Riešenia 3. sady domácich úloh

Úloha 1

Pre účely tejto úlohy za *slovo* považujeme ľubovoľnú postupnosť písmen. Písmeno je v slove *šťastné*, pokiaľ sa v ňom nachádza na siedmich za sebou idúcich miestach a nikde inde. Napr. `a` je štastné v slove `bcaaaaaaadcdcc`, ale `a` nie je šťastné v slovách `bacadagafaaha` či `baaaaaaaccbab`. Určte, koľko existuje

a) *(0,5 b)* slov dĺžky 9 zložených z písmen a, b, c, d

b) *(0,5 b)* slov dĺžky 16 zložených z písmen a, b, c, d, e, f, g, h, i, ktorých písmená sú zoradené abecedne;

c) *(1 b)* slov dĺžky 20 zložených z písmen a, b, c, d, e, f, ktoré obsahujú práve dve šťastné písmená;

d) *(1,5 b)* slov dĺžky 60 zložených z písmen anglickej abecedy (ktorých je 26), ktoré obsahujú každé písmeno aspoň raz.

Vaše tvrdenia neformálne zdôvodnite. Počty v úlohách okrem d) sú malé nie preto, aby ste nahradili matematické riešenie vygenerovaním všetkých možností pomocou počítača, ale aby ste si tým vedeli pomôcť a kvôli bonusu. Vaše zdôvodnenia musia byť efektívne ručne overiteľné. V podúlohe d) môžete vo výsledku uviesť jednu sumu. Zvyšné výsledky uveďte v uzavretom tvare (teda bez súm, troch bodiek a podobných zdĺhavých vecí).

Riešenie časti a)

Pre každé z 9 miest máme 4 možnosti, aké písmeno tam môžeme dať. Aplikovaním pravidla súčinu dostávame 4^9 slov. (K rovnakmu záveru sa dá dospieť aj využitím variácií s opakovaním.)

Riešenie časti b)

Riešenie cez guličky a paličky. Každé slovo s abecedne zoradenými písmenami môžeme jednoznačne reprezentovať ako postupnosť pozostávajúcu zo 16 guličiek a 8 paličiek. V každej takejto postupnosti nám 8 paličiek rozdelí guličky do 9 (potenciálne prázdnych úsekov), kde prvý úsek predstavuje písmená `a`, druhý úsek `b`, ... až deviaty úsek písmená `i`. Z každého písmena zoberieme toľko, koľko guliečiek máme v príslušnom úseku. Počet spôsobov, ako usporiadať 16 guličiek a 8 paličiek je

$$\binom{24}{16} = \binom{24}{8},$$

nakoľko z celkových 24 vyberáme 16 miest pre guličky, čím je pozícia paličiek jasne určená.

Riešenie cez kombinácie s opakovaním. Vyberieme, aké písmená bude slovo obsahovať. Máme 9 písmen a chceme z nich vybrať 16, čiže ide o kombinácie 16-tej triedy z prvkov $\{\texttt{a}, \texttt{b}, \ldots, \texttt{i}\}$, ktorých je

$$\binom{9+16-1}{16} = \binom{24}{16}$$

možností. Pre každú takúto možnosť existuje práve jeden spôsob, ako môžeme z nich vytvoriť slovo, a to taký, kde písmená zoradíme do abecedného poradia. Teda $\binom{24}{16}$ je hľadaný počet

Komentár. Vo viacerých riešeniach som videl použitý vzorec $\binom{n+k-1}{k-1}$ pre počet kombinácií s opakovaním, ktorý takto nebol uvedený na prednáške ani v skriptách. V zásade je tento vzorec správny, ale na vyčísľovanie kombinácií s opakovaním n-tej triedy z k prvkov, teda keď z k prvkov vyberáme n – a to je kus divné, keďže na prednáške sme vždy vyberali z n prvkov k. Zaujímalo by ma, odkiaľ ste vlastne taký vzorec zobrali, tak mi kľudne napíšte.

Nebránime vám používať aj tento vzorec, napokon je na vás, ako si premenné vo vzorci pomenujete. Len chceme zdôrazniť, že ak používate či už tento alebo iný vzorec z prednášky, tak musí byť jasné, ako ho používate, teda ako do neho dosadzujete. V tomto prípade by ste mali uviesť, koľkátu triedu kombinácie s opakovaním uvažujete a z koľkých prvkov. Alebo uviesť, z čoho si vyberáte a koľkokrát. V tomto smere vás chceme varovať pred neúplným riešením nasledovného typu:

„Kombinácie s opakovaním: $n = 16$, $k = 9$, $\binom{n+k-1}{k-1} = \binom{24}{8}$.“

Toto riešenie vyzerá tak, že ste tipli vzorec na kombinácie s opakovaním (a ešte k tomu nesprávne, lebo to má byť $\binom{n+k-1}{k}$) a tipli ste, že doňho treba dosadiť $n = 9$, $k = 16$ (tiež nesprávne, lebo nevyberáme z 16-tich prvkov 9, ale opačne). Tieto dve chyby sa vyrušili a náhodou vyšiel správny výsledok. Riešenia takéhoto typu považujeme za nedostatočne odôvodnené.

Riešenie časti c)

1. Najskôr si zo šiestich písmen vyberieme dvojicu šťastných písmen: $\binom{6}{2}$ možností.
2. Teraz ideme postupne tvoriť výsledné slovo. Máme dva úseky šťastných písmen tvoriace $2 \cdot 7 = 14$ znakov, čím nám ostane ešte $20 - 14 = 6$ zvyšných miest – tými začneme. Na každom z nich môže byť ľubovoľné z ostávajúcich 4 nie šťastných písmen: 4^6 možností
3. Teraz do tohto 6-znakového slova vložíme úseky šťastných písmen. Prvý úsek (napr. ten, kde šťastné písmeno je skôr v abecede) môžeme vložiť na 7 miest: 7 možností.
4. Na vloženie druhého šťastného úseku máme 8 možností.

Spolu tak máme na základe (zovšeobecneného) pravidla súčinu

$$\binom{6}{2} \cdot 6^4 \cdot 7 \cdot 8 \qquad \text{možností.}$$

Jednotlivé časti riešenia možno aj prehadzovať, napr. môžeme najprv vybrať pozície pre šťastné úseky a potom vybrať výsledné písmená.

Riešenie časti d)

Na riešenie použijeme princíp zapojenia a vypojenia. Nech M označuje množinu všetkých 60-znakových slov z písmen anglickej abecedy. Podľa pravidla súčinu $|M| = 26^{60}$. Vypočítame teraz zlé možnosti, teda počet slov, ktoré neobsahujú aspoň jedno písmeno. Rozdeľme si všetky tieto zlé možnosti do 26 množín $M_1, M_2, \ldots, M_{26}$, pričom množina M_i bude obsahovať slová z M, ktoré neobsahujú i-te písmeno anglickej abecedy. Pre jednoduchosť ďalej v riešení budeme i-te písmeno stotožňovať s číslom i, teda `a` je 1, `b` je 2 až `z` je 26. Počet všetkých zlých možností teda dostaneme ako $|M_1 \cup M_2 \cup \cdots \cup M_{26}|$.

Podľa princípu zapojenia a vypojenia potrebujeme zistiť

$$|M_{i_1} \cap M_{i_2} \cap \cdots \cap M_{i_k}|.$$

To predstavuje počet prvkov množiny všetkých slov z M, ktoré neobsahujú žiadne z písmen $i_1, i_2, \ldots, i_k$. Inými slovami, ide o počet slov, ktoré neobsahujú žiadne z k predpísaných písmen. Počet týchto slov je podľa pravidla súčinu

$$(26-k)^{60},$$

nakoľko na každé zo 60 miest máme na výber $26-k$ písmen (vyberáme z 26 znakovej abecedy, ale k písmen máme zakázaných.

Ďalej určíme k-ty čiastkový súčet S_k, ktorý podľa princípu zapojenia a vypojenia dostaneme sčítaním hodnoty $|M_{i_1} \cap M_{i_2} \cap \cdots \cap M_{i_k}|$ pre všetky možné voľby k zakázaných písmen $i_1, i_2, \ldots, i_k$. Každá takáto voľba je výber k písmen z 26, čiže spolu ich máme $\binom{26}{k}$. Na základe predošlého odseku má všetkých $\binom{26}{k}$ sčítancov rovnakú hodnotu $(26-k)^{60}$, teda

$$S_k = \binom{26}{k}(26-k)^{60}.$$

Teraz už len dokončíme hľadaný počet zlých možností

$$|M_1 \cup M_2 \cup \cdots \cup M_{26}| = \sum_{k=1}^{26}(-1)^{k+1}S_k = \sum_{k=1}^{26}(-1)^{k+1}\binom{26}{k}(26-k)^{60}.$$

Hľadaný počet dobrých možností teda je

$$26^{60} - \sum_{k=1}^{26}(-1)^{k+1}\binom{26}{k}(26-k)^{60},$$

čo ešte vieme upraviť na

$$(-1)^0\binom{26}{0}(26-0)^{60} + \sum_{k=1}^{26}(-1)^k\binom{26}{k}(26-k)^{60} = \sum_{k=0}^{26}(-1)^k\binom{26}{k}(26-k)^{60}.$$

Bonus

Pre každú úlohu najskôr uvedieme riešenie, ktoré priamo zodpovedá uvedenému riešeniu k úlohe 1. Potom uvedieme aj nejaké alternatívy

Podúloha a)

Riešenie cez variácie – product. V riešení sme využili variácie s opakovaním. Tie vieme v Pythone generovať cez funkciu `itertools.product`.

```
from itertools import product

def a1():
    for choice in product('abcd', repeat=9):
        print(''.join(choice))
```

Riešenie založené na pravidle súčinu Riešenie, ktoré by zodpovedalo jednoduchému použitiu pravidla súčinu, by vyzeralo takto.

```
1  def a2():
2      pismena = 'abcd'
3      for a in pismena:
4          for b in pismena:
5              for c in pismena:
6                  for d in pismena:
7                      for e in pismena:
8                          for f in pismena:
9                              for g in pismena:
10                                 for h in pismena:
11                                     for i in pismena:
12                                         print(a+b+c+d+e+f+g+h+i)
```

Na tejto situácii si vieme aj pekne všimnúť, prečo čísla štyri medzi sebou násobíme. Cyklus `for a in pismena:` vykoná štyri opakovania, v každom sa vykonajú 4 opakovania cyklu `for b in pismena:`, čo dáva zatiaľ $4 \cdot 4$ opakovaní a takto postupujeme ďalej, až kým dostaneme 4^9 opakovaní.

Riešenie cez rekurziu. Pokiaľ nechceme využívať knižničné funkcie alebo chceme napísať o niečo všeobecnejší alebo krajší program ako 9 vnorených for cyklov, tak môžeme si takéto generovanie slov naprogramovať aj sami s využitím rekurzie.

```
1  # Vrati zoznam vsetkych k-znakovych slov skladajucich sa z pismen pismena
2  def slova(pismena, k):
3      # Ak k = 0, tak existuje jedine slovo dlzky 0 - prazne slovo
4      if k == 0:
5          return ['']
6      vysledok = []
7      # V opacnom pripade rekurzivne ziskame vsetky slova dlzky k - 1
8      for var in slova(pismena, k - 1):
9          # A ku kazdemu z nich pridame na koniec kazde z pismen
10         for p in pismena:
11             vysledok.append(var + p)
12     return vysledok
13
14 for slovo in slova('abcd', 9):
15     print(slovo)
```

Ako o takomto programe bez spustenia zistíme, koľko možností vypíše. Vieme dokázať matematcikou indukciou, že pre každé prirodzené k funkcia `slova(pismena, k)` vypíše n^k slov, kde n je počet znakov v premennej `pismena`. Pre $k = 0$ vráti zoznam s jediným (práznym slovom) a $n^0 = 1$, čo sedí.

Uvažujme teraz $k > 0$ a predpokladajme, že funkcia `slova(pismana, k - 1)` vráti n^{k-1} slov. Vďaka indukčnému predpokaldu cyklus `for var in slova(pismena, k - 1):` z riadku 8 vykoná n^{k-1} opakovaní. Cyklus `for p in pismena` vykoná n opakovaní. To nám dáva celkovo $n^{k-1} \cdot n = n^k$ opakovaní a v každom sa do zoznamu `vysledok` vloží práve jedno slovo. Teda funkcia `slova(pismena, k)` vráti n^k slov, čo sme mali dokázať.

Podobne možno naprogramvoať aj funkciu, ktorá vráti variácie k-tej triedy ľubovoľnej množiny A – teda to, čo robí funkcia `itertools.product`. Dôkaz pre túto funkciu by vyzeral podobne – porovnajte si ho s dôkazom pre počet variácií s opakovaním z prednášky.

Z programátorského hľadiska je krajšie a pamäťovo efektívnejšie rekurzívnu funkciu postaviť tak, aby priamo vypisovala slová miesto ich ukladania do zoznamu. Tento prístup použijeme pri riešení podúlohy b). Prístup cez zoznam sme zvolili preto, lebo lepšie zodpovedá dôkazu počtu variácií s opakovaním.

Podúloha b)

Riešenie cez kombinácie s opakovaním Keďže v podstate ide cez kombinácie s opakovaním, tak najjednoduchším spôsobom je využitie funkcie `itertools.combinations_with_replacement`, ktorá nám práve tie generuje. Na tomto programe si vieme aj ujasniť, že pri tejto úlohe ide o kombinácie s opakovaním 16. triedy z 9 prvkov $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{i}\}$. Na základe toho potom vieme dosadiť aj do vzorca. (Teda nejde o kombinácie s opakovaním 9. triedy zo 16 prvkov alebo niečo iné.)

```
from itertools import combinations_with_replacement

def b1():
    for choice in combinations_with_replacement('abcdefghi', 16):
        print(''.join(choice))
```

Riešenie cez guličky a paličky. Toto riešenie je náročnejšie na implementáciu, no ukazuje ako možno s využitím myšlienky z prednášky implementovať funkciu na kombinácie s opakovaním.

```
from itertools import combinations

def b2():
    gulicky = 16
    palicky = 8
    # Z gulicky + palicky miest vyberieme gulicky miest, kam ulozime gulicky
    for pozicie_guliciek in combinations(range(gulicky + palicky), gulicky):
        # Zostavime rozlozenie guliciek a paliciek
        p = ['I'] * (gulicky + palicky)
        for i in pozicie_guliciek:
            p[i] = 'O'
        # Mozeme odkomentovat, ak chceme vidiet, vygenerovane
        # rozmiestnenia guliciek a paliciek
        # print(p)

        # Vygenerovane rozmiestnenie guliciek a paliciek prevedieme na slovo
        slovo = []
        # Zaciname znakom a
        znak = 'a'
        for i in range(len(p)):
            # Ak stretneme gulicku, znamena to, ze pridavame do vysledneho slova
            # aktualny znak
            if p[i] == 'O':
                slovo.append(znak)
            # Ak streme palicku, posuvame symbol o 1 v abecede
            # Kedze paliciek mame 8, tak posun spravime 8-krat,
            # teda skoncime na znaku i
            if p[i] == 'I':
                znak = chr(ord(znak) + 1)
        # Vysledn slovo vyiseme
        print(''.join(slovo))
```

Rekurzívne riešneie Aj pri tejto úlohe sa dá spraviť pomerne jednoduché rekurzívne riešenie. V parametri `prefix` si budeme udržiavať doteraz vygenerované slovo a v jednom rekurentom volaní pridáme všetky možné znaky, pričom možné sú len tie, ktoré neporušia abecedné poradie. Tu je ešte viacero miest, kde sa dá pohrať s efektivitou. Trochu pokaziť sa dá tým, že túto kontrolu budeme robiť neoptimálne, napr. porovnávať nový znak so všetkými vygenrovanými v `prefix` (stačí porovnať s posledným) alebo zistiť, či sa slovo zmení po jeho usporiadaní. Efektívny spôsob je poslať do rekurntného volania informáciu o tom, ktoré písmená má zmysel skúšať.

```
# Vypise vsetky slova dlzky k, ktorych znaky su v poradi ako su uvedene v parametri
    znaky
def b3(znaky='abcdefghi', k=16, prefix=''):
    # Ak k = 0, tak len vypsime prefix
    # (prdiavame k nemu vlastne slovo dlzky 0 a to je len jedno)
    if k == 0:
        print(prefix)
        return
    # V opacnom priapde rekurzivne vyskusme vsetky znaky, ktorym mozeme zacat
    for i in range(len(znaky)):
        # Ak zaciname znakom znaky[i], tak dalej v slove mozu byt uz len znaky
        # totozne alebo dalej v abecede, teda znaky[i:]
        b3(znaky[i:], k - 1, prefix + znaky[i])
```

Počet vypísaných možností opäť môžeme dokázať matematickou indukciou. Budeme dokazovať, že pre každé prirodzené k, pre každé celé $n > 0$ funkcia `b3(znaky, k, prefix)`, kde `znaky` má n znakov, vypíše $\binom{n+k-1}{k}$ slov. Indukciu robíme podľa premennej k. Reťazec `prefix` ignorujeme, nakoľko počet vypísaných slov neovplyvňuje. (Ak by sme by sme chceli byť poriadnejší, tak do dokazovaného tvrdenia pridáme kvantifikáciu „pre každý reťazec `prefix`“.

Pre $k = 0$ funkcia vypíše jedno slovo a $\binom{n-1}{0} = 0$, keďže $n > 0$. Bázu máme overenú.

Uvažujme teraz $k > 0$ a predpokladajme platnosť tvrdenia pre $k - 1$, teda predpokladáme, že pre každé celé $n > 0$ funkcia `b3(znaky, k - 1, prefix)`, kde `znaky` má n znakov, vypíše $\binom{n+k-2}{k-1}$ slov. Uvažujme i-te opakovanie cyklu `for i in range(len(znaky)):`, pre $i \in \{0, 1, \ldots, n-1\}$. V tomto opakovaní voláme funkciu `b3(znaky[i:], k - 1, prefix + znaky[i])`, v ktorej `znaky[i:]` obsahuje $n - i$ znakov. Preto podľa indukčnho predpokladu táto funkcia vypíše $\binom{n-i+k-2}{k-1}$ slov. Podľa pravidla súčinu dostávame, že celé volanie funkcie `b3(znaky, k, prefix)` vypíše

$$\sum_{i=0}^{n-1} \binom{n-i+k-2}{k-1} = \binom{n+k-2}{k-1} + \binom{n+k-3}{k-1} + \cdots + \binom{k}{k-1} + \binom{k-1}{k-1}$$

možností. Ostáva už len ukázať, že táto suma je rovná $\binom{n+k-1}{k}$. To možno spraviť opäť matematickou indukciou alebo aj počítaním dvomi spôsobmi – napokon sumu na tento štýl sme počítali aj na cvičeniach.

Tento rekurzívny prístup vlastne odhaľuje zaujímavý vzťah kombinačných čísel

$$\binom{n+k-1}{k} = \sum_{i=0}^{n-1} \binom{n-i+k-2}{k-1}.$$

príp. aj vzťah medzi počtom $C'_k(n)$ kombináciami s opakovaním k-tej triedy z n prvkov

$$C'_k(n) = \sum_{i=0}^{n-1} C'_{k-1}(n-i).$$

Podúloha c)

Uvádzame jeden z možných programov, ktorý sleduje riešenie z úlohy 1.

```
def c1():
    pismena = {'a', 'b', 'c', 'd', 'e', 'f'}
    # Vyberieme stastne pismena
    for x, y in combinations(pismena, 2):
        # Vyberieme zo zvysnych pismen na zvysnych 6 pozicii
```

```
        for zvysne in product(pismena - {x, y}, repeat=6):
            # Vyberieme poziciu, kam vsunieme prvy usek stastnych pismen
            # (usek zatial povazujeme za jeden objekt, aby sme ho vsunutim druheho
            # useku nenarusili)
            for i in range(7):
                slovo2 = zvysne[:i] + (x * 7,) + zvysne[i:]
                # Vyberieme poziciu, kam vsunieme druhy usek stastnych pisme
                for j in range(8):
                    slovo3 = slovo2[:j] + (y * 7,) + slovo2[j:]
                    # Vypiseme vysledne slovo
                    print(''.join(slovo3))
```

Úloha 2

V závislosti od nezáporného celého čísla $n \in \mathbb{N}$ vypočítajte sumu

$$\sum_{k=0}^{n} \frac{(-2)^{k+1}}{n+1-k}\binom{n}{k}.$$

$$\sum_{k=0}^{n} \frac{(-2)^{k+1}}{n+1-k}\binom{n}{k} = \sum_{k=0}^{n} \frac{(-2)^{k+1}}{n+1-k} \cdot \frac{n!}{k!(n-k)!} = \sum_{k=0}^{n} (-2)^{k+1} \frac{n!}{k!(n+1-k)!} =$$

$$\sum_{k=0}^{n} \frac{(-2)^{k+1}}{n+1} \cdot \frac{(n+1)!}{k!(n+1-k)!} = \sum_{k=0}^{n} \frac{(-2)^{k+1}}{n+1}\binom{n+1}{k} = \frac{-2}{n+1}\sum_{k=0}^{n} (-2)^{k}\binom{n+1}{k} =$$

$$\frac{-2}{n+1}\left(\sum_{k=0}^{n+1} (-2)^{k}\binom{n+1}{k} - (-2)^{n+1}\binom{n+1}{n+1}\right) = \frac{-2}{n+1}\left((-2+1)^{n+1} - (-2)^{n+1}\right) =$$

$$\frac{-2}{n+1}\left((-1)^{n+1} - (-2)^{n+1}\right)$$