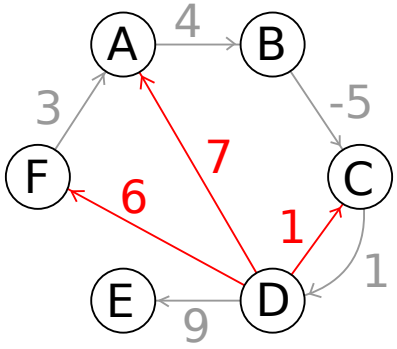
Dynamické programovanie

Truc Lam, Bui

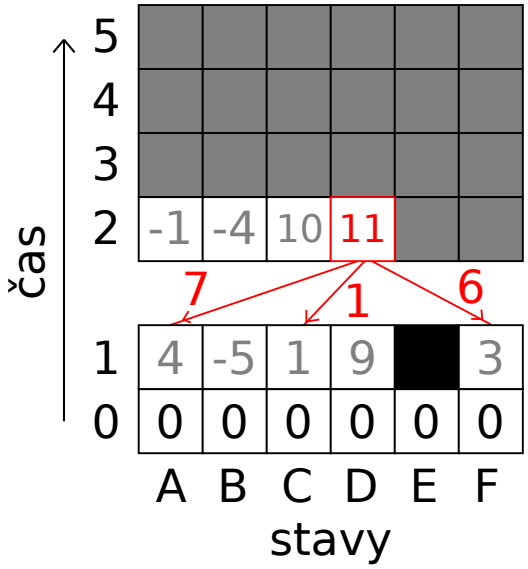
April 3, 2018

Motivačná úloha z matematickej optimalizácie.

Markovovské rozhodovacie procesy. Zadaný je markovovský rozhodovací proces vo forme orientovaného ohodnoteného grafu. Vrcholy grafu sú stavy, orientované hrany medzi vrcholmi sú akcie. Hrany sú ohodnotené tým, akú odmenu dostaneme, ak vykonáme príslušnú akciu. Úlohou je maximalizovať celkovú odmenu po $k = 1, 2, \ldots, K$ krokoch.



Príklad markovovského rozhodovacieho procesu.



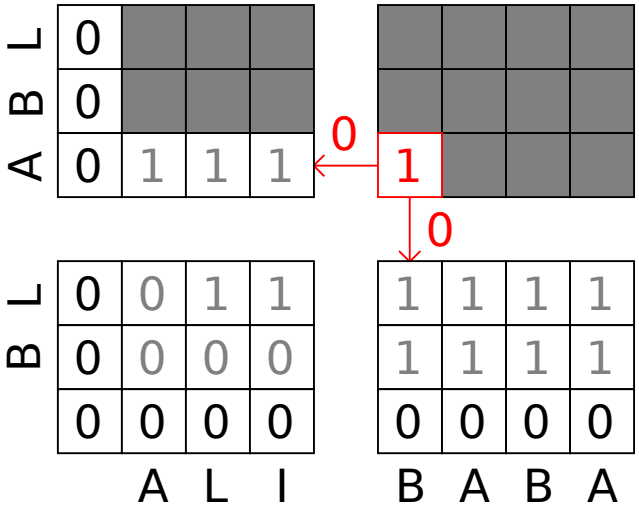
Tabuľka riešení. Postupne vypĺňame tabuľku zdola nahor.

Aká je časová zložitosť? Stavov v dynamickom programovaní (*problémov*) máme $n \cdot (K + 1)$, a v konkrétnom stave spravíme toľko operácii, koľko hrán vychádza z aktuálneho vrchola. Teda celková časová zložitosť je $O((n + m) \cdot K)$.

A motivačné úlohy z informatiky.

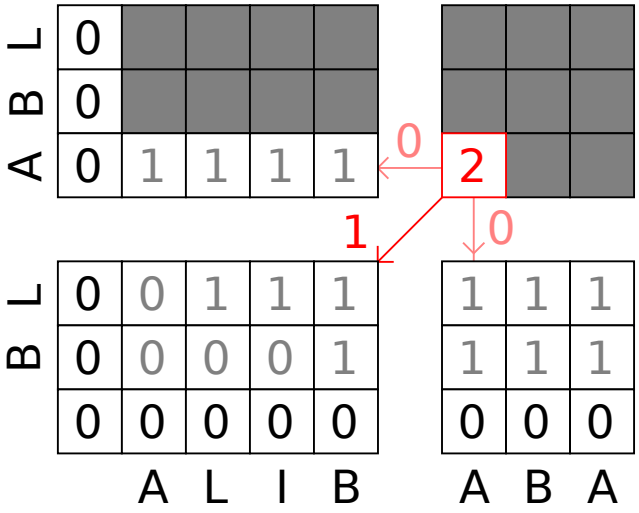
Najdlhšia spoločná podpostupnosť. Predstavme si, že hľadáme najdlhšiu spoločnú podpostupnosť dvoch postupností. Ak je aspoň jedna z nich prázdna, nie je čo riešiť. V opačnom prípade sa pozrieme na ich posledné prvky.

- Ak sú rôzne, tak aspoň jeden z týchto dvoch prvkov v podpostupnosti nebude. Máme teda dve možnosti podľa toho, z ktorej postupnosti sa rozhodneme ubrať posledný prvok. Vyberieme si tú lepšiu.



Tabuľku vypĺňame zdola nahor, zľava doprava. Príklad nezhody.

- Ak sú rovnaké, tak môžu byť v spoločnej podpostupnosti, a zrejme sa nám ich v nej oplatí mať. Potom riešime ten istý problém až na to, že obe postupnosti sú o jedna kratšie.

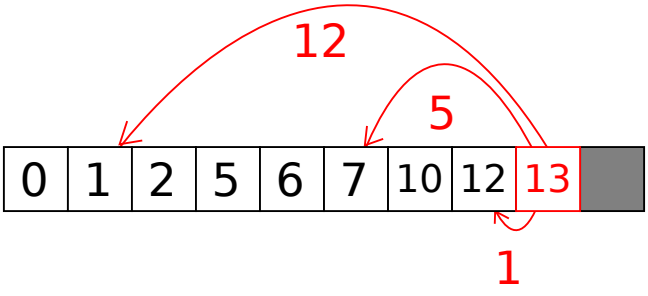


Príklad zhody. Vždy sa nám oplatí zobrať zhodné znaky.

Aká je časová zložitosť? Stavov máme $n \cdot m$, v každom spravíme konštantne veľa operácii. Časová zložitosť je teda $O(n \cdot m)$.

Knapsack. Máme n rôznych typov vecí. Každý typ má svoju celočíselnú hmotnosť a svoj úžitok. Chceme si na túru odniesť veci s hmotnosťou nanajvýš C tak, aby sme mali čo najväčší úžitok. Aký najväčší úžitok vieme dosiahnuť? Ktoré veci a v akom počte si máme zobrať?

Buď sa nám do batoha už nič nezmestí, a odpoveď je 0. Alebo sa nám do batoha ešte nejaká vec zmestí. Vyskúšame teda postupne všetky veci, ktoré by sa nám do batoha ešte zmestili. Keď skúšame do batoha vložiť vec s hmotnosťou w, nezávisle od jej úžitku nás zaujíma odpoveď na otázku: "Aký najväčší úžitok viem dosiahnuť s kapacitou $C - w$?"



Pažravá stratégia nie je vždy najlepšia.

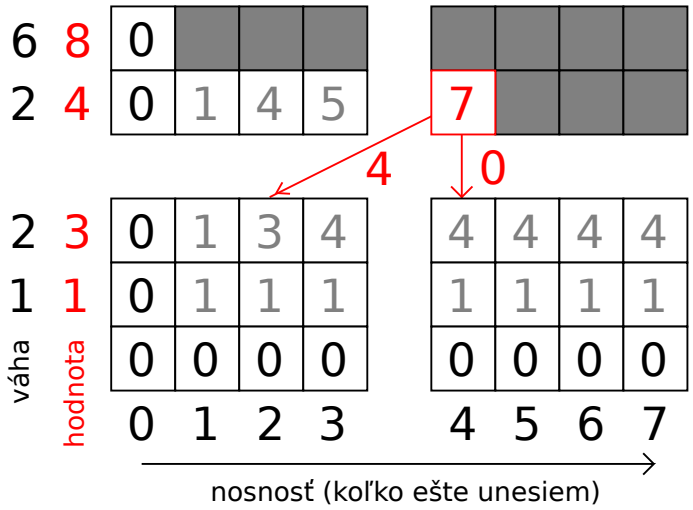
Aká je časová zložitosť? Stavov je $C + 1$, v každom z nich robíme n operácii. Časová zložitosť je preto $O(n \cdot C)$.

0-1 Knapsack. Rovnaké ako predchádzajúca úloha až na to, že z každého typu veci si môžeme zobrať nanajvýš jeden. Aký najväčší úžitok môžeme dosiahnuť s kapacitou C?

Predstavme si, že si veci usporiadame do radu. Pôjdeme sprava doľava, a pre každú vec sa rozhodneme, či ju na túru berieme alebo nie. Pri tom si v hlave pamätáme, akú najväčšiu hmotnosť sme ešte schopní so sebou zobrať.

V každom kroku sa teda pýtame otázku: "Môžem so sebou odniesť ešte veci hmotnosti nanajvýš C'. Práve stojím pri i-tej veci zľava. Aký najväčší úžitok viem dosiahnuť?"

V tomto momente sú len dve možnosti. Buď i-tu vec zoberiem, alebo nie (prípadne ak je ťažšia, ako zvládneme, tak máme len možnosť ju nezobrať). Vyberieme si teda tú lepšiu z nich.



Ani tu nie je pažravá stratégia vždy najlepšia.

Aká je časová zložitosť? Všetkých stavov je $(n + 1) \cdot (C + 1)$, v každom spravíme konštantne veľa operácii. Časová zložitosť je preto $O(n \cdot C)$.

Ukážme si tiež úlohy, kde naším cieľom **nie je** nájsť spomedzi viacerých možností tú najlepšiu.

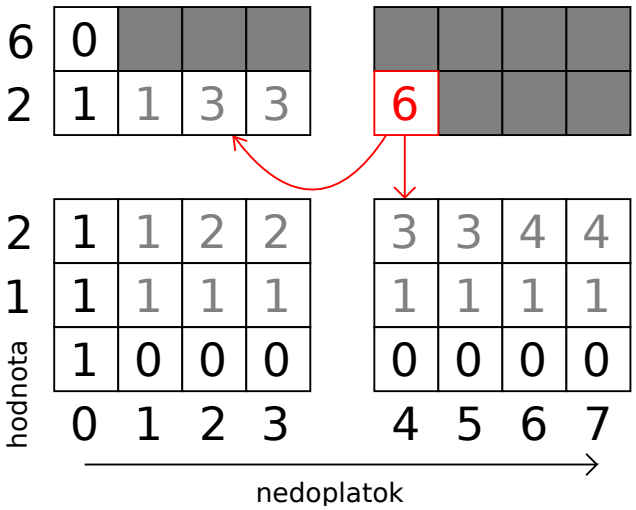
Mince. Máme k dispozícii n rôznych druhov mincí, každú v nekonečnom množstve. Každá minca má svoju celočíselnú hodnotu. Koľkými spôsobmi vieme zaplatiť sumu C? Nezáleží na poradí, v akom mince zaplatíme, iba na ich počtoch.

Predstavme si, že si mince usporiadame do radu. Postavme sa k poslednej minci, a spýtajme sa: "Chcem ešte ďalšiu mincu tohto typu, alebo nie?" Ak áno, zostaneme stáť pri tejto minci a znížime si náš nedoplatok o hodnotu mince. Ak nie, posunieme sa k ďalšej minci (doľava), pričom náš nedoplatok zostane rovnaký.

Pritom prvú možnosť môžeme použiť jedine v prípade, keď hodnota mince neprevyšuje náš nedoplatok—chceme totiž zaplatiť presnú sumu. Ak sa niekedy dostaneme na nedoplatok 0, našli sme jeden spôsob, akým možno zaplatiť pôvodnú sumu.

Uvedomme si, že každému možnému *zaplateniu* (koľkokrát ktorú mincu použijeme) zodpovedá práve jeden takýto *postup*, a každému postupu zodpovedá práve jedno zaplatenie. Teda počet rôznych postupov vedúcich k nedoplatku 0 je presne počet spôsobov, akým vieme zaplatiť pôvodnú sumu.

Náš stav S je určený tým, pri ktorej minci stojíme a aký je náš aktuálny nedoplatok. Nás zaujíma počet spôsobov, ktorými sa vieme z S dostať k nedoplatku 0. Na to nám stačí len sčítať tieto počet spôsobov pre všetky možné stavy, do ktorých sa vieme dostať z S. Tieto možnosti sú nanajvýš dve.



Aká je časová zložitosť? Stavov je $(n+1) \cdot (C+1)$, v každom z nich spravíme len konštantne veľa operácii. Teda časová zložitosť je $O(n \cdot C)$.

Subset sum. Zadaných je n kladných celých čísel. Vieme z nich vybrať niekoľko čísel tak, aby bol súčet vybraných čísel presne C?

Riešenie je podobné, ako pre 0-1 knapsack.

Čo majú všetky tieto úlohy spoločné?

Optimálna podštruktúra. Veľký problém vieme vyriešiť vhodným skombinovaním riešení menších podproblémov. Takéto štruktúry zvyknú byť popísané rekurzívne.

Prekrývajúce sa podproblémy. Priamočiary rekurzívny algoritmus je neefektívny preto, že viackrát vyhodnocuje tie isté podproblémy. Namiesto toho možno použiť *memoizáciu*: keď vyriešime podproblém, jeho riešenie si zapamätáme. Vždy, keď ho neskôr potrebujeme, si naňho v konštantnom čase spomenieme (namiesto opätovného počítania).

Sú problémy, v ktorých sa vyskytuje optimálna podštruktúra bez prekrývajúcich podproblémov. Takýmto riešeniam potom hovoríme *divide and conquer* (po slovensky *rozdeľuj a panuj*). Príkladom takéhoto problému je triedenie a algoritmy *quicksort* a *mergesort*.

Ďalej si uvedomme, že sú dve spôsoby, akými možno implementovať dynamické programovanie.

Tabuľkový, bottom-up. Prvý spôsob je, že si vytvoríme tabuľku podproblémov, a začneme ju vypĺňať od najmenších problémov až po tie najväčšie. Toto sa ľahko implementuje, nevýhodou je, že možno zbytočne počítame niektoré políčka tabuľky.

Lenivý, top-down. Budeme mať nejakú štruktúru, ktorá si bude schopná pamätať k už vyriešeným podproblémom ich hodnoty. Napríklad pole a `unordered_map` pre spomínanie v konštantnom čase, `map` v logaritmickom čase.

Spustíme rekurzívne riešenie s memoizáciou na ten problém, ktorý chceme vyriešiť. Ten si vyžiada riešenia tých podproblémov, ktoré potrebuje. Tie sa pozrú na ďalšie podproblémy, ... Takýmto spôsobom sa vypočítajú len tie hodnoty, ktoré skutočne potrebujeme, a môže to dokonca viesť k lepšej časovej zložitosti, ako tabuľkové riešenie.

Nakoniec, úloha pre náročných.

Backpacks. (Rýchlostné vs. výberko 2017.) Ideme s partiou kamarátov na výlet. Máme $v \leq 24$ vecí, ktoré si chceme vziať so sebou, a $b \leq 100$ batohov, do ktorých ich môžeme dať. Každá vec má svoju hmotnosť m_i. Do batohu môžeme dať aj viac vecí naraz, súčet ich hmotností však nesmie prekročiť nosnosť n_j dotyčného batohu. Zistite, či vieme všetky veci zobrať so sebou, a ak áno, koľko najmenej batohov potrebujeme.