(1), care reprezintă suma primului element:

* S(1) = v[1]. Stocăm rezultatul determinat pentru S(1), astfel încât acesta poate fi folosit în calcularea următoarelor probleme.

Pentru calcularea lui S(2), care reprezintă suma primelor 2 elemente putem aborda problema în 2 moduri:

* S(2) = v[1] + v[2]. Mod ineficient, întrucât putem să ne folosim de rezultatele determinate până acum.
* S(2) = S(1) + v[2]. Mod eficient, conform principiului programării dinamice, întrucât pentru a calcula S(2) am folosit unul din rezultatele determinate anterior, și anume rezultatul problemei determinării lui S(1).

În concluzie, se poate determina o formulă generală pentru calculul   
sumelor, și anume:

S(0) = 0

S(n) = S(n-1) + v[n].

În general, metoda programării dinamice se aplică problemelor de optimizare. Aceste probleme sunt caracterizate de faptul că pot avea mai multe soluții, fiecare soluție având câte o valoare, astfel încât se dorește determinarea unei soluții optime. O astfel de soluție se numește o soluție optimă, în contrast cu soluția optimă a problemei, deoarece pot fi mai multe soluții care realizează soluția optimă.

Dezvoltarea unui algoritm bazat pe programarea dinamică poate fi împărțită într-o secvență de 4 pași:

1. Se caracterizează structura unei soluții optime
2. Se definește recursiv valoarea unei soluții optime
3. Se calculează de jos în sus valoarea unei soluții optime
4. Din informațiile calculate se construiește de sus în jos o soluție optimă.

Pașii 1-3 sunt baza unei abordări de tipul programării dinamice, însă pasul 4 poate fi omis în momentul în care se dorește determinarea unei singure soluții optime.

În continuare vom evidenția tehnica programării dinamice printr-un exemplu mai complex, și anume, înmulțirea unui șir de matrice.

Ne propunem să calculăm produsul matriceal: M = M1M2...Mn. Întrucât înmulțirea matricelor este asociativă, putem efectua aceste înmulțiri în mai multe moduri. Înainte de a considera un exemplu, să observăm că înmulțirea clasică a unei matrice de (p, q) elemente cu o matrice de (q, r) elemente necesită p\*q\*r înmulțiri scalare.

Spre exemplu, dacă se dorește calcularea produsului matricelor A, B, C, D, unde A are (13, 5) elemente, B are (5, 89) elemente, C are (89, 3) elemente și D are (3, 34) elemente, în funcție de ordinea efectuării înmulțirilor matriceale (prin parantezare), numărul total de înmulțiri scalare poate să fie diferit, astfel:

(((AB)C)D) = 5785 + 3471 + 1326 = 10582 înmulțiri scalare, rezultat obținut astfel:

AB = (13, 5) \* (5, 89) = 13 \* 5 \* 89 = 5785 => AB = (13, 89) elem

(AB)C = (13, 89) \* (89, 3) = 13 \* 89 \* 3 = 3471 => (AB)C = (13, 3) elem

((AB)C)D = (13, 3) \* (3, 34) = 13 \* 3 \* 34 = 1326 => ((AB)C)D = (13, 34) elem.

((AB)(CD)) = 54201 înmulțiri scalare.

((A(BC))D) = 2856 înmulțiri scalare.

(A((BC)D)) = 4055 înmulțiri scalare.

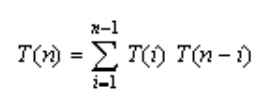
(A(B(CD))) = 26418 înmulțiri scalare.

Cea mai eficientă metodă este de aproape 19 ori mai rapidă decât cea mai ineficientă (54201/2856 = 18.97). Astfel, deducem faptul că ordinea de efectuare a înmulțirilor matriceale poate avea un impact dramatic asupra eficienței algoritmului realizat.

Pentru a determina care este ordinea optimă de efectuare a înmulțirilor matriceale, trebuie să parantezăm expresia lui M în toate modurile posibile și să calculăm de fiecare dată care este numărul de înmulțiri scalare necesare.

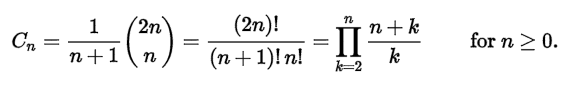
Notăm în continuare cu T(n) numărul de moduri în care se poate paranteza complet un produs de n matrice. Presupunem că facem prima „tăietură” între a i-a și a (i+1)-a matrice a produsului existent:

M = (M1M2...Mi)(Mi+1Mi+2...Mn), obținând astfel 2 membrii: membrul stâng al înmulțirii și membrul drept al înmulțirii. Pentru membrul stâng există acum T(i) moduri de a paranteza înmulțirea, iar pentru membrul drept există T(n-i) moduri de a paranteza. Astfel, pentru că i poate lua orice valoare între 1 și n-1, obținem recurența:



având T(1) = 1. Astfel, folosind recurența putem calcula toate valorile lui T(n). Spre exemplu:

T(5) = 14 = T(1)T(4) + T(2)T(3) + T(3)T(2) + T(4)T(1) = 1\*5 + 1\*2 + 2\*1 + 5\*1

 T(10) = 4862. Aceste valori pentru T(n) sunt cunoscute ca numerele catalane. Numerele catalane au următoarea formulă generală:

Numărul de soluții în care putem paranteza complet un produs de n matrice, T(n) = C(n) este (4n/n3/2), numărul fiind exponențial în n.

**Structura unei parantezări optime**

Prin convenție vom nota Mi..j matricea obținută în urma evaluării produsului (MiMi+1...Mj). O parantezarea optimă a produsului M1M2...Mn împarte produsul între Mk și Mk+1 pentru un număr k din intervalul 1, n. Pentru o anumită valoarea a lui k, calculăm mai întâi matricele M1..k și Mk+1..n și, apoi, le înmulțim pentru a produce rezultatul final M1..n. Costul total pentru această parantezare este, deci, costul calculului matricei M1..k, plus costul matricei Mk+1..n, plus costul înmulțirii celor două matrice rezultate.

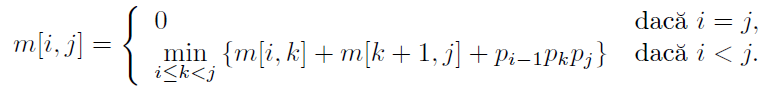
Observația foarte importantă este aceea că parantezarea subșirului M1M2...Mk în cadrul parantezării optime totale a prodului M1..n trebuie să fie optimă pentru M1..k. Acest lucru este datorat faptului că dacă ar fi existat o modalitate de parantezare mai puțin costisitoare a lui M1..k, înlocuirea respectivei parantezări în M1..n ar produce o altă parantezare al cărei cost ar fi mai mic decât costul optimului, ceea ce ar produce o contradicție, prin definiția conceptului de cost optim. O observație asemănătoare este valabilă și pentru parantezarea subșirului Mk+1..n care trebuie să fie optimă. Astfel, o soluție optimă a unei instanțe a problemei înmulțirii șirului de matrice conține soluții optime pentru instanțele subproblemelor.

**O soluție recursivă**

În cazul problemei înmulțirii șirului de matrice, o subproblemă constă în determinarea costului minim al unei parantezări a șirului MiMi+1...Mj, pentru 1<=i<=j<=n. Fie m[i,j] numărului minim de înmulțiri scalare necesare pentru a calcula matricea Mi..j; costul minim pentru calcului lui M1..n va fi m[1,n].

Putem defini m[i,j] în mod recursiv, în felul următor. Dacă i=j, șirul constă dintr-o singură matrice Mi..i = Mi și pentru a calcula produsul nu este necesară nicio înmulțire scalară, astfel m[i,i] = 0 pentru i = 1, 2, ..., n. Pentru a calcula m[i,j], pentru i<j, vom folosi o soluție optimă. Dacă descompunerea optimă în paranteze împarte produsul MiMi+1...Mj între Mk și Mk+1, cu i<=k<j, costum m[i,j] este egal cu costul minim pentru calculul subprodusului Mi..k și Mk+1..j, plus costul înmulțirii acestor două matrice rezultat. Pentru că evaluarea produsului Mi..kMk+1..j necesită pi-1pkpj înmulțiri scalare, vom obține:



 Ecuația prezentată anterior presupune cunoașterea lui k, lucru care nu este posibil. Există doar j-i valori pentru k, și anume: k=i, i+1,…,j-1. Pentru a determina parantezarea optimă trebuie să verificăm toate valorile m[i,j] pentru a o putea găsi pe cea mai bună. Astfel, definiția recursivă a costului minim pentru parantezarea produsului MiMi+1…Mj devine:

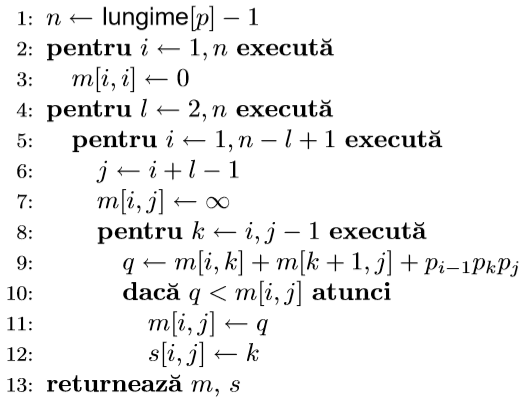
Valorile m[i,j] exprimă costul soluțiilor optime ale subproblemelor. Pentru a putea urmări modul de construcție a soluției optime, definim s[i,j] care va conține valoarea k pentru care împărțirea produsului MiMi+1...Mj produce o parantezare optimă, astfel încât s[i,j] = k pentru care:



**Calculul costului optimal**

Algoritmul prezentat în vederea obținerii soluției optime presupune că matricea Mi are dimensiunile pi-1 și pi pentru i = 1, 2, ..., n. Parametrul de intrare al algoritmului este secvența p=(p0,p1,p2,...,pn), unde lungime[p]=n+1. Algoritmul prezentat folosește un tablou auxiliar

m[1..n, 1..n] pentru costurile m[i,j] și un alt tablou s[1..n, 1..n] care stochează valoarea lui k pentru care s-a obținut costul optim al lui m[i,j].



Algoritmul prezentat anterior inițializează, mai întâi, m[i,i] cu valoarea 0, pentru i = 1, 2, ..., n, ceea ce reprezintă costul minim al șirurilor de lungime 1 (liniile 2, 3). Folosind recurența calculăm m[i,i+1] pentru i = 1, 2, ..., n-1, ceea ce reprezintă costul minim pentru șirurile de lungime 2, la prima execuție a ciclului din liniile 4-12. La următoarea iterație prin ciclu, se calculează m[i,i+2], pentru i = 1, 2, ..., n-2, adică costul pentru șirurile de lungime 3. În final, se observă faptul că la fiecare pas, calcularea lui m[i,j], cu i<=j, depinde de m[i,k] și m[k+1,j], care au fost anterior calculate.

Examinarea structurii de ciclu imbricat din algoritmul prezentat conduce la constatarea că timpul de execuție al algoritmului este O(n3). În fiecare dintre cele 3 cicluri imbricate, indicii lor (l, i și k) iau cel mult n valori. Astfel, timpul de execuție reprezintă un progres remarcabil față de metoda exponențială care verifică toate parantezările posibile.

**Un exemplu practic al algoritmului prezentat:**

p=(13, 5, 89, 3, 34), corespunzător dimensiunilor matricelor:

M1 (13, 5)

M2 (5, 89)

M3 (89, 3)

M4 (3, 34)

1. m[1,1] = 0

m[2,2] = 0

m[3,3] = 0

m[4,4] = 0

1. m[1,2] = m[1,1] + pqr = 0 + 13\*5\*89 = 5785; s[1,2] = 1

m[2,3] = m[2,2] + pqr = 0 + 5\*89\*3 = 1335; s[2,3] = 2

m[3,4] = m[3,3] + pqr = 0 + 89\*3\*34 = 9078; s[3,4] = 3

1. m[1,3] = min(m[1,1] + m[2,3] + 13\*5\*3,

m[1,2] + m[3,3] + 13\*89\*3)

= min(1530, 9256) = 1530; s[1,3] = 1

m[2,4] = min(m[2,2] + m[3,4] + 5\*89\*34,

m[2,3] + m[4,4] + 5\*3\*34

= min(24298, 1845) = 1845; s[2,4] = 3

1. m[1,4] = min( {k=1} m[1,1] + m[2,4] + 13\*5\*34,

{k=2} m[1,2] + m[3,4] + 13\*89\*34,

{k=3} m[1,3] + m[4,4] + 13\*3\*34)

= min(4055, 54201, 2856) = 2856; s[1,4] = 3

**Construirea unei soluții optime**

Algoritmul prezentat ne indică numărul optim de înmulțiri scalare necesare pentru a calcula produsul șirului de matrice, dar nu ne arată și cum trebuie efectuată această înmulțire. Pentru a determina modul în care vom efectua înmulțirea vom folosi tabloul s[1..n, 1..n]. Fiecare s[i,j] conține valoarea k pentru care se efectuează parantezarea optimă a produsului MiMi+1...Mj, adică între Mk și Mk+1. Produsul matricei M1..n se efectuează optim prin calculul: M1..s[1,n] și Ms[1,n]+1..n.

Concluzie: Înmulțirea realizată pentru exemplul de mai sus în care

p=(13, 5, 89, 3, 34) se reduce la o parantezare în funcție de s, după cum urmează:

* Calculul pleacă de la M1..4, pentru care,s[1,4] = 3 ne indică faptul că vom înmulți M1..3 cu M4..4.
* M1..3 pentru care, s[1,3] = 1 ne indică faptul că vom înmulți M1..1 cu M2..3.
* M2..3 pentru care, s[2,3] = 2 ne indică faptul că vom înmulți M2..2 cu M3..3.

Astfel, înmulțirea devine: ((M1,(M2,M3)),M4)