Cvičenia č. 4

2-AIN-105, Zima 2013

22. októbra 2013

1. Uvažujme klasický problém platenia mincami, t.j. máme danú sadu mincí $a_1, a_2, \ldots, a_k$ (kde $a_1 = 1$) a chceme pomocou čo najmenej mincí zaplatiť sumu n. Napíšte riešenie hrubou silou pomocou rekurzie a následne ho skúste vylepšiť.

Riešenie: Hrubá sila môže vyzerať napríklad nasledovne:

```
def zaplat(x):
  if x == 0:
    return 0
  best = x
  for minca in A:
    if minca > x:
      continue
    best = min(best, zaplat(x - minca))
  return best

zaplat(n)
```

Táto rekurzia skúša všetky možnosti zaplatenia sumy x. Aká je jej časová zložitosť? V tomto prípade sa odhaduje veľmi ťažko. Preto pre jednoduchosť ju skúsime analyzovať pre prípad keby máme len dve mince: $1, 2$. Nech na začiatku $n = 7$. Spočítajme počet volaní `zaplat(x)` pre rôzne x:

x	7	6	5	4	3	2	1	0
počet volaní	1	1	2	3	5	8	13	21

Toto sú Fibonacciho čísla. Celková časová zložitosť pre `zaplat(x)` je teda súčet prvých $x + 1$ Fibonacciho čísel, čo je $F_{x+2} - 1$. To je rádovo $O(\Phi^x)$, kde $\Phi = \frac{1+\sqrt{5}}{2}$. Čiže toto riešenie hrubou silou má exponenciálnu časovú zložitosť.

Existuje ale jednoduché zlepšenie. Všimnime si, že keď zavoláme `zaplat(x)`, tak nám vždy vráti ten istý výsledok. To znamená, že nám ju stačí spočítať raz, výsledok si niekto uložiť a potom ho používať. Tejto technike sa hovorí memoizácia. Inými slovami každú kombináciu parametrov spočítame iba raz a výsledok si uložíme do poľa. Celkovo by program vyzeral takto:

```
memo = [-1, -1, ..., -1]

def zaplat(x):
  if x == 0:
    return 0
  if memo[x] != -1:
    return memo[x]
  best = x
  for minca in A:
    if minca > x:
      continue
    best = min(best, zaplat(x - minca))
  memo[x] = best
  return best

zaplat(n)
```

Všimnite si, že sme do pôvodného programu pridali len 4 riadky. Poďme sa teraz pozrieť na časovú zložitosť. Pokiaľ už máme vypočítanú hodnotu volanej funkcie, tak jej časová zložitosť je konštantná. Celkovo sa úsek kódu za memoizáciou vykoná maximálne n krát. Zložitosť jednéj evaluácie je $O(k)$. Čo znamená, že celková zložitost je $O(nk)$, rovnaká ako pri dynamickom programovaní.

Uvažujme ešte jemnú modifikáciu predchádzajúceho programu:

```
memo = [-1, -1, ..., -1]

def zaplat(x):
  if x == 0:
    return 0
  if memo[x] != -1:
    return memo[x]
  best = x
  for minca in A:
    if minca > x:
      continue
    best = min(best, zaplat(x - minca))
  memo[x] = best
  return best

for i in 1..n:
  zaplat(i)
```

Tento program má tiež časovú zložitosť $O(nk)$. A navyše funguje presne rovnako ako riešenie pomocou dynamického programovania.

2. Na vstup je reťazec dĺžky n, nájdite v ňom počet výskytov podpostupnosti *pes* (aj nesúvislej).

Riešenie: Pre každú pozíciu v stringu si spočítame koľko tam končí podpostupností p, koľko pe a koľko pes. Nech $p[i]$ je daný počet na pozícii i, obdobne majme $pe[i]$ a $pes[i]$. $p[i]$ vieme spočítať pomerne triviálne v lineárnom čase. $pe[i]$ vieme spočítať nasledovne: Ak na pozícii i nie je znak e, tak $pe[i] = pe[i-1]$. Ak tam znak e je, tak $pe[i] = pe[i-1] + p[i-1]$ (spočítame koľko možnosti máme z predchádzajúcich pozícií a koľko možností mame s použitím posledného e). Obdobne $pes[i] = pes[i-1]$, ak na pozícii i nie je znak s a $pes[i] = pes[i-1] + pe[i-1]$, ak na pozícii i je znak s. Celkovú časovú zložitosť máme $O(n)$.

3. Daná je matica A z núl a jednotiek rozmerov $r \times s$. Nájdite v nej najväčší štvorec, tvorený iba z jednotiek.

Riešenie: Existuje veľa pomalších riešení. Optimálne riešenie dostaneme napríklad nasledovne: Nech $B[i][j]$ je rozmer najväčšieho štvorca, ktorý má pravý dolný roh na pozícii (i, j) (maximum z tejto matice je riešenie našej úlohy). $B[i][j]$ vieme spočítať nasledovne. Ak $A[i][j] = 1$, tak $B[i][j] = 0$. Ak $A[i][j] = 0$, tak $B[i][j] = 1 + \min(B[i-1][j], B[i][j-1], B[i-1][j-1])$. To máme z toho, že ak $B[i][j] = x$, tak $B[i-1][j], B[i][j-1], B[i-1][j-1]$ musia byť aspoň $x - 1$. Toto vieme implementovať so zložisťou $O(rs)$.