DELTA TopGun

(15) Dynamické programování

Luboš Zápotočný, Tomáš Faltejsek, Michal Havelka

2023

Obsah

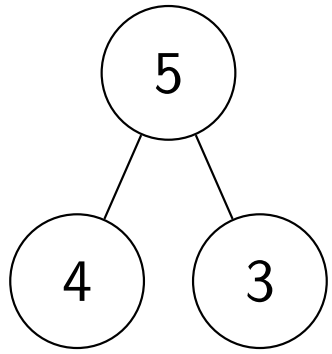
Fibonacciho posloupnost

Fibonacciho posloupnost

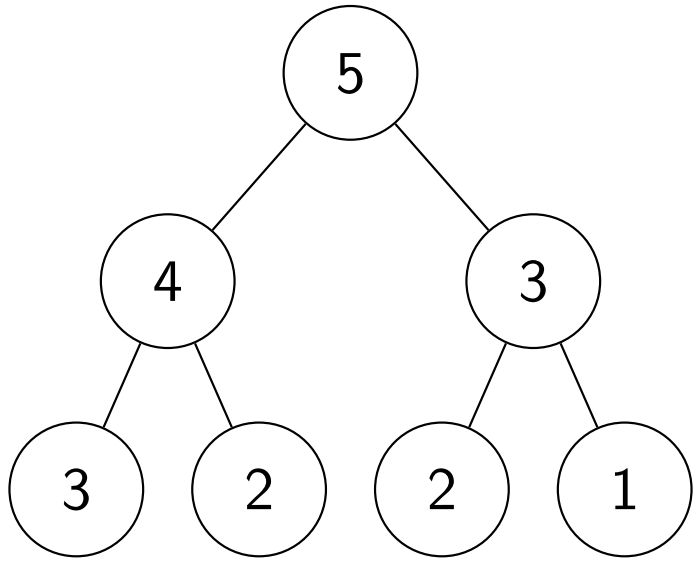
$$\mathrm{F}(n) = \begin{cases} 0, & \text{pro n} = 0 \\ 1, & \text{pro n} = 1 \\ \mathrm{F}(n-1) + \mathrm{F}(n-2), & \text{pro n} \geq 2 \end{cases}$$

Fibonacciho posloupnost - strom volání

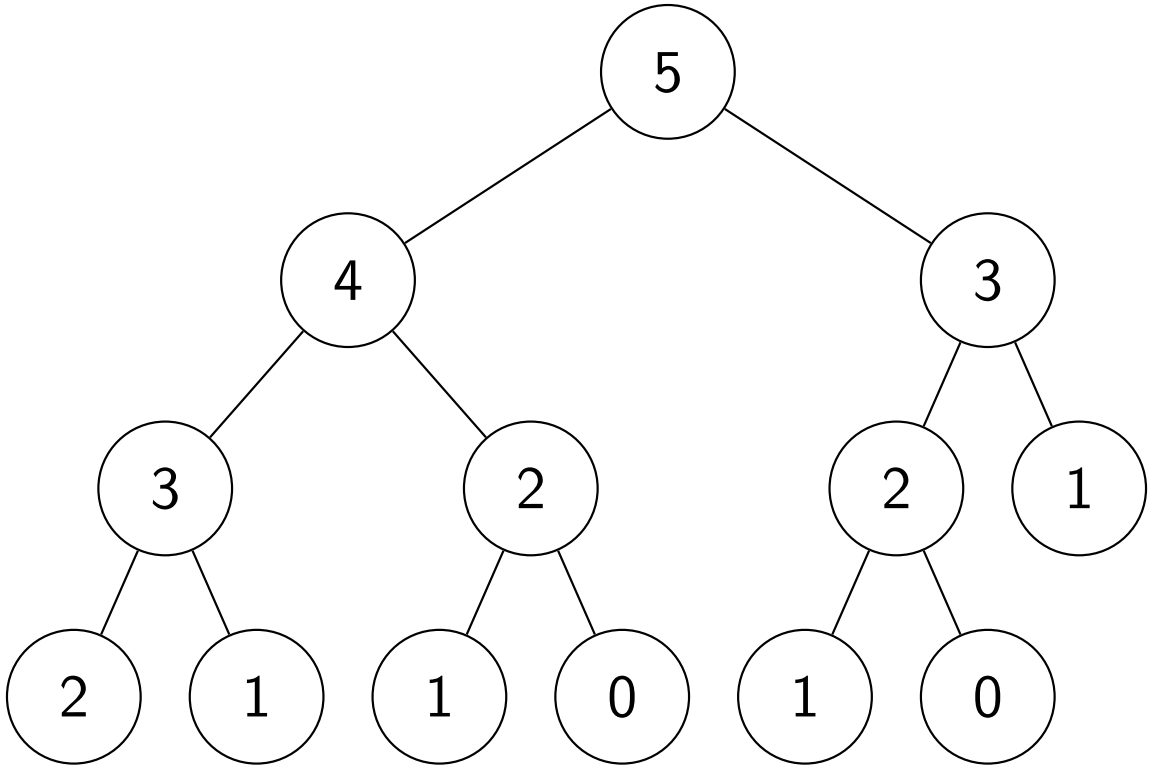
Fibonacciho posloupnost - strom volání



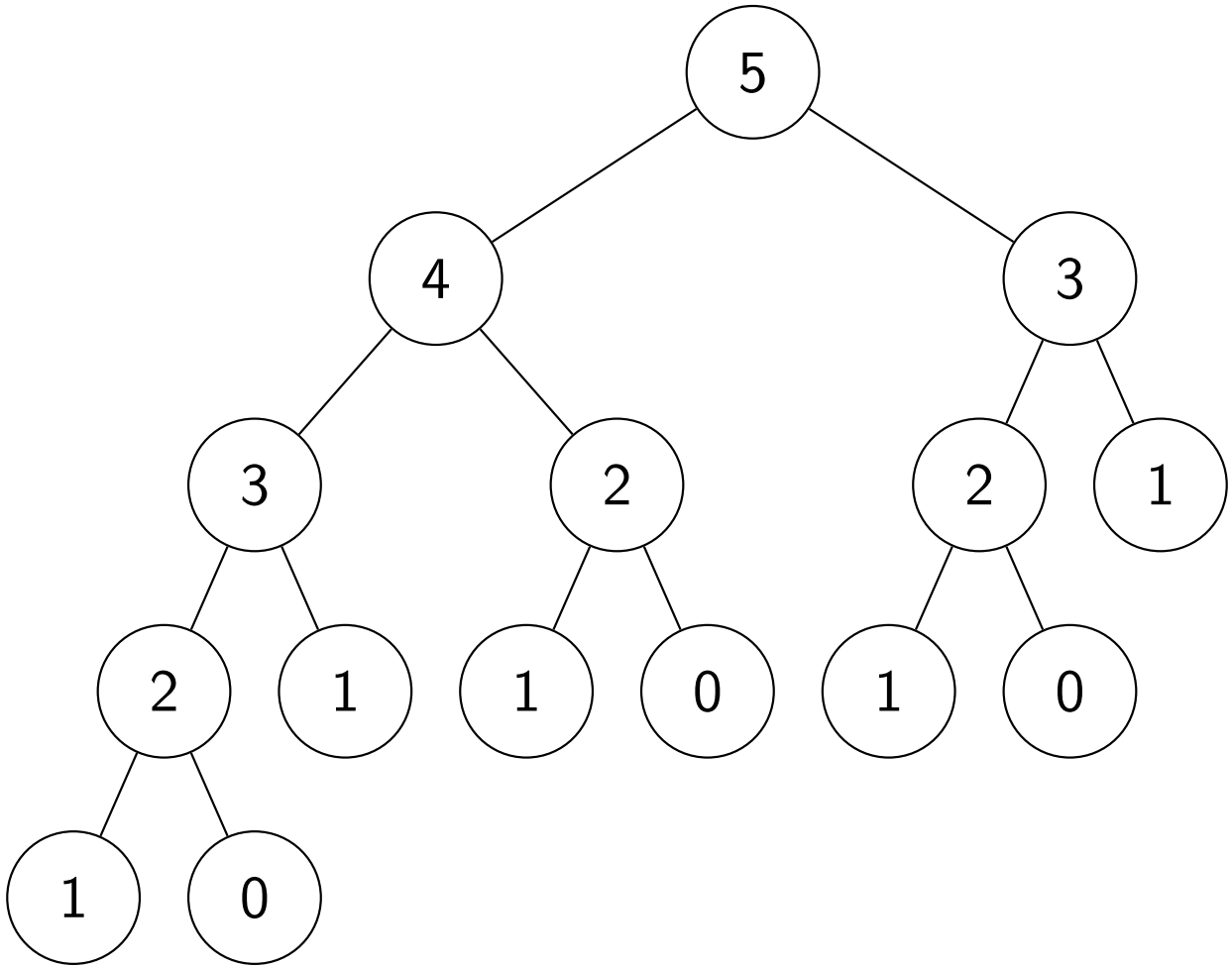
Fibonacciho posloupnost - strom volání



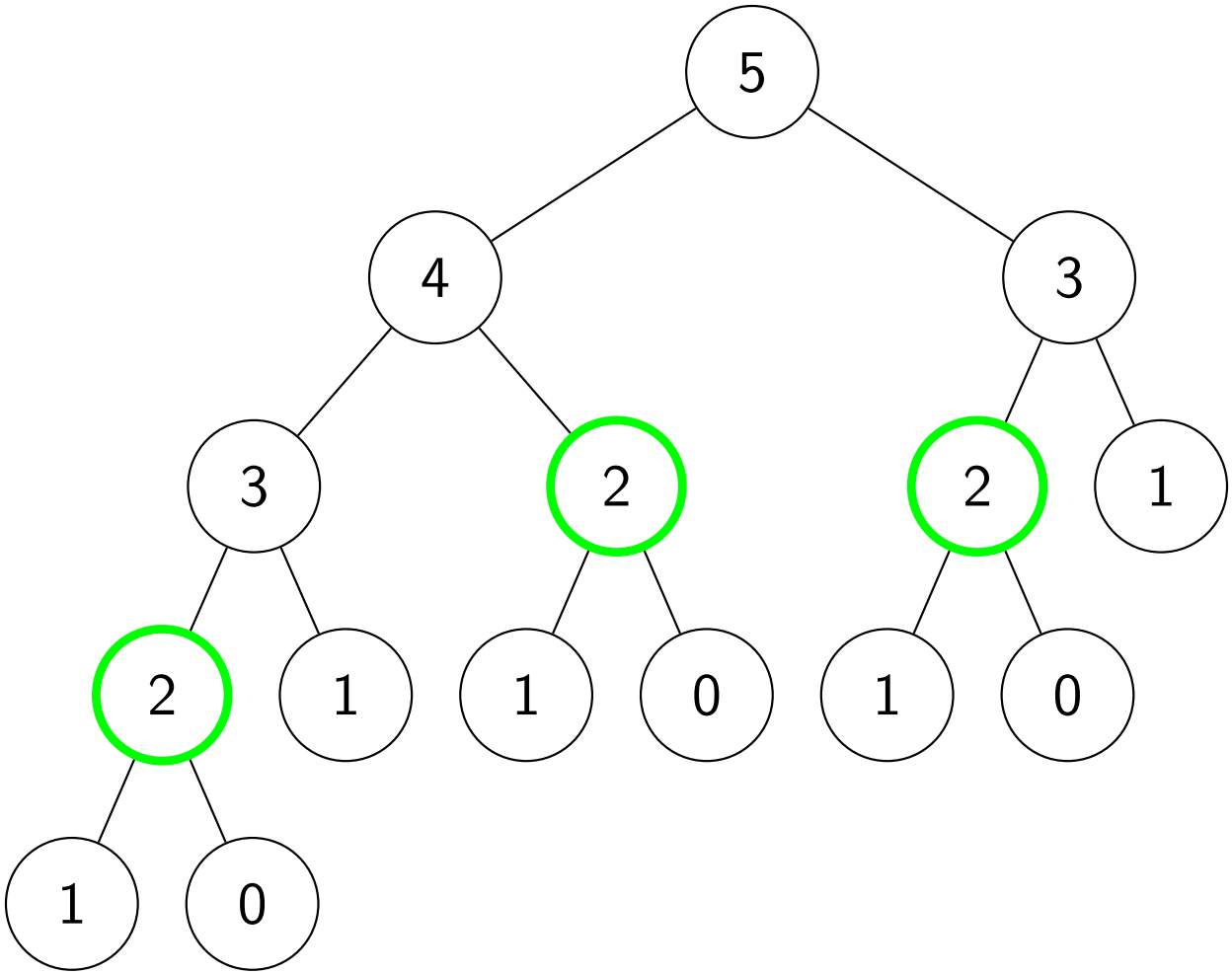
Fibonacciho posloupnost - strom volání



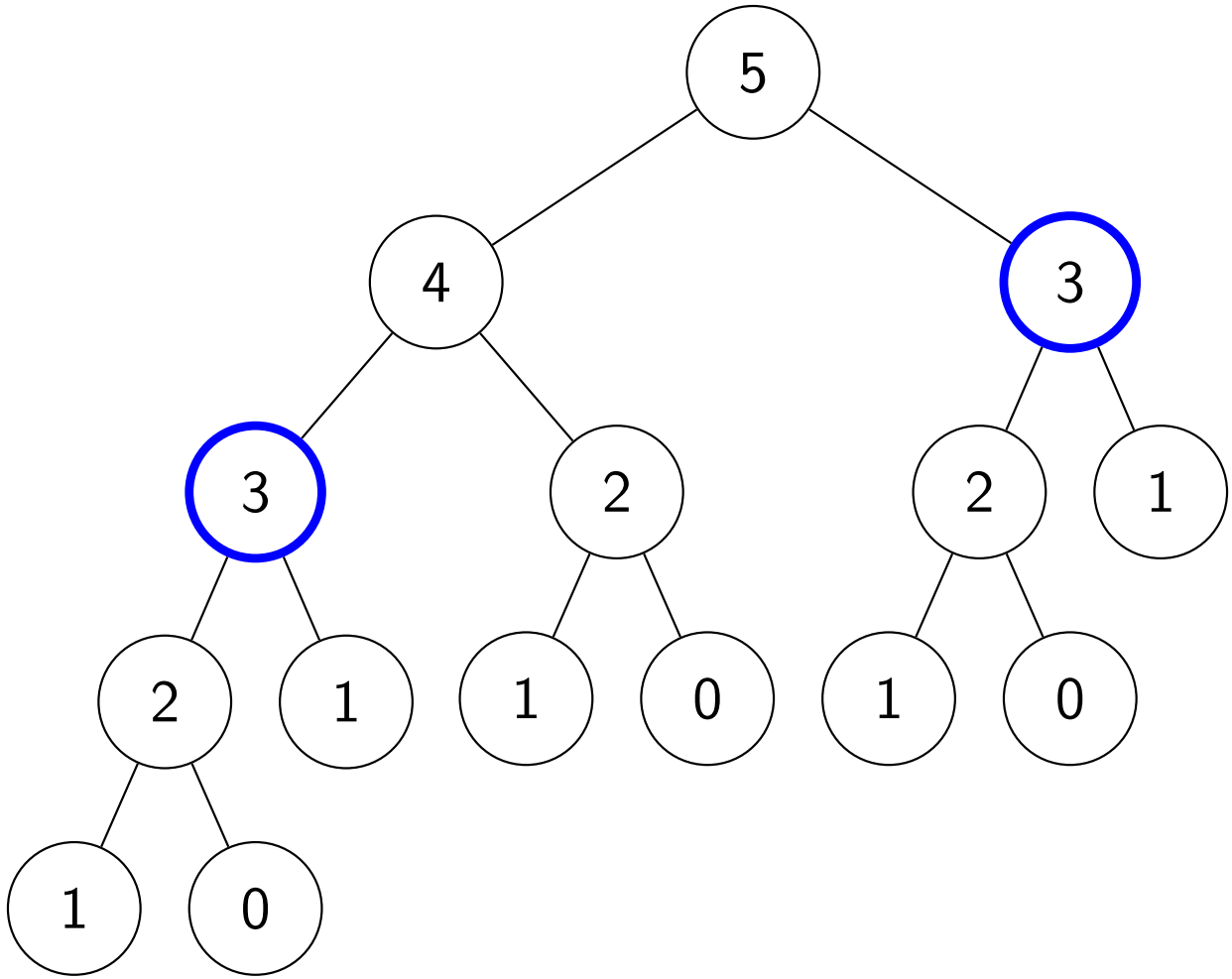
Fibonacciho posloupnost - strom volání



Fibonacciho posloupnost - strom volání - opakovaná volání



Fibonacciho posloupnost - strom volání - opakovaná volání



Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Toto (globální) pole bude složit jako taková cache pro mezivýsledky

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Toto (globální) pole bude složit jako taková cache pro mezivýsledky

Na začátku každého volání funkce fib(n) se podíváme, zdali v naší cache tabulce není již vypočítaná příslušná hodnota

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Toto (globální) pole bude složit jako taková cache pro mezivýsledky

Na začátku každého volání funkce fib(n) se podíváme, zdali v naší cache tabulce není již vypočítaná příslušná hodnota

Pokud se hodnota v tabulce nacházet nebude

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Toto (globální) pole bude složit jako taková cache pro mezivýsledky

Na začátku každého volání funkce fib(n) se podíváme, zdali v naší cache tabulce není již vypočítaná příslušná hodnota

Pokud se hodnota v tabulce nacházet nebude, rekurzivně necháme vypočítat dvě předchozí čísla

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Toto (globální) pole bude složit jako taková cache pro mezivýsledky

Na začátku každého volání funkce fib(n) se podíváme, zdali v naší cache tabulce není již vypočítaná příslušná hodnota

Pokud se hodnota v tabulce nacházet nebude, rekurzivně necháme vypočítat dvě předchozí čísla, do tabulky uložíme výsledek

Fibonacciho posloupnost - strom volání - memoizace

Jak můžeme tento výpočetní strom optimalizovat?

Alokujme si jednoduché pole celých čísel o velikosti $n + 1$, kde n značí kolikáté Fibinacciho číslo chceme vypočítat

0	1	2	3	4	5
0	0	0	0	0	0

Toto (globální) pole bude složit jako taková cache pro mezivýsledky

Na začátku každého volání funkce fib(n) se podíváme, zdali v naší cache tabulce není již vypočítaná příslušná hodnota

Pokud se hodnota v tabulce nacházet nebude, rekurzivně necháme vypočítat dvě předchozí čísla, do tabulky uložíme výsledeka tento výsledek také vrátíme

Fibonacciho posloupnost - memoizace - ukázka kódu

Ukázka a porovnání rekurzivního řešení a rekurzivního řešení s memoizací

Fibonacciho posloupnost - bottom up - ukázka kódu

Ukázka a porovnání předchozích řešení a bottom up přístupu

Plnění batohu - ukázka kódu

Ukázka možných implementací algoritmus pro řešení plnění batohu

Plnění batohu - ukázka kódu

Ukázka možných implementací algoritmus pro řešení plnění batohu

Porovnání efektivity a rozsahu zpracovatelných hodnot

Top down přístup

Tento postup lze použít, pokud řešení problému můžeme rozdělit na několik (menších) částí, které můžeme rekurzivně vyřešit a z výsledků podproblému sestavit výsledek hlavního problému

Top down přístup

Tento postup lze použít, pokud řešení problému můžeme rozdělit na několik (menších) částí, které můžeme rekurzivně vyřešit a z výsledků podproblému sestavit výsledek hlavního problému

Pokud některé podproblémy počítáme několikrát dokola, můžeme si jednotlivé mezivýsledky uložit do externí tabulky/matice a při dalších rekurzivních voláních již tuto hodnotu pouze vyčíst z tabulky/matice

Top down přístup

Tento postup lze použít, pokud řešení problému můžeme rozdělit na několik (menších) částí, které můžeme rekurzivně vyřešit a z výsledků podproblému sestavit výsledek hlavního problému

Pokud některé podproblémy počítáme několikrát dokola, můžeme si jednotlivé mezivýsledky uložit do externí tabulky/matice a při dalších rekurzivních voláních již tuto hodnotu pouze vyčíst z tabulky/matice

Tento přístup je většinou rozšíření původní rekurzivní úlohy

Top down přístup

Tento postup lze použít, pokud řešení problému můžeme rozdělit na několik (menších) částí, které můžeme rekurzivně vyřešit a z výsledků podproblému sestavit výsledek hlavního problému

Pokud některé podproblémy počítáme několikrát dokola, můžeme si jednotlivé mezivýsledky uložit do externí tabulky/matice a při dalších rekurzivních voláních již tuto hodnotu pouze vyčíst z tabulky/matice

Tento přístup je většinou rozšíření původní rekurzivní úlohy

Většinou méně efektivní, protože si musíme udržet i potenciálně velký call stack

Bottom up přístup

Tento postup lze požít, pokud nalezneme algoritmus, které postupně vyplňuje nějakou tabulkovou strukturu

Bottom up přístup

Tento postup lze požít, pokud nalezneme algoritmus, které postupně vyplňuje nějakou tabulkovou strukturu

Postupně vyplňujeme tabulku a vyplňujeme jednotlivá políčka na základě v minulosti vyplněních políček (koukáme se na minulá políčka a díky ním vyplňujeme aktuální políčko)

Bottom up přístup

Tento postup lze požít, pokud nalezneme algoritmus, které postupně vyplňuje nějakou tabulkovou strukturu

Postupně vyplňujeme tabulku a vyplňujeme jednotlivá políčka na základě v minulosti vyplněních políček (koukáme se na minulá políčka a díky ním vyplňujeme aktuální políčko)

Po kompletním vyplnění často čteme výsledek z naposledy vyplněného políčka

Bottom up přístup

Tento postup lze požít, pokud nalezneme algoritmus, které postupně vyplňuje nějakou tabulkovou strukturu

Postupně vyplňujeme tabulku a vyplňujeme jednotlivá políčka na základě v minulosti vyplněních políček (koukáme se na minulá políčka a díky ním vyplňujeme aktuální políčko)

Po kompletním vyplnění často čteme výsledek z naposledy vyplněného políčka

Tento postup většinou nepotřebuje žádnou paměť navíc či rekurzivní zanoření po celou dobu výpočtu

Bottom up přístup

Tento postup lze požít, pokud nalezneme algoritmus, které postupně vyplňuje nějakou tabulkovou strukturu

Postupně vyplňujeme tabulku a vyplňujeme jednotlivá políčka na základě v minulosti vyplněních políček (koukáme se na minulá políčka a díky ním vyplňujeme aktuální políčko)

Po kompletním vyplnění často čteme výsledek z naposledy vyplněného políčka

Tento postup většinou nepotřebuje žádnou paměť navíc či rekurzivní zanoření po celou dobu výpočtu

Nevýhoda oproti memoizaci je ta, že bottom up vyplňuje kompletně memoizační tabulku
Memoizace tabulku vyplňuje v obráceném pořadí (začíná v poslední buňce) a ve většině případů vyplní řádově méně hodnot to této tabulky

Modifikace algoritmu pro plnění batohu

Jak by se musel algoritmus s memoizací pro plnění batohu modifikovat, aby také vypsal jednotlivé předměty v batohu?

Modifikace algoritmu pro plnění batohu

Jak by se musel algoritmus s memoizací pro plnění batohu modifikovat, aby také vypsal jednotlivé předměty v batohu?

Jak by se musel algoritmus bottom up přístupu pro plnění batohu modifikovat, aby také vypsal jednotlivé předměty v batohu?