Vyčíslitelnost a složitost výpočtů

Dynamické programování

Podstata DP: DP je ideální pro problémy, které mají *překrývající se podúlohy* a *optimální podstrukturu*:

- Překrývající se podúlohy: Podúlohy se opakují, což umožňuje efektivně využít již získaná řešení.
- Optimální podstruktura: Optimální řešení celého problému lze sestavit z optimálních řešení podúloh.

Dynamické programování

Memoizace (Top-Down přístup):

- V top-down přístupu se problém řeší rekurzivně od nejvyšší úrovně, přičemž se výsledky ukládají do paměti (např. v poli nebo mapě), aby se při dalším výskytu stejné podúlohy jednoduše načetly.

Tabulace (Bottom-Up přístup):

- V bottom-up přístupu řešíme problém iterativně od základní úrovně. Postupujeme po jednotlivých krocích, přičemž každá následující hodnota závisí na dříve vypočítaných hodnotách.

Dynamické programování

- Problém "balení batohu" (*Knapsack problem*) je klasický optimalizační problém, který lze řešit různými přístupy, včetně dynamického programování. Zadání je obvykle následující:
- Máme batoh s pevnou kapacitou W a několik předmětů, každý s danou hodnotou a váhou.
- Cílem je vybrat takové předměty, aby jejich celková váha nepřesáhla kapacitu batohu W a celková hodnota vybraných předmětů byla co nejvyšší.

Dynamické programování

- **Máme batoh** s omezenou nosností (kapacitou).
- **Máme předměty**, z nichž každý má:
  - **Hodnotu**, která představuje jeho význam nebo užitečnost.
  - **Hmotnost**, která určuje, kolik místa v batohu zabere.
- **Cíl:** Vybrat předměty tak, aby:
  - Celková váha nepřesáhla kapacitu batohu.
  - Celková hodnota vybraných předmětů byla co nejvyšší.

Dynamické programování

- Každý předmět může být buď:
  - **Zahrnut do batohu** (vezmeme ho), nebo
  - **Vynechán** (nevezmeme ho).
- Není možné vzít jen část předmětu

Dynamické programování

Postup řešení

- Problém se řeší tak, že postupně rozhodujeme pro každý předmět, zda ho:
  - **Vezmeme do batohu:** Získáme jeho hodnotu, ale také snížíme zbývající kapacitu batohu.
  - **Nevezmeme:** Hodnota se nezmění, ale máme více prostoru pro jiné předměty.
- Tyto volby opakujeme pro všechny předměty a postupně hledáme kombinaci, která přinese nejvyšší možnou hodnotu.

Dynamické programování

Strategie řešení

Problém řešíme pomocí dynamického programování, kde rozdělujeme úlohu na menší podproblémy:

1. Začínáme s prázdným batohem a postupně zkoumáme jednotlivé předměty.
2. Pro každý předmět se rozhodujeme, zda ho do batohu přidat.
3. Výsledky průběžně ukládáme, abychom nemuseli opakovaně počítat stejné situace.
4. Na konci zjistíme maximální hodnotu, které lze dosáhnout při dané kapacitě batohu.

Dynamické programování

Představte si, že máte následující předměty:

- **Předmět 1:** Hodnota 60, hmotnost 10.
- **Předmět 2:** Hodnota 100, hmotnost 20.
- **Předmět 3:** Hodnota 120, hmotnost 30.

Batoh má kapacitu 50.

- Pokud vezmete **předmět 1 a 2**, hodnota je 60+100=160, váha je 10+20=30.
- Pokud vezmete **předmět 2 a 3**, hodnota je 100+120=220, váha je 20+30=50.
- Maximální hodnota je 220, takže nejlepší je vzít **předmět 2 a 3**.

Dynamické programování

- **Partition problem** je klasický problém dynamického programování, jehož cílem je zjistit, zda lze danou množinu rozdělit na dvě podmnožiny tak, aby součet prvků v obou podmnožinách byl stejný.
- **Vstup:** Množina S s n kladnými celými čísly: $S=\{s^1,s^2,\ldots,s^n\}$
- **Cíl:** Zjistit, zda lze S rozdělit na dvě podmnožiny S1 a S2 , tak aby:

$$sum(S1)=sum(S2)$$

Dynamické programování

- Pokud je součet všech prvků v množině lichý, řešení neexistuje.
- Pokud je součet sudý, hledáme podmnožinu S1 , jejíž součet je polovina celkového součtu: target sum=sum(S)/2