Základy jazyka Python

Optimalizácia, dynamické programovanie

prednáška 6

Katedra kybernetiky a umelej inteligencie
Technická univerzita v Košiciach
Ing. Ján Magyar

Optimalizácia kódu

- cieľom je zvýšiť výkon existujúceho programu
- súčasť ladenia kódu
- eliminácia zbytočných a viacnásobných operácií

Viacnásobné prechádzanie zoznamom

```
myList = [1240, -25, 37.24, -12, 0, 35000, 24, 17.23]

for x in range(len(myList)):
    if myList[x] < 0:
        myList[x] = 0

for x in range(len(myList)):
    myList[x] *= 1.05

print(myList)
```

Zbytočné operácie

```
def isPrime(number):
    for x in range(2, number):
        if number % x == 0:
            return False
    return True


isPrime(123475862311)
```

Dynamické programovanie

- slúži pre optimalizáciu problémov s exponenciálnou zložitosťou
- aplikácia
  - pretínajúce sa podproblémy - Fibonacciho čísla
  - optimálna štruktúra - knapsack problem

Fibonacciho čísla

```
steps = 0

def fib(a):
    global steps
    steps += 1
    print("Calculating fib for", a)
    if a == 0 or a == 1:
        return 1
    else:
        return fib(a - 1) + fib(a - 2)
```

Memorizácia

- čiastočné výsledky ukladáme do tabuľky
- ak sme ešte nevypočítali výsledok pre daný vstup, vypočítame a pridáme ho do tabuľky
- pri opätovnom volaní funkcie iba načítame výsledok z tabuľky (table lookup)
- všetky volania funkcie musia pracovať s tou istou tabuľkou

Teoréma optimálnej štruktúry

Ak riešenia podproblémov sú lokálne optimálne, potom celkové riešenie bude globálne optimálne.

Knapsack problem (problém batohu)

- klasický problém optimalizačných algoritmov
- zlodej sa vlámal do bytu a chce ukradnúť cennosti v najvyššej možnej hodnote
- zlodej má iba jeden batoh s maximálnou nosnosťou `W`
- každý objekt v byte je popísaný pomocou dvojice `(m, w)` kde `m` je hodnota objektu a `w` je jeho hmotnosť
- úlohou je nájsť množinu najcennejších objektov, ktorých celková hmotnosť nepresiahne `W`

Riešenie knapsack problému

- brute force
- vhodná reprezentácia - rozhodovací strom
  - každý uzol je reprezentovaný ako trojica `(i, w, m)`
  - ľavá vetva obsahuje prípady kde sme nevybrali objekt s indexom `i`
  - pravá vetva obsahuje prípady kde sme vybrali objekt s indexom `i`
  - optimálne riešenie je list s najväčšou hodnotou `m`

príklad:
w = [5, 3, 2]
m = [9, 7, 8]
W = 5

```
def maxVal(w, m, i, aW):
    global numCalls
    numCalls += 1
    if i == 0:
        if w[i] <= aW:
            return m[i]
        else:
            return 0
    without_i = maxVal(w, m, i - 1, aW)
    if w[i] > aW:
        return without_i
    else:
        with_i = m[i] + maxVal(w, m, i - 1, aW - w[i])
    return max(with_i, without_i)
```

```
def fastMaxVal(w, v, i, aW, m):
    global numCalls
    numCalls += 1
    try:
        return m[(i, aW)]
    except KeyError:
        if i == 0:
            if w[i] <= aW:
                m[(i, aW)] = v[i]
                return v[i]
            else:
                m[(i, aW)] = 0
                return 0
        without_i = fastMaxVal(w, v, i - 1, aW, m)
        if w[i] > aW:
            m[(i, aW)] = without_i
            return without_i
        else:
            with_i = v[i] + fastMaxVal(w, v, i - 1, aW - w[i], m)
        res = max(with_i, without_i)
        m[(i, aW)] = res
        return res
```

Otázky?