Výpočetní technika a lékařská informatika

Strategie při řešení problémů

Mgr. Markéta Trnečková, Ph.D.

Řešení problémů

If you find a good move look for a better one.

- Když stojíme před složitým úkolem, můžeme ho zkusit vyřešit přímo, krok za krokem. Někdy to funguje, ale jindy se rychle ukáže, že cesta je příliš dlouhá nebo složitá.
- Občas pomůže podívat se na problém z jiné perspektivy: rozdělit ho na menší části, zkusit využít opakující se vzory, nebo si představit, že máme neomezený čas a sílu a projdeme všechny možnosti.
- Stejný úkol se tak dá řešit různými způsoby – některé jsou rychlé a elegantní, jiné sice pracné, ale spolehlivé. Volba správného postupu je klíčová pro to, zda se nám podaří dojít k výsledku efektivně.

Iterace

- **Základní myšlenka**
  - opakování určité posloupnosti kroků – každé opakování nazýváme **iterace**
  - hodí se, když je úloha složena z podobných částí
  - lze snadno zastavit po dosažení cíle
- **Příklad**
  - chceme sečíst čísla od 1 do 100
  - postup: začít s nulovým součtem, přičítat čísla jedno po druhém
  - po 100 krocích získáme výsledek 5050

Iterace

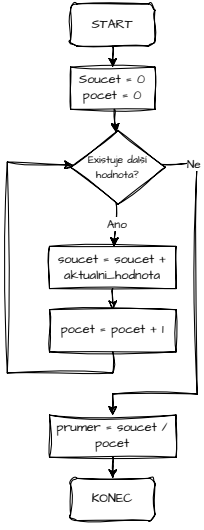
Příklad

Máte k dispozici seznam hodnot tepové frekvence (v úderech za minutu), které byly naměřeny v průběhu 24 hodin.

Vaším úkolem je navrhnout algoritmus (postup), jak z těchto hodnot spočítat průměrnou tepovou frekvenci. K zápisu postupu použijte buď pseudokód nebo nakreslete flow chart pomocí `https://www.drawio.com/` (viz první seminář).

Iterace

1. `nastav soucet = 0`
2. `nastav pocet = 0`
3. `opakuj dokud existuje další hodnota`
   1. `soucet = soucet + aktualni_hodnota`
   2. `pocet = pocet + 1`
4. `prumer = soucet / pocet`



Iterace

- jednotlivé iterace mohou obsahovat další iterace (vnořené nebo také nested)

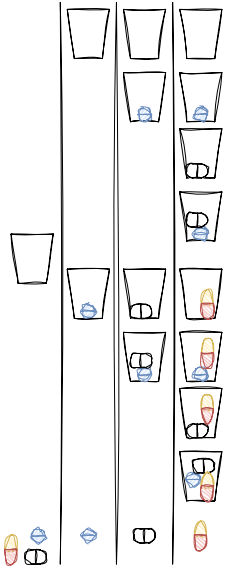
Příklad

Máme 3 různé léky. Jak bychom pomocí iterace vytvořili všechny možné kombinace těchto léků?

Item množině všech podmnožin říkáme power set (potenční množina.

Iterace

1. začneme prázdnou množinou
2. pro všechny léky udělej
   1. z aktuální množiny vytvoř kopii
   2. pro všechny nové podmnožiny
      1. přidej aktuální lék



Rekurze

- **Základní myšlenka**
  - funkce volá sama sebe s jednodušším nebo menším vstupem
  - vždy musí existovat **základní případ**, kdy se volání zastaví
  - vhodné pro úlohy, které lze rozdělit na menší podobné podproblémy
- **Příklad**
  - chceme spočítat faktoriál čísla n
  - postup: faktorial$(n) = n \cdot$ fakctorial$(n-1)$
  - základní případ: faktorial$(1) = 1$
  - například factorial$(5) = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$
- rekurzivní algoritmy jsou většinou jednodušší, ale za cenu toho, že se spouští spousta funkcí (velké režie během vykonávání kódu)

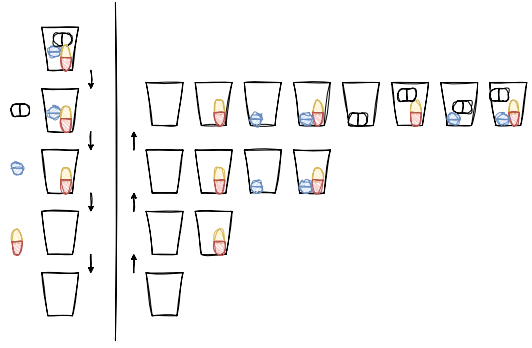
Příklad

Máme 3 různé léky. Jak bychom pomocí rekurze vytvořili všechny možné kombinace těchto léků?

Rekurze

`rekurzivni_funkce(leky)`

1. `vyber libovolný lék z leky`
2. `mnozina2 = rekurzivni_funkce(leky - vybraný lék)`
3. `mnozina3 vznikne z mnozinina2 tak, že přidáme všem podmnožinám vybraný lék`
4. `mnozina = mnozina2 + mnozina3`
5. `vrať mnozina`

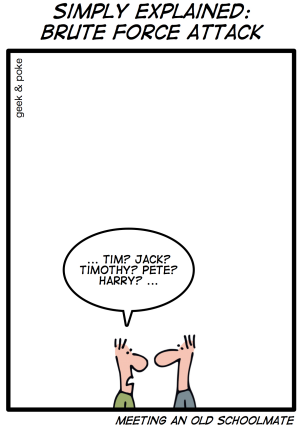


Brute force

- **Základní myšlenka**
  - vyzkoušíme všechny možné možnosti, dokud nenajdeme řešení
  - jednoduché a spolehlivé, ale často časově náročné
  - vhodné, když počet možností není příliš velký nebo když chceme mít jistotu
- **Příklad**
  - prolamování hesla

Příklad

Máme 3 různé léky. Chceme najít kombinaci, která maximalizuje účinek léčby. Jaký je účinek léčby se dá spočítat pomocí známé funkce. Jak bychom tento problém řešili?



Cvičení

Příklad

Máme 3 různé léky. Chceme najít kombinaci, která maximalizuje účinek léčby. Jaký je účinek léčby se dá spočítat pomocí známé funkce. Jak bychom tento problém řešili?

Brute force.

Příklad

Kolik existuje možných kombinací?

Příklad

Kolik by bylo kombinací, kdybychom měli 4 léky (5, 6, ...)?

Backtracking

- **Základní myšlenka**
  - procházíme všechna možná řešení postupně
  - pokud zjistíme, že částečné řešení nemůže vést k cíli, vrátíme se zpět a zkusíme jinou cestu
  - kombinuje princip „zkus vše“ (brute force) s inteligentním prořezáváním neúspěšných cest
- **Příklad**
  - chceme najít kombinaci pilulek, která maximalizuje účinek
  - postup: zkoušíme pilulky jednu po druhé, pokud částečná kombinace nemůže vést k lepšímu výsledku než dosavadní maximum, vrátíme se a zkusíme jinou kombinaci
  - výsledek: efektivnější než prostý brute force, protože nezkoušíme zbytečně všechny kombinace

Backtracking

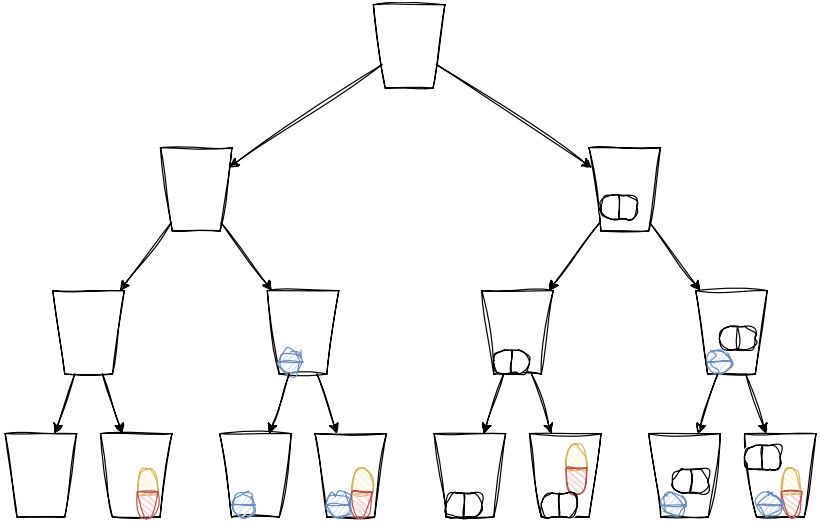
Příklad

Máme 3 různé léky. Každý má určitou „účinnost“ a „vedlejší účinky“ (např. čísla reprezentující závažnost). Chceme najít nejlepší kombinaci, která maximalizuje účinnost, ale součet vedlejších účinků nesmí překročit stanovený limit.

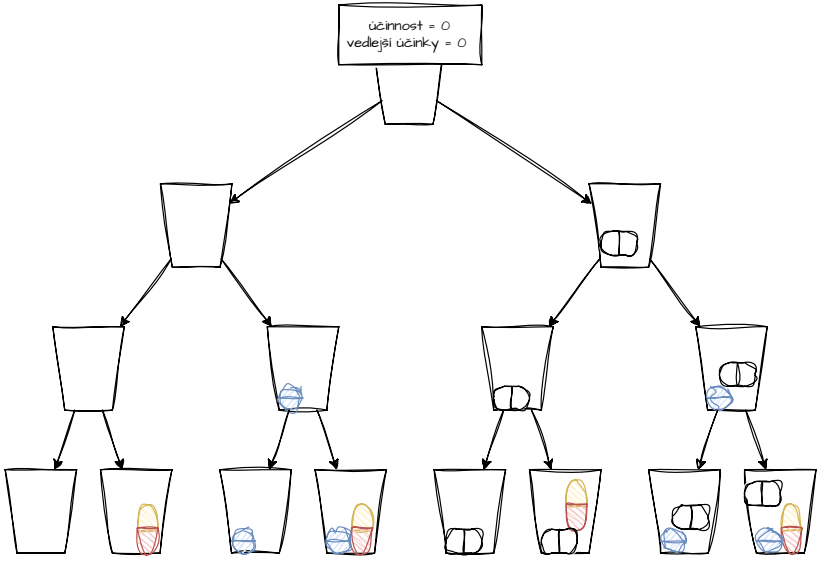
Lék	účinnost	vedlejší účinky
	3	2
	5	5
	3	1

Limit vedlejších účinků $= 5$.

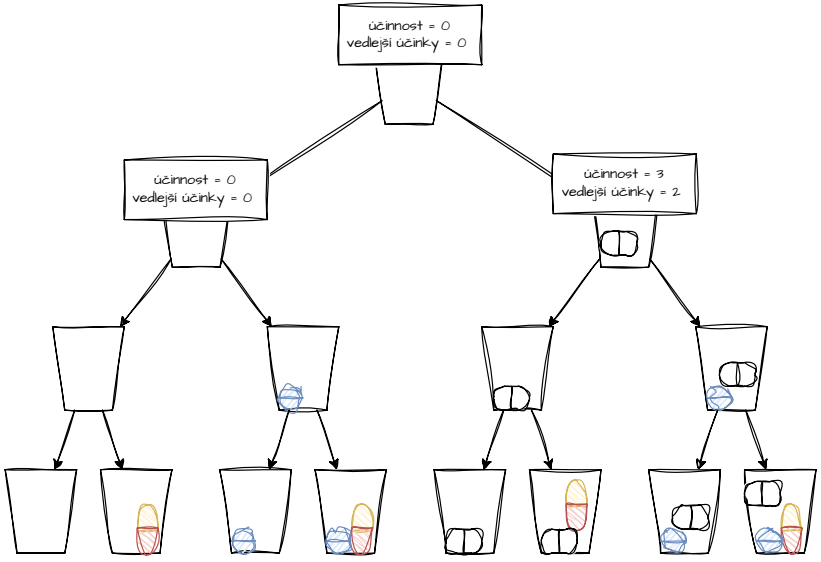
Backtracking



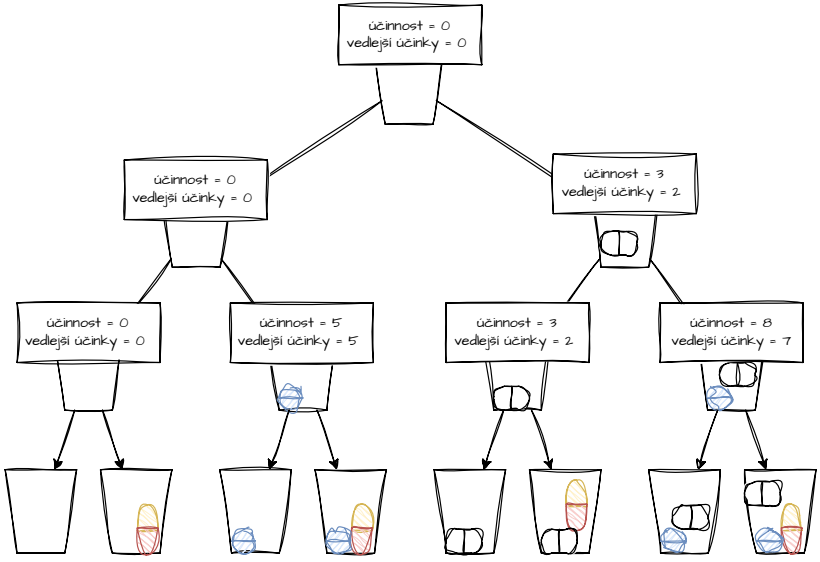
Backtracking



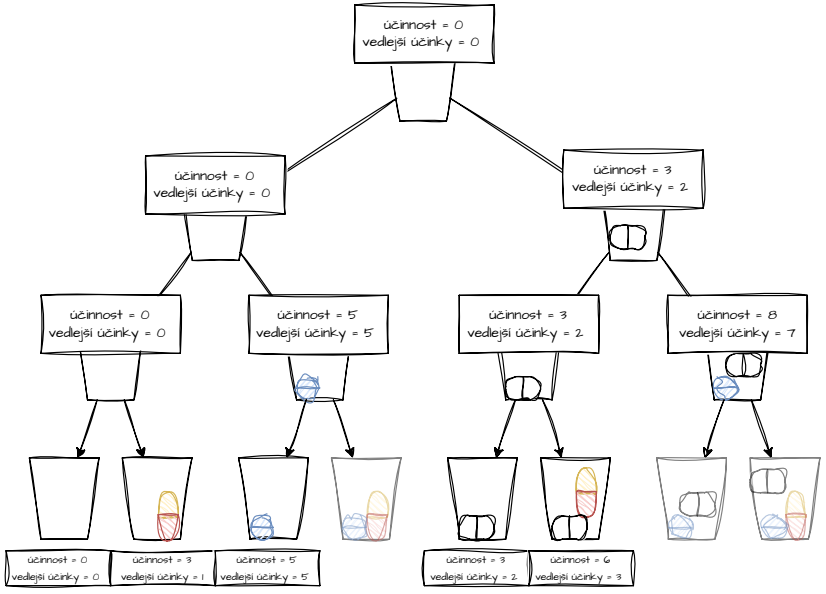
Backtracking



Backtracking



Backtracking



Heuristiky

- **Základní myšlenka**
  - ne vždy je nutné nebo možné najít **optimální** řešení
  - heuristiky jsou pravidla nebo postupy, které hledají řešení „dost dobré“ v rozumném čase
  - využívají odhad, intuici nebo jednoduchá kritéria výběru
- **Využití**
  - vhodné pro složité problémy s obrovským prohledávacím prostorem
  - často v medicíně: hledání nejvhodnější léčby, plánování operací, rozvrhování služeb

Greedy (hladový) přístup

- **Základní myšlenka**
  - v každém kroku zvolíme možnost, která se právě teď zdá nejlepší
  - rozhodnutí je rychlé a lokálně optimální, ale nemusí vést k optimálnímu výsledku celkově
- **Ukázkový příklad z medicíny**
  - chceme naplánovat podávání léků pacientovi
  - greedy strategie: vždy vybrat ten lék, který má v daný okamžik nejvyšší účinnost

Příklad

Máme 3 různé léky. Každý má určitou „účinnost“ a „vedlejší účinky“ (např. čísla reprezentující závažnost). Chceme najít nejlepší kombinaci, která maximalizuje účinnost, ale součet vedlejších účinků nesmí překročit stanovený limit (Limit vedlejších účinků $= 5$).

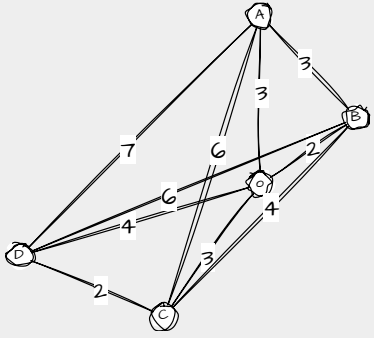
Lék	účinnost	vedlejší účinky
	3	2
	5	5
	3	1

Jaký bude výsledek, pokud zvolíme greedy přístup?

Greedy (hladový) přístup

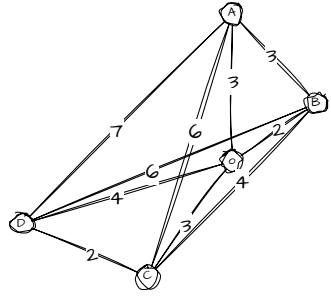
Příklad

Doktor má ordinaci v bodě O a musí navštívit své pacienty v místech A, B, C a D. Cílem je naplánovat cestu, aby navštívil všechny pacienty a vrátil se zpět do ordinace. K plánování využijte greedy přístup.



Greedy (hladový) přístup

1. Z ordinace je nejblížší nenavštívený pacient pacient B. (délka trasy 2)
2. Od pacienta B je nejbližší nenavštívený pacient pacient A. (celková délka trasy 5)
3. Dále pokračujeme k pacientovi C. (délka trasy 11)
4. Nakonec navštívíme pacienta D. (délka trasy 13)
5. Vrátíme se do ordinace. (délka trasy 17)



Příklad

Existuje lepší trasa?

Greedy (hladový) přístup

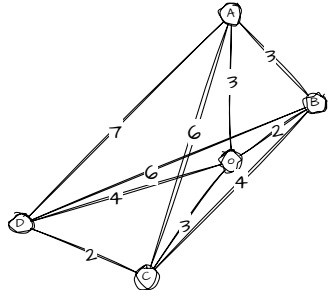
Příklad

Existuje lepší trasa?

Ano.

$O \to A \to B \to C \to D \to O$.

Délka trasy: 16

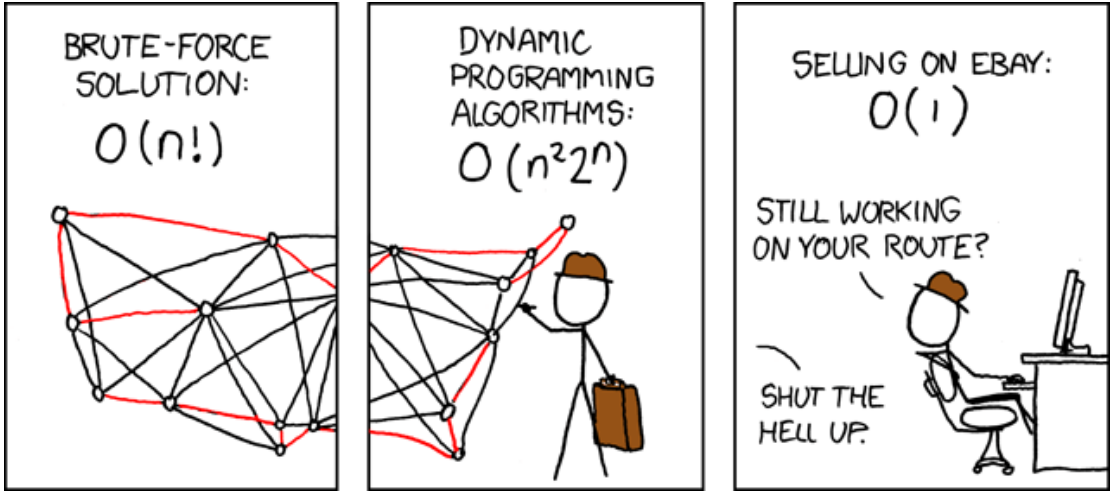


Příklad

Kolik existuje různých možností v jakém pořadí navštívit pacienty?

Problém obchodního cestujícího

The Travelling Salesman Problem

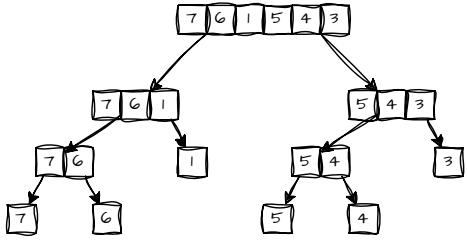


Ukázka různých algoritmů, pro řešení tohoto problému:

`https://visualgo.net/en/tsp`

Rozděl a panuj

- **Základní myšlenka**
  - složitý problém rozdělíme na menší části
  - tyto části vyřešíme samostatně
  - nakonec výsledky spojíme v řešení celku
- **Výhody**
  - úlohy se stávají jednodušší a přehlednější
  - často vede k efektivním algoritmům (např. rychlé řazení, binární vyhledávání)



Rozděl a panuj – příklad

- **Situace**
  - máme tisíce záznamů EKG signálů pacientů
  - chceme zjistit, zda je v signálu arytmie
- **Postup**
  - rozdělíme signál na menší časové úseky
  - analyzujeme každý úsek zvlášť (např. detekce vrcholů QRS)
  - výsledky spojíme: pokud některé úseky ukážou nepravidelnost, pacient může mít arytmii
- **Výsledek**
  - složitý signál zvládneme analyzovat systematicky a efektivně

Rozděl a panuj

Příklad

Pacient potřebuje individuální dávku léku, jehož interval bezpečného dávkování je 50–150 mg. Z reakce pacienta na podanou dávku dokážeme zjistit jen to, zda je dávka příliš nízká, optimální nebo příliš vysoká.

Navrhněte algoritmus, který najde optimální dávku co nejrychleji.

Příklad

Na jakém principu bude algoritmus fungovat?

Rozděl a panuj

Příklad

Simulujte postup pro tajnou dávku $X = 123$ mg.

Příklad

Porovnejte počet pokusů při použití tohoto postupu s počtem pokusů potřebných k nalezení optimální dávky postupným zkoušením všech možností.

Dynamické programování

- **Základní myšlenka**
  - rozdělení problému na menší podproblémy, které se často opakují
  - výsledky podproblémů uložíme (memoizace/tabulka), aby se nemusely počítat znovu
  - efektivní pro úlohy s překrývajícími se podproblémy (overlapping subproblems)
- **Výhody**
  - výrazně zrychluje výpočet oproti jednoduché rekurzi
  - minimalizuje zbytečné opakované výpočty
- **Příklad**
  - plánování dávkování léku během dne s různými intervaly a maximálními bezpečnými limity
  - cíl: maximalizovat účinek léčby bez překročení limitů
  - dynamické programování: uložíme optimální kombinace pro menší časové úseky a postupně je spojíme

Dynamické programování

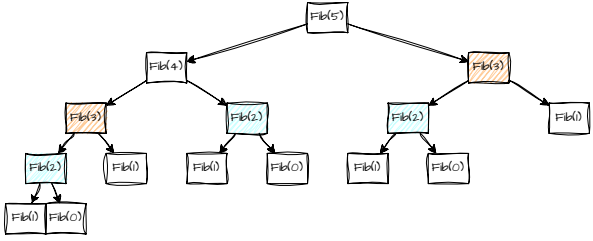
Příklad

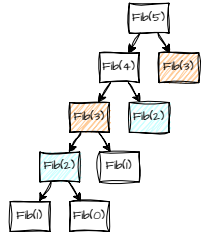
Princip dynamického programování si ukážeme na výpočtu fibonacciho čísla (posloupnosti).

$$Fib(n) = \begin{cases} 0 & \text{pro } n = 0, \\ 1 & \text{pro } n = 1, \\ Fib(n-1) + Fib(n-2) & \text{pro } n \geq 2. \end{cases}$$

Prvních pár čísel: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$

Vizualizace: `https://visualgo.net/en/recursion`





Dynamické programování

Příklad

Pacient potřebuje lék během dne, který lze podat ve třech časech: ráno, odpoledne, večer. Každá dávka má jinou účinnost:

- ráno: 2 jednotky (účinek 2)
- odpoledne: 3 jednotky (účinek 3)
- večer: 4 jednotky (účinek 4)

Celková dávka nesmí překročit 5 jednotek.

Najděte kombinaci dávek, která maximalizuje účinek bez překročení limitu. Porovnejte přístup brute force (zkusit všechny kombinace) a dynamické programování (uložit mezivýsledky menších kombinací).

Dynamické programování

Brute force

- všechny možné kombinace dávek: 0/1 dávka ráno, odpoledne, večer
- celkem 8 možností (0–1 dávka každého času)
- kontrola limitu a výběr maximálního účinku

Ráno	Odpoledne	Večer	Celkový limit	Celkový účinek	Platná kombinace
0	0	0	0	0	ano
0	0	4	4	4	ano
0	3	0	3	3	ano
0	3	4	7	7	ne
2	0	0	2	2	ano
2	0	4	6	6	ne
2	3	0	5	5	ano
2	3	4	9	9	ne

Optimální kombinace = ráno + odpoledne = 5 jednotek (účinek 5)

Dynamické programování

Dynamické programování

- vytvoříme tabulku s maximálním účinkem pro každý čas a dostupný limit (řádky časy podání, sloupce dostupný limit)
- každá buňka představuje maximální účinek pro daný čas a limit

	0	1	2	3	4	5
ráno	0	0	2	2	2	2
odpoledne	0	0	2	3	3	5
večer	0	0	2	3	4	5

- **Krok po kroku**
  1. Začínáme ráno: pokud limit $< 2 \rightarrow$ účinek $= 0$, pokud $\geq 2 \rightarrow$ účinek $= 2$
  2. Odpoledne: pro každý limit porovnáme:
     - nepodat dávku $\rightarrow$ hodnota z řádku výše
     - podat dávku $\rightarrow$ hodnota z řádku výše pro zmenšený limit o počet jednotek (3) + účinek odpolední dávky
     - vybereme maximum
  3. Večer: stejný postup – zvažujeme nepodat / podat dávku a aktualizujeme maximum
  4. Konečný výsledek maximum $= 5 \rightarrow$ optimální kombinace: ráno + odpoledne
     (vezmeme maximum z posledního řádku a jdeme směrem nahoru v tomto sloupci. Pokud je hodnota stejná, dávku jsme nepodali, pokud ne, podali)

Branch and Bound

- **Základní myšlenka**
  - efektivní strategie pro hledání optimálního řešení v kombinatorických a optimalizačních problémech
  - **Branch (větvení)**: rozdělení prostoru řešení na menší podproblémy
  - **Bound (omezení)**: odhad nejlepší možné hodnoty v dané větvi
  - pokud odhad ukazuje, že nemůže být lepší než současné nejlepší řešení → větev se neprohledává (ořezání)
- **Výhody**
  - výrazně snižuje počet kombinací, které je třeba zkontrolovat oproti brute force
  - zachovává záruku nalezení optimálního řešení
- **Příklad – medicínský kontext**
  - doktor vybírá sadu vyšetření pro pacienta, aby maximalizoval diagnostickou hodnotu, ale nepřekročil časový limit ordinace
  - každé vyšetření je buď vybráno, nebo ne (**branch**)
  - odhad maximální možné diagnostické hodnoty (**bound**) → pokud menší než známé nejlepší řešení → větev se ořezává