UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

ALGORITMI
ŞI
PROGRAMARE

Cursul 11

Paradigme de rezolvare a problemelor

Ionescu Vlad
vlad.ionescu@ubbcluj.ro

Paradigme

- Paradigmă = mulțime de concepte și șabloane pentru cum ar trebui făcut un lucru
- Vom analiza patru paradigme:
  - Greedy
  - Divide et Impera
  - Backtracking
  - Programare dinamică

Greedy

- Greedy = lacom
- Presupune alegerea optimului local la fiecare pas
- Avantaje:
  - De obicei este ușor de găsit și de implementat un algoritm greedy
  - Deseori duce la algoritmi eficienți
- Dezavantaje:
  - Uneori nu furnizează soluția optimă
  - Poate fi dificil de demonstrat corectitudinea

Greedy

- Problema restului pentru o sumă
  - Se dă o sumă S
  - Dorim să alegem un număr minim de bancnote astfel încât suma lor să fie exact S
  - Optim pentru tipuri reale de bancnote
  - Cum găsim un contraexemplu pentru alte tipuri de bancnote?
- Problema spectacolelor
  - Se dau **n** spectacole prin timpii lor de început $\mathbf{s_i}$ și de sfârșit $\mathbf{e_i}$
  - Care este numărul maxim de spectacole care pot fi planificate astfel încât să nu existe suprapuneri?

Divide et Impera

- “Dezbină și cucerește”
- Presupune împărțirea problemei în subprobleme distincte, rezolvarea acestora și combinarea rezultatelor pentru a obține rezolvarea problemei inițiale

Divide et Impera

- Căutarea binară
  - Căutarea unui element într-un șir ordonat se poate face mai eficient
  - Alegem mijlocul șirului: dacă elementul căutat este mai mic decât acesta, reducem căutarea la partea stângă, altfel o reducem la partea dreaptă.
  - Atenție la implementare – este ușor de greșit
- Quicksort
- Merge sort
- Exponențierea logaritmică

Backtracking

- Un algoritm backtracking va reveni la pași precedenți pentru a schimba decizii
- De obicei sunt folosiți pentru probleme de determinare a optimului (în lipsa unor algoritmi mai eficienți) și pentru probleme în care se cere enumerarea tuturor soluțiilor
- De cele mai multe ori sunt ineficienți
  - Dar nu întotdeauna, de exemplu: parcurgerea în adâncime, scanarea Graham
  - Ineficienți = complexitate exponențială

Backtracking

- Vom folosi doar implementări recursive
- Deși se pot implementa și iterativ, de cele mai multe ori acest lucru nu aduce beneficii, datorită complexității exponențiale
- Generarea permutărilor, aranjamentelor și combinărilor
- Atenție la transmiterea parametrilor: se transmit referințe!

Programare dinamică

- Se aplică problemelor care au:
  - **Substructură optimă**: dacă o soluție optimă poate fi obținută din soluții optime pentru subprobleme
  - **Subprobleme suprapuse**: dacă aceleași subprobleme sunt folosite de mai multe ori. Altfel spus, dacă o implementare recursivă directă s-ar autoapela de mai multe ori cu aceeași parametri

    Exemplu: fibonacci recursiv
- Folosită pentru probleme de optimizare și pentru probleme de numărare (a nu se confunda cu enumerare)

Programare dinamică

- Cel mai lung subșir crescător
- Combinări de n luate câte k
- Cel mai lung drum jos-dreapta într-o matrice
- Plata unei sume cu număr minim de bancnote
- Tehnica **memoizării**: abordări recursive care nu recalculează aceeași subproblemă de mai multe ori

Questions and Answers

Q & A