

Programare dinamică

BUZATU GIULIAN & NIȚĂ ALEXANDROS

Programare dinamică



Programare dinamică

Programarea dinamică este o tehnică de programare formată din 4 componente.

Acestea fiind:

1. Starea + definiția
2. Recurența
3. Inițializarea
4. Starea finală



Programare dinamică

Programarea dinamică este o tehnică de programare formată din 4 componente.

Acestea fiind:

1. Starea + definiția = parametrii funcției de dinamică
2. Recurența = tranziția între stări
3. Inițializarea = care sunt stările inițiale?
= ce valori trebuie să aibă?
= ce valori trebuie să aibă restul stărilor?
4. Starea finală = de unde calculăm răspunsul?

Tipuri de recurență

1. Recurență înainte = când ajungem la pasul i , el trebuie să fie deja calculat
 - altfel spus, la pasul i , calculăm pentru alți pași viitori
2. Recurență înapoi = când ajungem la pasul i , totul înaintea lui trebuie să fie deja calculat
 - altfel spus, la pasul i , îl calculăm pe el

Observație: Trebuie să existe o ordine a stărilor pentru a avea dinamică.

Complexitate

Complexitatea unei dinamici diferă în funcție de problemă, deoarece aceasta este doar o tehnică de programare, nu un algoritm care are o complexitate fixă.

După pasul al doilea, vom calcula complexitatea dinamicii noastre și, dacă este prea mare pentru limitele problemei, avem două posibilități:

- optimizăm recurența;
- reducem starea;

Memoizare vs. Tabulare

Einstein: Never memorize something you can look up
Person who invented Dynamic Programming:



Printre tehnicile folosite în programarea dinamică se numără:

1. Memoizarea

Implică crearea unei funcții ce împarte problema în subprobleme și reținerea rezultatelor pentru a nu rezolva aceste subprobleme de mai multe ori. Se implementează de obicei recursiv.

2. Tabularea

Este procedeul invers. Rezolvăm problemele mai mici, stocându-le într-o structură de date, pentru a obține problema mare. Se poate implementa în mod iterativ, ducând la programe mai rapide, deoarece nu se mai încarcă stiva din cauza apelurilor recursive.

Memoizare vs. Tabulare

```
std::vector<int> fibo(n + 1, -1);

int Fibonacci (int n, std::vector<int> & fibo) {
    if (fibo[n] != -1)
        return fibo[n];
    if (n == 1)
        fibo[n] = 0;
    else if (n == 2)
        fibo[n] = 1;
    else
        fibo[n] = Fibonacci (n - 1, fibo) + Fibonacci (n - 2, fibo);
    return fibo[n];
}
```

Exemplu memoizare

```
int Fibonacci (int n) {
    std::vector<int> Fibo(n + 1);
    Fibo[1] = 0, Fibo[2] = 1;
    for (int i = 3; i <= n; i += 1)
        Fibo[i] = Fibo[i - 1] + Fibo[i - 2];
    return Fibo[n];
}
```

Exemplu tabulare

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/culori3>

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/culori3>

Hint: Ce fel de recurență vrem să avem, înainte sau înapoi?

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/culori3>

Soluție: Preferăm să facem o dinamică înapoi, deoarece ne este mai ușor să calculăm posibilitățile pentru o anumită culoarea scândurii actuale, în funcție de scândura precedentă, decât să facem invers. Astfel, putem observa că anumite scânduri, pot urma doar după alte scânduri. Mai clar:

- alb vine doar după albastru;
- albastru vine după alb și roșu;
- roșu vine după albastru și verde;
- verde vine după roșu și galben;
- galben vine după verde;

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/culori3>

Soluție: Deci, la fiecare pas, vom calcula numărul de posibilități ca scândura actuală să ia o anumită culoare în funcție de posibilitățile de la scândura precedentă. La început fiecare culoare poate apărea pe prima scândură într-un singur mod, iar răspunsul va fi suma posibilităților ca ultima scândură să fie fie albă, fie albastră, etc.

Observație: Pentru 100 de puncte, trebuie să implementăm soluția folosindu-ne de numere mari.

Implementare: https://infoarena.ro/job_detail/2955768?action=view-source

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Hint 1: Cum putem împărți problema în subprobleme?

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Hint 1: Cum putem împărți problema în subprobleme?

Hint 2: Cum putem scrie matematic această recurență?

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Soluție: Să considerăm vectorul lis unde $lis[i]$ este lungimea celui mai lung subșir care se termină cu elementul de pe poziția i . Atunci, subșirul care se termină pe poziția i este format din subșiruri care se termină pe pozițiile $1, 2, \dots, i - 1$. De aceea vom alege maximumul dintre $lis[1], lis[2], \dots, lis[i - 1]$, cu proprietatea că elementul de la indexul i trebuie să fie mai mare decât ultimul element din subșir. Apoi, vom determina maximumul din vectorul de soluții. Matematic, recurența se scrie $lis[i] = \max_{j < i} (lis[j] + 1)$, cu condiția $a[j] < a[i]$.

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Soluție: Pentru reconstruirea șirului vom folosi un vector în care ținem minte indexul de unde am format soluția.

Complexitate: $O(n^2)$

Implementare: https://infoarena.ro/job_detail/3184375?action=view-source

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Cum putem îmbunătății complexitatea?

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Hint: Vrem ca ultimul element al unui potențial subșir crescător de lungime maximă să fie cât mai mic posibil.

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Soluție: Vom construi șirul d astfel încât $d[l]$ va reține cel mai mic număr cu care se poate termina un subșir de lungime l . Pentru a obține un subșir crescător de lungime l avem nevoie să găsim un subșir crescător de lungime $l - 1$. De aceea, trebuie, pentru fiecare element a din v să decidem unde îl punem în d . Îl vom pune înaintea celui mai mic număr mai mare decât el. Deoarece d reține capetele subșirurilor **crescătoare**, acesta este sortat, deci putem folosi căutarea binară pentru a obține indicele unde trebuie pus a . Pentru refacerea drumurilor vom ține minte la ce indice este fiecare element din d și predecesorul acestuia.

Problemă – SCLM (LIS)

Să rezolvăm problema: <https://infoarena.ro/problema/scmax>

Complexitate: $O(n \log n)$

Implementare: https://infoarena.ro/job_detail/3184408?action=view-source

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/100m>

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/100m>

Hint: Să încercăm să găsim o formulă pentru numărul de astfel de configurări pentru n atleți și k locuri obținute.

Problemă

Să rezolvăm problema: <https://infoarena.ro/problema/100m>

Soluție: Pentru a obține n atleți pe k poziții, putem avea $n - 1$ atleți aflați pe $k - 1$ poziții, caz în care putem pune ultimul atlet pe k poziții noi sau putem avea $n - 1$ atleți aflați pe k poziții, caz în care punem ultimul atlet în oricare din pozițiile date. Obținem astfel recurența $dp[n][k] = k \cdot dp[n - 1][k - 1] + k \cdot dp[n - 1][k]$. Vom folosi doar 2 linii din matricea dp pentru că nu avem suficientă memorie. La final va trebui să calculăm $\sum_{i=1}^n dp[n][i]$, pentru a considera toți atleții pe toate pozițiile disponibile.

Complexitate: $O(n^2)$

Implementare: https://infoarena.ro/job_detail/3184399?action=view-source

Temă

- <https://codeforces.com/contest/1829/problem/H>
- <https://infoarena.ro/problema/tairos>
- <https://infoarena.ro/problema/indep>
- <https://infoarena.ro/problema/lapte>

Probleme suplimentare

- <https://infoarena.ro/problema/kgraf>
- <https://infoarena.ro/problema/s2c>
- <https://infoarena.ro/problema/echipe>

Lectură suplimentară

- https://youtube.com/watch?v=oBt53YbR9Kk&ab_channel=freeCodeCamp.org
- https://youtube.com/watch?v=aPQY_2H3tE&ab_channel=Reducible