Algoritmy a Dátové Štruktúry

Jana Katreniaková
katreniakova@dcs.fmph.uniba.sk

Algoritmy - základné techniky

- Greedy algoritmy
- Rozdeľuj a panuj
- Dynamické programovanie
- Backtracking (nabudúce)

Greedy algoritmy

Myšlienka

- Snažíme sa nájst optimálnu možnost z viacerých konfigurácii alebo všeobecne možností
- Možnosti vieme nejako vhodne ohodnotit optimalizacnou funkciou

Kedy to môže fungovať?

- Bude fungovat ak vieme optimálnu konfiguráciu dosiahnut postupnostou lokálnych vylepšení
- Teda nie na všetko existuje greedy algoritmus (funkčný)

Greedy algoritmy

Myšlienka

- Snažíme sa nájst optimálnu možnost z viacerých konfigurácii alebo všeobecne možností
- Možnosti vieme nejako vhodne ohodnotit optimalizacnou funkciou

Kedy to môže fungovať?

- Bude fungovat ak vieme optimálnu konfiguráciu dosiahnut postupnostou lokálnych vylepšení
- Teda nie na všetko existuje greedy algoritmus (funkčný)

Greedy: Vydávanie mincí

Úloha

- Máme danú sadu mincí
- Chceme pre danú hodnotu ju zaplatiť/vydať pomocou minimálneho počtu mincí

Algoritmus

- Použijeme najväčšiu mincu čo vieme vydať.

Príklady

- 0.32 0.08 0.01
- 0.3 0.2 0.05 0.01 (skús vydat 0.40)

Greedy: Vydávanie mincí

Úloha

- Máme danú sadu mincí
- Chceme pre danú hodnotu ju zaplatiť/vydať pomocou minimálneho počtu mincí

Algoritmus

- Použijeme najväčšiu mincu čo vieme vydať.

Príklady

- 0.32 0.08 0.01
- 0.3 0.2 0.05 0.01 (skús vydat 0.40)

Greedy: Vydávanie mincí

Úloha

- Máme danú sadu mincí
- Chceme pre danú hodnotu ju zaplatiť/vydať pomocou minimálneho počtu mincí

Algoritmus

- Použijeme najväčšiu mincu čo vieme vydať.

Príklady

- 0.32 0.08 0.01
- 0.3 0.2 0.05 0.01 (skús vydat 0.40)

Greedy: Zlomkový problém batohu

Úloha

- Dané deliteľné predmety: váha w_i a cena b_i
- Chceme pre batoh danej veľkosti ho zaplniť čo najhodnotnejšími premetmi — optimalizujeme súčet hodnôt častí zobratých predmetov

Algoritmus

- Každý predmet má zlomkovú hodnotu – teda pomer b_i/w_i. Predmety si utriedime podľa tejto hodnoty.
- Berieme najhodnotnejší a koľko najviac z neho vieme zobrať (buď celý alebo po kapacitu batohu).

Príklady

- b_i: 12, 32, 40, 30, 50
- w_i: 4, 8, 2, 6, 1

Greedy: Zlomkový problém batohu

Úloha

- Dané deliteľné predmety: váha w_i a cena b_i
- Chceme pre batoh danej veľkosti ho zaplniť čo najhodnotnejšími premetmi — optimalizujeme súčet hodnôt častí zobratých predmetov

Algoritmus

- Každý predmet má zlomkovú hodnotu – teda pomer b_i/w_i. Predmety si utriedime podľa tejto hodnoty.
- Berieme najhodnotnejší a koľko najviac z neho vieme zobrať (buď celý alebo po kapacitu batohu).

Príklady

- b_i: 12, 32, 40, 30, 50
- w_i: 4, 8, 2, 6, 1

Greedy: Zlomkový problém batohu

Úloha

- Dané deliteľné predmety: váha w_i a cena b_i
- Chceme pre batoh danej veľkosti ho zaplniť čo najhodnotnejšími premetmi — optimalizujeme súčet hodnôt častí zobratých predmetov

Algoritmus

- Každý predmet má zlomkovú hodnotu – teda pomer b_i/w_i. Predmety si utriedime podľa tejto hodnoty.
- Berieme najhodnotnejší a koľko najviac z neho vieme zobrať (buď celý alebo po kapacitu batohu).

Príklady

- b_i: 12, 32, 40, 30, 50
- w_i: 4, 8, 2, 6, 1

Greedy: Snaživý Tomáš

Úloha

- Dané sú prednášky – začiatok, koniec
- Chceme stihnúť maximálny počet prednášok (ale celých)

Algoritmus

- Vyberieme si najskôr končiacu prednášku, ktorú môžeme stíhať

Príklad

- Pon 7:20-9:45, Ut 9:00-9:45, Pon 9:00-10:35

Greedy: Snaživý Tomáš

Úloha

- Dané sú prednášky – začiatok, koniec
- Chceme stihnúť maximálny počet prednášok (ale celých)

Algoritmus

- Vyberieme si najskôr končiacu prednášku, ktorú môžeme stíhať

Príklad

- Pon 7:20-9:45, Ut 9:00-9:45, Pon 9:00-10:35

Greedy: Snaživý Tomáš

Úloha

- Dané sú prednášky – začiatok, koniec
- Chceme stihnúť maximálny počet prednášok (ale celých)

Algoritmus

- Vyberieme si najskôr končiacu prednášku, ktorú môžeme stíhať

Príklad

- Pon 7:20-9:45, Ut 9:00-9:45, Pon 9:00-10:35

Rozdeľuj a panuj

Myšlienka

- Rekurzívne ak máme problém väcší ako k:
  - Rozdeľ na disjunktné podproblémy
  - Vyrieš podproblémy
  - Spoj riešenia podproblémov do výsledku

Kedy to môže fungovať

- Je možné ak vieme nájsť rozumné rozdelenie na rozumné disjunktné menšie podproblémy

Rozdeľuj a panuj

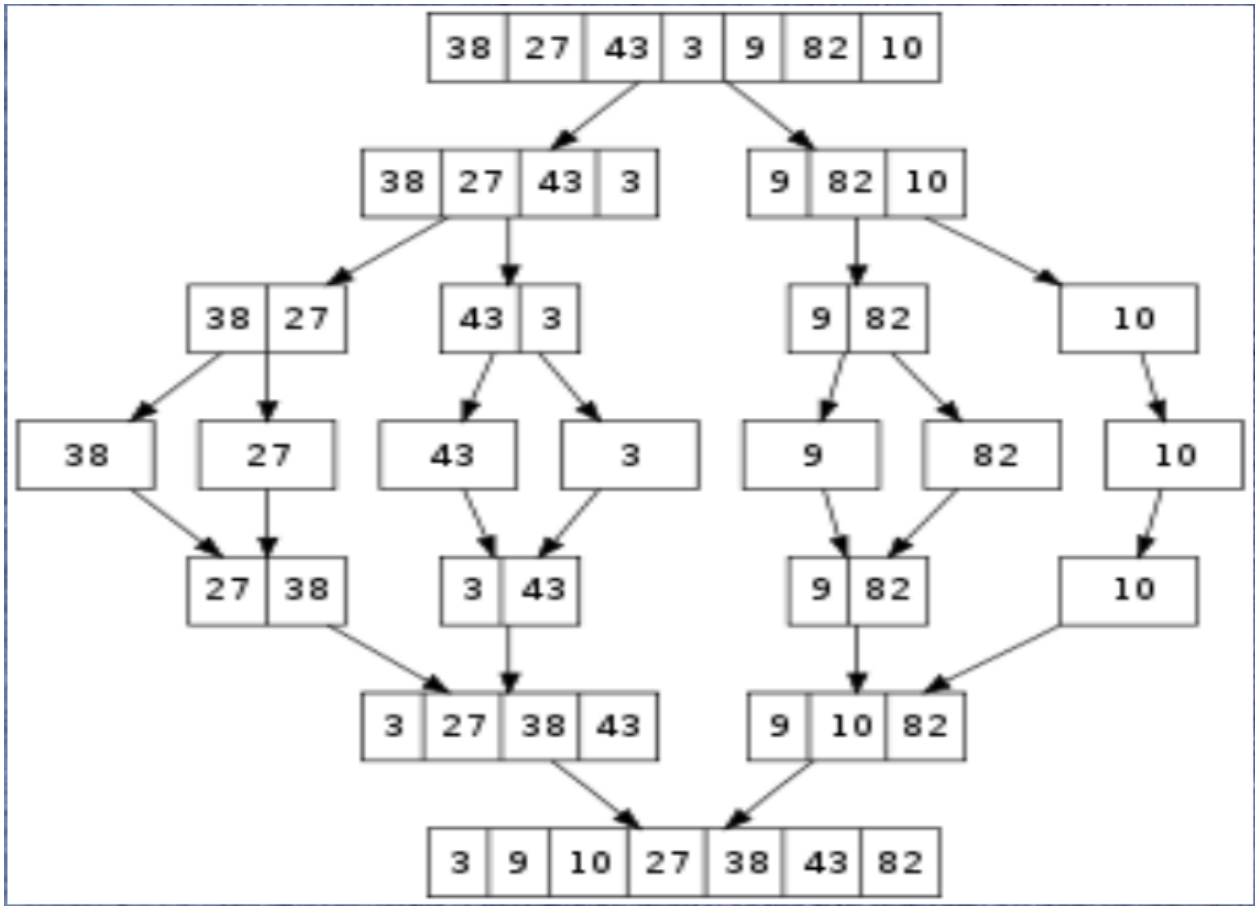
Myšlienka

- Rekurzívne ak máme problém väcší ako k:
  - Rozdeľ na disjunktné podproblémy
  - Vyrieš podproblémy
  - Spoj riešenia podproblémov do výsledku

Kedy to môže fungovať

- Je možné ak vieme nájsť rozumné rozdelenie na rozumné disjunktné menšie podproblémy

Rozdeľuj a panuj: Mergesort



Rozdeľuj a panuj: Násobenie čísel

Úloha

- Vynásobte dve dlhé čísla I a J

Algoritmus

- Rozdelím každé číslo na horné a dolné bity $I = I_H.2^{n/2} + I_L$ a $J = J_H.2^{n/2} + J_L$
- Násobením dostávam ... teda čas $T(n) = 4.T(n/2)$
- Ale: $I_H.J_L + I_L.J_H = (I_H - I_L).(J_L - J_H) + I_H.J_H + I_L.J_L$

Rozdeľuj a panuj: Násobenie čísel

Úloha

- Vynásobte dve dlhé čísla I a J

Algoritmus

- Rozdelím každé číslo na horné a dolné bity $I = I_H.2^{n/2} + I_L$ a $J = J_H.2^{n/2} + J_L$
- Násobením dostávam ... teda čas $T(n) = 4.T(n/2)$
- Ale: $I_H.J_L + I_L.J_H = (I_H - I_L).(J_L - J_H) + I_H.J_H + I_L.J_L$

Dynamické programovanie

Myšlienka

- Globálne optimálne riešenie sa dá dosiahnuť optimalizovaním podproblémov
- Riešime počínajúc najjednoduchšími a 'skladáme' z nich komplikovanejšie

Kedy to môže fungovať

- Jednoduché podproblémy - máme triviálne prípady ktoré sa dajú jednoducho riešiť
- Podproblémy nie sú nezávislé – prekrývajú sa (vieme ich skladať dokopy)

Dynamické programovanie

Myšlienka

- Globálne optimálne riešenie sa dá dosiahnuť optimalizovaním podproblémov
- Riešime počínajúc najjednoduchšími a 'skladáme' z nich komplikovanejšie

Kedy to môže fungovať

- Jednoduché podproblémy - máme triviálne prípady ktoré sa dajú jednoducho riešiť
- Podproblémy nie sú nezávislé – prekrývajú sa (vieme ich skladať dokopy)

Dynamické programovanie: Súvislá rastúca podpostupnosť

Úloha

- Máme konečnú postupnosť N čísel $x_1..x_N$.
- Chceme nájsť takú postupnosť $x_i..x_{i+k-1}$ že $x_j \leq x_{j+1}, \forall j \in i..i+k-2$ a z takýchto postupností je najdlhšia

Algoritmus

- $D(x_i)$ bude dĺžka najdlhšej rastúcej postupnosti končiacej v x_i.
- $D(x_1) = 1$
- $D(x_i) = D(x_{i-1}) + 1$ ak $x_i \geq x_{i+1}$ a v opačnom prípade $D(x_i) = 1$

Dynamické programovanie: Súvislá rastúca podpostupnosť

Úloha

- Máme konečnú postupnosť N čísel $x_1..x_N$.
- Chceme nájsť takú postupnosť $x_i..x_{i+k-1}$ že $x_j \leq x_{j+1}, \forall j \in i..i+k-2$ a z takýchto postupností je najdlhšia

Algoritmus

- $D(x_i)$ bude dĺžka najdlhšej rastúcej postupnosti končiacej v x_i.
- $D(x_1) = 1$
- $D(x_i) = D(x_{i-1}) + 1$ ak $x_i \geq x_{i+1}$ a v opačnom prípade $D(x_i) = 1$

Dynamické programovanie: Súvislá rastúca podpostupnosť

Úloha

- Je daných N matíc $A_1..A_N$ a ich veľkosti a chceme ich násobiť tak, aby sme vykonali minimum operácií

Príklad

- $A_1 : 3x100$ $A_2 : 100x5$ $A_3 : 5x5$
- $(A_1 x A_2) x A_3$ je 3.100.5 + 3.5.5 = 1575 operácií
- $A_1 x (A_2 x A_3)$ je 100.5.5 + 3.100.5 = 4000 operácií

Algoritmus

- $N_{i,j} = min_{i \leq k < j}\{N_{i,k} + N_{k+1,j} + d_i.d_{k+1}.d_{j+1}\}$

Dynamické programovanie: Súvislá rastúca podpostupnosť

Úloha

- Je daných N matíc $A_1..A_N$ a ich veľkosti a chceme ich násobiť tak, aby sme vykonali minimum operácií

Príklad

- $A_1 : 3x100$ $A_2 : 100x5$ $A_3 : 5x5$
- $(A_1 x A_2) x A_3$ je 3.100.5 + 3.5.5 = 1575 operácií
- $A_1 x (A_2 x A_3)$ je 100.5.5 + 3.100.5 = 4000 operácií

Algoritmus

- $N_{i,j} = min_{i \leq k < j}\{N_{i,k} + N_{k+1,j} + d_i.d_{k+1}.d_{j+1}\}$

Dynamické programovanie: Súvislá rastúca podpostupnosť

Úloha

- Je daných N matíc $A_1..A_N$ a ich veľkosti a chceme ich násobiť tak, aby sme vykonali minimum operácií

Príklad

- $A_1 : 3x100$ $A_2 : 100x5$ $A_3 : 5x5$
- $(A_1 x A_2) x A_3$ je 3.100.5 + 3.5.5 = 1575 operácií
- $A_1 x (A_2 x A_3)$ je 100.5.5 + 3.100.5 = 4000 operácií

Algoritmus

- $N_{i,j} = min_{i \leq k < j}\{N_{i,k} + N_{k+1,j} + d_i.d_{k+1}.d_{j+1}\}$

Dynamické programovanie: Knapsack problém

Úloha

- Podobne ako v zlomkovom probléme batohu, len nemôžeme predmety deliť

Príklad

- $A_1 : 3x100$ $A_2 : 100x5$ $A_3 : 5x5$
- $(A_1 x A_2) x A_3$ je 3.100.5 + 3.5.5 = 1575 operácií
- $A_1 x (A_2 x A_3)$ je 100.5.5 + 3.100.5 = 4000 operácií

Algoritmus

- $O_{i,j}$ najväčšia cena vecí v batohu veľkosti j, ak si vyberáme spomedzi vecí $1..i$
- $O_{i,j} = max(O_{k-1,i}, \{O_{k-1,i-s_k} \mid S_k \leq i\})$

Dynamické programovanie: Knapsack problém

Úloha

- Podobne ako v zlomkovom probléme batohu, len nemôžeme predmety deliť

Príklad

- $A_1 : 3x100$ $A_2 : 100x5$ $A_3 : 5x5$
- $(A_1 x A_2) x A_3$ je 3.100.5 + 3.5.5 = 1575 operácií
- $A_1 x (A_2 x A_3)$ je 100.5.5 + 3.100.5 = 4000 operácií

Algoritmus

- $O_{i,j}$ najväčšia cena vecí v batohu veľkosti j, ak si vyberáme spomedzi vecí $1..i$
- $O_{i,j} = max(O_{k-1,i}, \{O_{k-1,i-s_k} \mid s_k \leq i\})$

Dynamické programovanie: Knapsack problém

Úloha

- Podobne ako v zlomkovom probléme batohu, len nemôžeme predmety deliť

Príklad

- $A_1 : 3x100$ $A_2 : 100x5$ $A_3 : 5x5$
- $(A_1 x A_2) x A_3$ je 3.100.5 + 3.5.5 = 1575 operácií
- $A_1 x (A_2 x A_3)$ je 100.5.5 + 3.100.5 = 4000 operácií

Algoritmus

- $O_{i,j}$ najväčšia cena vecí v batohu veľkosti j, ak si vyberáme spomedzi vecí $1..i$
- $O_{i,j} = max(O_{k-1,i}, \{O_{k-1,i-S_k} \mid S_k \leq i\})$