Kombinatorika cez programovanie

Jozef Rajník

Úvod

Tento text ukazuje, ako si možno pri riešení kombinatorických úloh pomôcť programovaní. Je hlavne zameraný na súvis medzi matematickým riešením úloh a ako tieto riešenia zodpovedajú programom, ktoré nám vypíšu všetky možnosti.

Cieľom tohto textu je pomôcť pochopiť čitateľom niektoré dôležité aspekty kombinatoriky. Predsa len, program vieme spustiť, vidíme jeho výstup a môžeme ho aj otestovať. Je teda hmatateľnejší ako nejaké abstraktné matematické riešenie, ktoré si nie každý vie predstaviť.

Na začiatok sa patrí zdôrazniť, že cieľom nie je učiť sa písať efektívne algoritmy na generovanie kombinatorických objektov (hoci aj toho sa niekedy dotkneme). Niektoré programy môžu byť neefektívne alebo proti istým programátorským zásadám.

Programy tu budeme písať tak, aby sme o nich vedeli čo najpriamejšie povedať, koľko možností vypíšu.

V materiáli používame hlavne jazyk Python, nakoľko programy sú v ňom stručné, zväčša pochopiteľné a tiež obsahuje pre nás užitočné knižničné funkcie. Programovať takéto výpisy však môžete v hocijakom jazyku. Pri jednoduchších úlohách na tom ani veľmi nezáleží. Ak by ste chceli programovať v C++, tak kombinatorické funkcie môžete nájsť:

- V ročníkovom projektu od Nadyia balanchuk: `https://davinci.fmph.uniba.sk/~balanchuk2/rp.html`
- V knižnici `https://github.com/mraggi/discreture` (neskúšal som ju)
- Alebo si len sami naprogramujte potrebné funkcie. Programovalo ich veľa ľudí pred vami, tak googlenie so správnymi heslami / LLM chatbot vám isto nejaké nájde.

Dohody

Usporiadané n-tice budeme stotožňovať s n-prvkovými postupnosťami a taktiež aj zo zobrazeniami z množiny $\{0, 1, \ldots, n-1\}$ (čo je aj pomerne štandardná definícia konečných postupností). Rovnaký objekt budeme tiež niekedy nazývať ako *slovo*, najmä ak pôjde o postupnosti písmen napr. anglickej abecedy. Karteziánsky súčin viacerých množín n množín budeme brať ako komutatívnu operáciu, výsledkom ktorej je množina usporiadaných n-tíc. Preto pri tomto zápise nebudeme uvádzať zátvorky´. Ak sa v karteziánskom súčine vyskytne viacero množín za sebou, tak vieme skrátiť zápis s využitím umocňovania. Teda $M^3 = M \times M \times M$. (To napokon sedí aj s dohodou, v ktorej M^3 značí množinu zobrazené z množiny $\{0, 1, 2\}$ do množiny M, teda $M^{\{0,1,2\}}$). Teda ak $M = \{1\}$, tak podľa tejto dohody platí

$$M \times M \times M = M^2 \times M = M \times M^2 = M^3 = \{(1, 1, 1)\}.$$

Avšak ak by sme pridali zátvorky, tak by sme dostali

$$M \times (M \times M) = \{(1, (1, 1))\} \neq \{((1, 1), 1)\} = (M \times M) \times M.$$

Úlohy sú zadané tak, že sa pýtajú na počet možností, nakoľko hlavným cieľom tohto materiálu je matematickými úvahami určiť počet možností. Všetky úlohy sú myslené tak, že máte napísať program, ktorý vypíše všetky možnosti a o ktorom aj viete tento počet možností zistiť bez toho, aby ste ho spustili. To zvyčajne znamená, že sa chcete vyhnúť podmienkam (if, while).

Materiál je do značnej miery predbežný. Ak si všimente nejaké chyby, nahláste ich, prosím, na `jozef.rajnik@fmph.uniba.sk`.

1 Základné kombinatorické pravidlá

1.1 Pravidlo súčtu a súčinu

V tejto kapitole budeme pre zjednodušenie považovať čísla tiež za slová. Napokon, keď ich program vypisuje na obrazovku, tak sa aj tak k nim správa.

Príklad 1.1. K dispozícii máme cifry $1, 3, 4, 9$. Koľko z nich vieme zložiť

a) dvojciferných čísiel,

b) trojciferných čísiel,

c) štvorciferných čísiel,

Riešenie

a) 2-ciferné čísla

```
M = {1, 3, 4, 9}
for a in M:
    for b in M:
        print(a, b, sep='')
```

b) 3-ciferné čísla

```
M = {1, 3, 4, 9}
for a in M:
    for b in M:
        for c in M:
            print(a, b, c, sep='')
```

c) 4-ciferné čísla

```
M = {1, 3, 4, 9}
for a in M:
    for b in M:
        for c in M:
            for d in M:
                print(a, b, c, d, sep='')
```

Z týchto programov ľahko vypočítame, že vypíšu postupne 4^2, 4^3 a 4^4 možností. Vysvetlíme si to na podúlohe b). Príkaz `print` sa vykoná 4-krát vo for cykle `for c in M:`. Toto celé sa 4-krát zopakuje v cykle `for b in M:`, čo dá $4 \cdot 4 = 4^2$ opakovaní a ešte sa to 4-krát zopakuje cyklom `for a in M:`, čo nám dá celkovo $4^2 \cdot 4 = 4^3$ vypísaných možností.

Pravidlo súčinu v programovaní zodpovedá vnoreným for cyklom. Ak každý for cyklus má za každým rovnaký počet opakovaní, tak celkový počet opakovaní vypočítame súčinom týchto čiastkových hodnôt.

Príklad 1.2. Morzeova abeceda 1-znakové až 4-znakové slová zložené so znakov `.` a `-`. Koľko existuje takýchto slov?

Riešenie

```
M = {'.', '-'}
for a in M:
    print(a)
for a in M:
    for b in M:
        print(a + b)
for a in M:
    for b in M:
        for c in M:
            print(a + b + c)
for a in M:
    for b in M:
        for c in M:
            for d in M:
                print(a + b + c + d)
```

Tento program sa skladá zo štyroch častí, kde každá obsahuje niekoľko vnorených for cyklov. Prvá časť vypíše 2 možnosti, druhá časť 2^2 možností, tretia 2^3 možností a štvrtá z 2^4 možností. Ak chceme zistiť, koľko možností sa vypíše spolu, tak tieto hodnoty jednoducho sčítame.

Pravidlo súčtu v programovaní zodpovedá viacerým častiam programu zapísaným za sebou. Počet možností, ktoré nám program vypíše, dostaneme ako súčet možností vypísaných v jednotlivých častiach.

1.2 Funkcia na karteziánsky súčin

Písať veľké množstvo vnorených for cyklov nie je zrovna dobrý programátorský zvyk. Okrem toho je tu aj kus zdĺhavé a neprehľadné. Preto si ukážeme, ako si možno zjednodušiť použitie pravidla súčinu. Formálne pravidlo súčinu možno vyjadriť nasledovne.

Formálne možno pravidlo súčinu vyjadriť pomocou karteziánskeho súčinu množín.

Veta 1. *Nech $A_1, A_2, \ldots, A_n$ je ľubovoľných n množín, potom*

$$|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdot \cdots \cdot |A_n|.$$

Karteziánsky súčin nám reprezentuje množinu usporiadaných n-tíc, kde na prvé miesto môžeme vybrať ľubovoľný prvok z množiny A_1 (teda máme $|A_1|$ možností), ...

Pokiaľ sa nám nechce písať toľko for cyklov, tak práve táto matematická reprezentácia nám vie prísť vhod. Môžeme totiž využiť funkciu pre karteziánsky súčin `itertools.product`, ktorá bude iterovať cez všetky usporiadané n-tice karteziánbskeho súčinu n množín, ktoré zadáme ako parametre. Teda podúlohu 1d) vieme zapísať aj nasledovne:

```
from itertools import product

M = {1, 3, 4, 9}
for a, b, c, d in product(M, M, M, M):
    print(a, b, c, d, sep='')
```

V prípade, že robíme karteziánsky súčin tej istej množiny (ako teraz), tak nám stačí uviesť len jednu množiny a využiť voliteľný argument `repeat` na určenie počtu jej zopakovania.

```
for a, b, c, d in product(M, repeat=4):
    print(a, b, c, d, sep='')
```

Ak by sme chceli vypisovať naozaj usporiadané štvorice, tak nemusíme rozpisovať ich jednotlivé zložky a vieme ušetriť zopár znakov ešte takto:

```
for t in product(M, repeat=4):
    print(t)
```

Ak by sme naopak túto skratku chceli využiť na vypisovanie reťazcov (čo môžeme spraviť aj teraz, keďže medzi číslom a reťazcom v tejto úlohe nepotrebujeme rozlišovať), tak môžeme využiť metódu `''.join(t)`, ktorá reťazce v usporiadanej n-tici (resp. inom iterovateľnom objekte) spojí do jedného reťazca. Celý program by tak mohol vyzerať:

```
from itertools import product

M = {'1', '3', '4', '9'}
for t in product(M, repeat=4):
    print(''.join(t))
```

1.3 Využitie bijekcií – nie na všetko stačí len karteziánsky súčin

Len málo kombinatorických úloh je takých, kde premenné zo všetkých for cykloch len vypíšeme za sebou. Často potrebujeme vypisovať niečo iné. Ilustrujeme si to na tomto jednoduchom príklade.

Príklad 1.3. Koľko je 4-znakových slov zložených z písmen `U`, `K`, `T`, `G`, ktoré majú posledné dve písmená rovnaké?

Ľahko prídeme k takémuto programu.

```
M = {'U', 'K', 'T', 'G'}
for a in M:
    for b in M:
        for c in M:
            print(a + b + c + c)
```

Čo sa tu deje po formálnej stránke? Označme množinu hľadaných slov ako A. Naše tri for cykly v programe zodpovedajú množine $M \times M \times M = M^3$. Teraz by sme chceli ukázať, že takto sme dostali hľadaný počet možností, teda $|A| = |M^3|$. To formálne ukážeme tak, že nájdeme bijekciu $f\colon M^3 \to A$ (alebo opačným smerom). Tá bijekcia zjavne existuje a má predpis

$$f(a, b, c) = (a, b, c, c).$$

Porovnajte tento predpis s tým, čo máme v `print`-e.

Dôkaz, že ide o bijekciu je priamočiarky. Injektívnosť:

$$f(a,b,c) = f(d,e,f) \Rightarrow (a,b,c,c) = (d,e,f,f) \Rightarrow (a = d \wedge b = e \wedge c = f \wedge c = f) \Rightarrow (a,b,c) = (d,e,f).$$

Surjektívnosť: Každý prvok množiny A je tvaru (a,b,c,c) a dostaneme ho ako $f(a,b,c)$.

Ak chceme byť poriadne formálni, tak tento prístup vlastne používame pri všetkých úlohách, kde máme generovať čísla. Napr. úlohu 1.1 by sme vedeli naprogramovať aj takto.

```
M = {1, 3, 4, 9}
for a in M:
    for b in M:
        for c in M:
            print(100*a + 10*b + c)
```

Čiže tu využívame bijekciu $f\colon M^3 \to B$, kde B je množina hľadaných čísel, s predpisom $f(a,b,c) = 100a + 10b + c$.

Mať na pamäti tento aspekt riešenia kombinatorických úloh ale nie je iba o formálnosti. Práve táto časť býva jednou z najčastejších chýb, kvôli ktorým dostaneme zlý výsledok kombinatorickej úlohy. Pri zložitejších úlohách sa k týmto chybám dostaneme.

Na nasledovnom príklade si ukážeme dve veci. Prvou je sofistikovanejšie použitie pravidiel súčtu a súčinu – ide totiž o situáciu, kedy to nie je tak zjavné v zadaní. Vedieť si všetky možnosti rozdeliť na vhodné menšie časti (množiny), ideálne disjunktné, je dôležitým prvkom riešenia kombinatorických úloh. Druhou vecou je, že toto riešenie si zjednodušíme za použitia vhodnej bijekcie.

Príklad 1.4. Koľko je všetkých 4-znakových slov zložených z písmen `a`, `b` `c`, `d`, `e`, ktoré obsahujú práve jedno písmeno `a`.

Riešenie

Možnosti rozdelíme na prípady podľa toho, na ktorej pozícii sa písmeno `a` nachádza.

```
M = {'b', 'c', 'd', 'e'}
for a in M:
    for b in M:
        for c in M:
            print('a' + a + b + c)
for a in M:
    for b in M:
        for c in M:
            print(a + 'a' + b + c)
for a in M:
    for b in M:
        for c in M:
            print(a + b + 'a' + c)
for a in M:
    for b in M:
        for c in M:
            print(a + b + c + 'a')
```

Matematicky vieme túto množinu vyjadriť, ak označíme $M = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, ako

$$\{\mathtt{a}\} \times M^4 \cup M \times \{\mathtt{a}\} \times M^3 \cup M^2 \times \{\mathtt{a}\} \times M^2 \cup M^3 \times \{\mathtt{a}\} \times M \cup M^4 \times \{\mathtt{a}\}.$$

Počet prvkov tejto množiny je podľa pravidla súčinu a súčtu

$$4^4 + 4^4 + 4^4 + 4^4 + 4^4 = 5 \cdot 4^4.$$

Avšak na programe si môžeme všimnúť, že sa dá priamočiaro zostručniť nasledovne:

```
M = {'b', 'c', 'd', 'e'}
for a in M:
    for b in M:
        for c in M:
            print('a' + a + b + c)
            print(a + 'a' + b + c)
            print(a + b + 'a' + c)
            print(a + b + c + 'a')
```

Prípadne ešte nasledovne

```
M = {'b', 'c', 'd', 'e'}
```

```
for a in M:
    for b in M:
        for c in M:
            for i in range(4):
                slovo = a + b + c
                print(slovo[:i] + 'a' + slovo[i:])
```

Tu sme najprv vygenerovali slovo dĺžky 3 bez písmen a a potom sme doňho a-čko vložili, na čo máme 4 možnosti. Toto zas zodpovedaná matematickému riešeniu, v ktorom zavedieme množinu $M^3 \times \{0, 1, 2, 3\}$, ktorá má $4^3 \cdot 4$ prvkov, a nájdeme bijekciu z tejto množiny na množinu zo zadania.

1.4 Pravidlo rozdielu alebo častá chyba pri riešení

Príklad 1.5. Koľko je všetkých 4-znakových slov zložených z písmen a, b c, d, ktoré obsahujú aspoň jedno písmeno a.

Pri úlohách tohto typu študenti často napíšu nasledovné riešenie:

Nesprávne riešenie

Najprv vyberieme pozíciu, na ktorej sa bude nachádzať písmeno a – na to máme 4 možnosti. Potom ostávajú 3 miesta, na ktorých môžu byť ľubovoľné písmená, na každom mieste máme na výber 4, teda 4^3 možností. Spolu tak máme $4 \cdot 4^3 = 4^4$ možností.

Isteže, problémom v tomto riešení je neformálnosť. Napokon, doposiaľ v tomto texte sme si ešte neukazovali, ako formálne matematicky vyjadriť riešenie takéhoto typu. Uvedieme preto aj formálnejšiu verziu. Pôjde vlastne o podobné riešenie ako pri príklade 1.4, len dovolíme aj na zvyšných miestach písmeno a.

Nesprávne riešenie

Nech $M = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$. Možnosti si rozdelíme podľa toho, na ktorej pozícii sa nachádza písmeno a. Teda množinu zo zadania vieme vyjadriť ako

$$\{\mathtt{a}\} \times M^4 \cup M \times \{\mathtt{a}\} \times M^3 \cup M^2 \times \{\mathtt{a}\} \times M^2 \cup M^3 \times \{\mathtt{a}\} \times M.$$

Podľa pravidla súčtu a súčinu tak vieme určiť počet prvkov tejto množiny ako:

$$4^3 + 4^3 + 4^3 + 4^3 = 4^3.$$

Keď máme riešenie napísané takto formálne, tak ste si skôr tu všimli, kde je problém – predsa len, z formálnejšieho zápisu to viac kričí. Ak však nie, tak sa pozrieme na to, ako by sme niečo takéto naprogramovali. Opäť sa inšpirujeme riešením príkladu 1.4.

```
M = {'a', 'b', 'c', 'd', 'e'}
for a in M:
    for b in M:
        for c in M:
            print('a' + a + b + c)
for a in M:
    for b in M:
        for c in M:
            print(a + 'a' + b + c)
for a in M:
    for b in M:
        for c in M:
            print(a + b + 'a' + c)
for a in M:
    for b in M:
        for c in M:
            print(a + b + c + 'a')
```

Keď sa pozriete poriadne na výstup tohto programu, tak by ste si mohli všimnúť, v čom nastal problém – napr. slovo abba sme vypísali 2-krát. Dokonca slovo aaaa až 4-krát! Toto je jedna z najčastejších chýb pri riešení enumeračných úloh – započítanie tej istej možnosti viackrát. Vo formálnom riešení je problém v tom, že sme použili pravidlo súčtu. To však nemôžeme, lebo uvedené štyri množiny nie sú disjunktné (napr. všetky obsahujú (a, a, a, a).

Ako teda na to ísť správne? Použiť pravidlo súčtu je fajn nápad, len si potrebujeme všetky možnosti rozdeliť na disjunktné množiny. Napr. jeden z priamočiarych nápadov je podľa počtu a-čok: môže byť jedno, dve, tri alebo štyri.

Všetky štyri prípady vieme vyriešiť podobne ako s len jedným a-čkom, teda v príklade 1.4. No toto riešenie je celkom zdĺhavé. Efektívnejšie je rozdeliť si možnosti podľa výskytu **prvého** písmena a.

Riešenie

Všetky možnosti si rozdelíme na 4 množiny podľa toho, či sa a prvýkrát objaví na 1., 2., 3. alebo 4. mieste. Tieto množiny sú zjavne disjunktné.

```
M = {'a', 'b', 'c', 'd', 'e'}
# Prve a je 1. mieste
for a in M:
    for b in M:
        for c in M:
            print('a' + a + b + c)
# Prve a je 2. mieste
for a in M - {'a'}:
    for b in M:
        for c in M:
            print(a + 'a' + b + c)
# Prve a je 3. mieste
for a in M - {'a'}:
    for b in M - {'a'}:
        for c in M:
            print(a + b + 'a' + c)
# Prve a je 4. mieste
for a in M - {'a'}:
    for b in M - {'a'}:
        for c in M - {'a'}:
            print(a + b + c + 'a')
```

Po matematickej stránke to zodpovedá riešeniu, kde si množinu možností vyjadríme ako

$$\{\mathtt{a}\} \times M^4 \cup (M - \{\mathtt{a}\}) \times \{\mathtt{a}\} \times M^3 \cup (M - \{\mathtt{a}\})^2 \times \{\mathtt{a}\} \times M^2 \cup (M - \{\mathtt{a}\})^3 \times \{\mathtt{a}\} \times M.$$

Najjednoduchším matematickým riešením však je riešenie založené na nasledovnej úvahe: Všetkých 4-znakových slov zložených z písmen a, b c, d, je 4^4. Slov dĺžky 4, ktoré neobsahujú žiadnea, je 3^4 (na každé zo štyroch miest máme teraz len 3 možnosti). Po vylúčení týchto slov nám zostanú práve 4-znakové slová, ktoré obsahujú aspoň jedno a, ktorých teda je $4^4 - 3^4$. Táto úvaha sa nazýva pravidlo rozdielu a formálne sa dá vyjadriť nasledovne.

Veta 2. *Nech A a B sú množiny, pre ktoré platí $B \subseteq A$. Potom $|A \setminus B| = |A| - |B|$.*

Pri jeho používaní si teda musíme dávať pozor na podmienku $B \subseteq A$, teda, že či všetky „zlé“ možnosti, ktoré odčítavame, sme naozaj započítali vo všetkých možnostiach. Formálne teda toto riešenie zapíšeme ako

Riešenie

Hľadanú množinu vyjadríme ako

$$M^4 \setminus (M - \{\mathtt{a}\})^4.$$

Keďže zjavne platí $(M - \{\mathtt{a}\})^4 \subseteq M^4$, tak podľa pravidla rozdielu a pravidla súčinu máme

$$|M^4 \setminus (M - \{\mathtt{a}\})^4| = |M^4| - |(M - \{\mathtt{a}\})^4| = 4^4 - 3^4.$$

Žiaľ, pravidlo rozdielu nemá nejakú peknú alternatívu pri programovaní. Jeden zo spôsobov, pri ktorom aj naozaj využijeme množinový rozdiel, je nasledovný:

```
M = {'a', 'b', 'c', 'd'}
vsetky = set()
for a in M:
    for b in M:
        for c in M:
            for d in M:
                vsetky.add(a + b + c + d)
zle = set()
for a in M - {'a'}:
    for b in M - {'a'}:
        for c in M - {'a'}:
            for d in M - {'a'}:
                zle.add(a + b + c + d)
for slovo in vsetky - zle:
    print(slovo)
```

Avšak z programu takéhoto typu nevieme ľahko určiť počet možností, ktoré sa vypíšu. Isteže, do množiny `vsetky` sa vloží 4^4 slov a do množiny `zle` sa vloží 3^4 slov. No zádrhely sú nasledovné:

- Množina `vsetky` môže obsahovať v skutočnosti menej ako 4^4 slov – mohlo sa nám stať, že sme nejaké rovnaké slovo vložili viackrát. Rovnako to môže byť aj s množinou `zle`.
- Ak aj množiny `vsetky` a `zle` budú obsahovať správny počet prvkov, stále sa nám môže vypísať viac ako $4^4 - 3^4$ slov – ak množina `zle` bude obsahovať nejaké slová, ktoré nie sú v množine `vsetky`.

Na tieto úskalia je vhodné pamätať, keď budeme písať programy tohto typu. Vysporiadať sa dá s tým pridaním vhodných príkazov `assert`:

```
M = {'a', 'b', 'c', 'd'}
vsetky = set()
for a in M:
    for b in M:
        for c in M:
            for d in M:
                moznost = a + b + c + d
                assert moznost not in vsetky
                vsetky.add(moznost)
zle = set()
for a in M - {'a'}:
    for b in M - {'a'}:
        for c in M - {'a'}:
            for d in M - {'a'}:
                moznost = a + b + c + d
                assert moznost not in zle
                assert moznost in vsetky
                zle.add(moznost)
for slovo in vsetky - zle:
    print(slovo)
```

Riadky 8 a 16 nám zaručujú, že ak by sme išli nejaké slovo pridať druhýkrát, tak program padne. Riadok 17 zas spôsobí pád programu, ak do množiny `zle` vložíme niečo, čo nie je v množine `vsetky`. Takto sme dostali program, ktorý, ak skončí bez chyby, vypíše $4^4 - 3^4$ možností.

1.5 Zovšeobecnené pravidlo súčinu

Teraz sa pozrieme na sofistikovanejšie použitie pravidla súčinu, ktoré si ukážeme na nasledovnej úlohe.

Príklad 1.6. Vypíšte všetky a) 2, b) 3, c) 4-ciferné čísla zložené z cifier $1, 2, 3, 4, 5$, v ktorých sa neopakujú cifry.

Dvojciferné čísla ľahko vypíšeme nasledovným programom, v ktorom možnosti vypisujeme v piatich skupinách podľa prvej cifry.

```
M = {1, 2, 3, 4, 5}
for b in M - {1}:
    print(1, b, sep='')
for b in M - {2}:
    print(2, b, sep='')
for b in M - {3}:
    print(3, b, sep='')
for b in M - {4}:
    print(4, b, sep='')
for b in M - {5}:
    print(5, b, sep='')
```

Matematicky to zodpovedá zjednoteniu piatich po dvoch disjunktných množín

$$\{1\} \times (M - \{1\}) \cup \{2\} \times (M - \{2\}) \cup \{3\} \times (M - \{3\}) \cup \{4\} \times (M - \{4\}) \cup \{5\} \times (M - \{5\}).$$

Tak ako máme zapísaný program, tak v ňom vieme identifikovať opakujúci sa vzor, na základe ktorého ho vieme zjednodušiť s použitím ďalšieho for cyklu na:

```
M = {1, 2, 3, 4, 5}
for a in M:
    for b in M - {a}:
        print(a, b, sep='')
```

Vieme jednoducho vypočítať, koľko možností vypíše takýto program? Áno! Prvý for cyklus sa zopakuje 5-krát a druhý for cyklus sa vždy zopakuje 4-krát a zakaždým vypíše jednu možnosť. Teda spolu dostaneme $5 \cdot 4 = 20$ možností. Použili sme vlastne rovnakú argumentáciu ako pri pravidle súčinu.

Trojciferné a štvociferné čísla vieme tak vyjadriť podobne. Trojciferných čísel dostaneme $5 \cdot 4 \cdot 3 = 60$:

```
M = {1, 2, 3, 4, 5}
for a in M:
    for b in M - {a}:
        for c in M - {a, b}:
            print(a, b, c, sep='')
```

A štvorciferných čísel máme $5 \cdot 4 \cdot 3 \cdot 2 = 120$:

```
M = {1, 2, 3, 4, 5}
for a in M:
    for b in M - {a}:
        for c in M - {a, b}:
            print(a, b, c, sep='')
```

Vo formálnej matematike sa tento princíp nazýva zovšeobecnené pravidlo súčinu. Problémom je, že oproti bežnému pravidlu súčinu sa nedá tak pekne presne vyjadriť. Rozmyslite si, že množina usporiadaných dvojíc prvkov z množiny $M = \{1, 2, 3, 4, 5\}$ s rôznymi prvkami sa nedá vyjadriť ako karteziánsky súčin dvoch množín. Prejdeme si cez rôzne vyjadrenia tohto pravidla.

Pre účely programovanie bude najdôležitejšie nasledovné vyjadrenie cez programy.

Veta 3. *Ak máme n vnorených for cyklov takých, že pre každé $i \in \{1, 2, \ldots, n\}$ i-ty for cyklus vykoná* ***zakaždým*** *a_i opakovaní, tak spolu tieto for cykly vykonajú $a_1 \cdot a_2 \cdot \cdots \cdot a_n$ opakovaní.*

Z matematického vyjadrenia uvedieme najprv zrozumiteľné, aj keď nie formálne najpresnejšie vyjadrenie.

Veta 4 (Zovšeobecnené pravidlo súčinu, neformálna verzia)**.** *Nech X je konečná množina. Nech $k \geq 2$ a nech A je množina usporiadaných n-tíc $(x_1, x_2, \ldots, x_k) \in X^k$, ktorá spĺňa podmienky:*

(1) *prvok x_1 je možné z množiny X vybrať n_1 spôsobmi;*

(2) *pre každé $i \in \{1, \ldots, k-1\}$, po akomkoľvek výbere usporiadanej i-tice $(x_1, x_2, \ldots, x_i)$ je možné prvok x_{i+1} vybrať vždy n_{i+1} spôsobmi.*

Potom $|A| = n_1 \cdot n_2 \cdot \cdots \cdot n_k$.

Dôležité však pre nás hlavne bude, ako toto pravidlo používať. V našom prípade vieme množinu všetkých 4-ciferných čísel z úlohy 1.6 vyjadriť ako

$$R = \left\{\overline{abcd} \mid a \in M \wedge b \in M - \{a\} \wedge c \in M - \{a, b\} \wedge d \in M - \{a, b, c\}\right\},$$

pričom platí

- $|M| = 5$;
- $|M - \{a\}| = 4$ pre každé a;
- $|M - \{a, b\}| = 3$ pre každé a, b;
- $|M - \{a, b, c\}| = 2$ pre každé a, b, c.

Preto na základe zovšeobecneného pravidla súčinu vieme povedať, že $|R| = 5 \cdot 4 \cdot 3 \cdot 2 = 120$. Takéto vyjadrenie aj priamo zodpovedá uvedenému programu pre výpis týchto čísel.

Ak by sme chceli vetu 4 formulovať úplne formálne, tak to vieme spraviť na štýl toho, ako vieme zapísať množinu všetkých 4-ciferných čísel, v ktorých sa cifry neopakujú:

$$\{(a, b, c, d);\ a \in C - \{0\}, b \in C - \{a\}, c \in C - \{a, b\}, d \in C - \{a, b, c\}\},$$

kde $C = \{0, 1, \ldots, 9\}$ je množina cifier. Prvky a, b, c, d tak neberieme z pevných množín ako pri pravidle súčinu, ale volíme ich z množiny, ktorá je určená voľbou predošlých prvkov. Napr. prvok c vyberáme z množiny $C - \{a, b\}$, ktorej konkrétna podoba závisí od výberu a, b, avšak nie od výberu d. (Pripomíname, že je dôležité, že jej počet prvkov nezávisí na tom, aké a, b zvolíme.) Množinu typu $C - \{a, b\}$ formálne vyjadríme ako množinu $M_{a,b}$, ktoré môže byť pre rôzne voľby a, b iná.

Veta 5 (Zovšeobecnené pravidlo súčinu). *Nech X, M sú konečné množiny, $k \geq 2$ a nech pre každú usporiadanú i-ticu $(x_1, x_2, \ldots, x_i) \in X^i$, kde $i \in \{1, \ldots, k-1\}$, máme množinu $M_{x_1,\ldots,x_i}$, pričom sú splnené podmienky::*

(1) $|M| = n_1$;

(2) *pre každé $i \in \{1, \ldots, k-1\}$ a každú usporiadanú i-ticu $(x_1, x_2, \ldots, x_i) \in X^i$ platí* $|M_{x_1,\ldots,x_i}| = n_{i+1}$.

Potom

$$|\{(x_1, x_2, \ldots, x_k);\ x_1 \in M, (\forall i \in \{1, 2, \ldots, k-1\})(x_{i+1} \in M_{x_1,\ldots,x_i})\}| = n_1 \cdot n_2 \cdot \cdots \cdot n_k.$$

Zovšeobecnené pravilo súčino vieme použiť aj v prípade, ak počítame prvky karteziánskeho súčinu rôznych množín $X_1 \times X_2 \times \cdots \times X_k$. Vtedy nám stačí zvoliť vo vete $X = X_1 \cup X_2 \cup \cdots \cup X_k$.

Príklad 1.7. Koľko existuje 3-ciferných čísel zložených z cifier z množiny $M = \{1, 2, 3, 4, 5\}$, v ktorých je posledná cifra rôzna od prvých dvoch?

Nesprávne riešenie

Na výber prvej cifry a a aj druhej cifry b máme 5 možností. Tretia cifra musí byť rôzna od prvých dvoch, teda ju vyberáme z $M - \{a, b\}$, čo sú 3 možnosti. Hľadaných čísel je teda $5 \cdot 5 \cdot 3 = 75$. Toto riešenie zodpovedá programu.

```
M = {1, 2, 3, 4, 5}
for a in M:
    for b in M:
        for c in M - {a, b}:
            print(a, b, c, sep='')
```

Uvedený program je naozaj správny a vypíše správne možnosti. Len ich nevypíše 75. Problém je v tom, že množina $M - \{a, b\}$ nemá vždy 3 prvky – ak $a = b$, tak má až 4 prvky. Nie je tu teda splnené podmienka zovšeobecneného pravidla súčinu, kde musí platiť $|M - \{a, b\}| = 3$ pre **každú** voľbu a, b.

Zovšeobecnené pravidlo súčinu však stále vieme použiť, len trochu inak. Stačí, ak najprv vyberieme poslednú cifru.

Riešenie

Zvolíme poslednú cifru c z množiny M – 5 možností. Potom zvolíme prvé dve cifry $a, b \in M - \{c\}$ – na každú z nich máme 4 možnosti. Podľa zovšeobecneného pravidla súčinu tak máme spolu $5 \cdot 4 \cdot 4 = 80$ možností.

```
M = {1, 2, 3, 4, 5}
for c in M:
    for a in M - {c}:
        for b in M - {c}:
            print(a, b, c, sep='')
```

Problémom zovšeobecneného pravidla súčinu je, že nemá veľmi príjemnú formálnu podobu. No a preto aj ťažko napíšeme nejakú funkciu, ktorá by nám jeho použitie uľahčovala podobne ako funkcia `itertools.product`. Našťastie, na najčastejšie používané účely funkcie máme, o čom práve je ďalšia kapitola.

1.6 Úlohy na precvičenie

Úloha 1.1. Koľko je všetkých 4-ciferných čísiel zložených z cifier 0, 1, 2, 4, 7?

Úloha 1.2. Koľko je všetkých 6-znakových slov zložených z písmen `m`, `f`, `i`.

Úloha 1.3. Koľko existuje 3- až 5-ciferných čísel zložených z cifier 0, 2, 4, 7?

Úloha 1.4. Koľko existuje všetkých usporiadaných 5-tíc zložených z písmen $\{a, b, c, d\}$, ktoré obsahujú dva po sebe idúce výskyty písmena b a žiaden ďalší výskyt písmena b?

Úloha 1.5. Koľko existuje všetkých usporiadaných 5-tíc zložených z písmen $\{a, b, c, d\}$, ktoré obsahujú práve dve písmena b (nie nutne za sebou)?

Úloha 1.6. Koľko je párnych 5-ciferných čísiel zložených z cifier $1, 2, 3, 5, 8, 9$, v ktorých sú prvé dve cifry rovnaké.

Úloha 1.7. Koľko je všetkých 5-ciferných čísel, ktoré obsahujú práve jednu párnu cifru?

Úloha 1.8. Určte počet kladných deliteľov čísla 120.

Úloha 1.9. Určte počet kladných deliteľov čísla $3^5 \cdot 5^4 \cdot 7^2 \cdot 11^8$.

Úloha 1.10. Určte počet kladných deliteľov čísla $3^4 \cdot 4^5 \cdot 6^2 \cdot 7^6$.

Úloha 1.11. Koľko existuje 4-ciferných čísel zložených z cifier z množiny $M = \{1, 2, 3, 4\}$, v ktorých sa nenachádzajú dve rovnaké cifry bezprostredne za sebou?

Úloha 1.12. Koľko existuje všetkých postupností dĺžky 5 zložených z písmen $\{a, b, c, d\}$, ktoré obsahujú každé z písmen aspoň raz?

Úloha 1.13. Koľkými spôsobmi možno prvky množiny $M = \{1, 2, 3, 4, 5\}$ zoradiť do usporiadanej 5-tice tak, aby sa číslo 1 nachádzalo pred číslom 2 (nie nutne bez prostredne)?

2 Variácie a permutácie

Úloha 2.1. Koľko existuje všetkých 7-znakových slov zložených z písmen z množiny $M = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}\}$?

Úlohu tohto typu sme už riešili a teraz by ste ju bez problémov mali vedieť vyriešiť. Podľa zovšeobecneného pravidla súčinu nám vyjde $10 \cdot 9 \cdot \cdots \cdot 4 = 10^{\underline{7}} = 604\,800$ možností. Vypísať by sme ich vedeli takýmto programom.

```
M = set('abcdefghij')
for a in M:
    for b in M - {a}:
        for c in M - {a, b}:
            for d in M - {a, b, c}:
                for e in M - {a, b, c, d}:
                    for f in M - {a, b, c, d, e}:
                        for g in M - {a, b, c, d, e, f}:
                            print(a, b, c, d, e, f, g, sep='')
```

Avšak takýto program je na jednej strane celkom dlhý a kus neprehľadný s toľkým osadením. Na druhej strane, nevieme v ňom jednoducho meniť (ideálne hodnotou jednej premennej), koľko znakov majú obsahovať vypisované slová – musíme na to pridávať / odoberať for cykly.

S podobnými situáciami sa pri programovaní kombinatorických úloh budeme stretávať často. No a čo spraví programátor? Tieto časté situácie si pomenuje a napíše si k nim funkcie, ktoré môže využívať kedykoľvek niekde objaví príslušnú pomenovanú situáciu. Preto sa často v kombinatorike, či už takejto programovacej alebo tej počítacej, často stretneme s pomenovanými typmi kombinatorických konfigurácií. Teraz sa zoznámime s tými najjednoduchšími – variáciami.

V nasledujúcich častiach uvedieme len najdôležitejšie veci pre prácu so zavedenými pojmami. Potom v samostatnej časti sa detailne povenujeme tomu, ako sú spomenuté funkcie implementované, ako možno spomenuté tvrdenia formálne dokázať a ako tieto dve veci spolu súvisia.

2.1 Variácie s opakovaním

Definícia 1. Nech M je konečná množina. *Variáciou s opakovaním k-tej triedy z prvkov množiny M* nazývame ľubovoľnú usporiadanú n-ticu prvkov z M, teda ľubovoľný prvok množiny M^k.

Alternatívne možno variácie s opakovaním definovať ako zobrazenia z množiny $\{1, 2, \ldots, k\}$ do množiny M. Pri programovaní sa však s nimi stretneme ako s usporiadanými n-ticami, prípadne postupnosťami.

Veta 6. *Nech M je n-prvková množina. Počet všetkých variácií s opakovaním k-tej triedy z prvkov množiny M je*

$$n^k.$$

Platnosť tejto vety priamo vyplýva z pravidla súčinu. Na základe toho vieme všetky variácie s opakovaním k-tej triedy množiny M prejsť cyklom:

```
from itertools import product

for variacia in product(M, repeat=k):
```

2.2 Variácie bez opakovania a permutácie

Definícia 2. Nech M je konečná množina. *Variáciou bez opakovania k-tej triedy z prvkov množiny M* nazývame ľubovoľnú usporiadanú n-ticu prvkov z M, v ktorej sú každé dva prvky rôzne.

V prípade variácií bez opakovania je definícia cez zobrazenia jednoduchšia – ide o injektívne zobrazenia $\{1, 2, \ldots, k\} \to M$.

Veta 7. *Nech M je n-prvková množina. Počet všetkých variácií bez opakovania k-tej triedy z prvkov množiny M je*

$$n^{\underline{k}} = \underbrace{n \cdot (n-1) \cdot \cdots \cdot (n-k+1)}_{k \text{ členov}}.$$

Dôkaz. Všetky variácie bez opakovania k-tej triedy z n-prvkovej množiny M tvoria množinu usporiadaných k-tic $(a_1, a_2, \ldots, a_k) \in M^k$ takú, že

- $a_1 \in M$ – teda pre a_1 máme n možností;
- pre každé $i \in \{2, 3, \ldots, k\}$ volíme $a_i \in M - \{a_1, a_2, \ldots, a_{i-1}\}$, čo pre každú voľbu $a_1, a_2, \ldots, a_{i-1}$ dáva $n - i$ možností, nakoľko $a_1, a_2, \ldots, a_{i-1}$ sú navzájom rôzne.

Preto podľa zovšeobecneného pravidla súčinu je počet variácií bez opakovania $n(n-1)\ldots(n-k+1) = n^{\underline{k}}$. □

Špeciálnym prípadom variácií bez opakovania sú permutácie.

Definícia 3. Nech M je n-prvková množina. Permutáciou množiny M nazývame ľubovoľnú variáciu bez opakovania n-tej triedy prvkov množiny M, teda ľubovoľnú usporiadanú n-ticu, ktorá obsahuje každý prvok množiny M práve raz.

Ekvivalente možno permutácie definovať ako bijekcie $M \to M$.

Podľa vety 7 je počet všetkých permutácií n-prvkovej množiny n rovný $n^{\underline{n}} = n \cdot (n-1) \cdot \cdots \cdot 1 = n!$.

V Pythone vieme iterovať cez všetky permutácie množiny (resp. ľubovoľného iterovateľného objektu) M s využitím funkcie `itertools.permutations`

```
from itertools import permutations

for p in permutations(M):
```

Táto funkcia sa dá využiť aj na variácie s opakovaním k-tej triedy, hoci využitie tohto názvy úplne nekorešponduje s matematickou terminológiou. Stačí nastaviť dobrovoľný argument `r` na hodnotu k.

```
from itertools import permutations

for p in permutations(M, r=k):
```

2.3 Implementácia a dôkaz pre variácie s opakovaním

UNDER CONSTRUCTION

2.4 Implementácia a dôkaz pre variácie bez opakovania

UNDER CONSTRUCTION Obe funckie sú však pomerne známe programátorské cvičenia a dajú sa spraviť rekurzívne. Dokonca sa dajú spraviť tak, aby rekurzívne funkcie priamo zodpovedali dôkazom matematickou indukciou na prednáškach. Rekurzii, príp. inému opakovaniu, sa tu len tak bez problémov nevyhneme. Preto podobne sa nevyhneme použitiu matematickej indukcie na dôkaz týchto viet. Maximálne ju vieme skryť do nejakého iného tvrdenia.

Riešenia

Riešenie 1.1.

```
M = {0, 1, 2, 4, 7}
for a in M - {0}:
    for b in M:
        for c in M:
            for d in M:
                print(a, b, c, d, sep='')
```

Riešenie 1.2.

```
for a in 'mfi':
    for b in 'mfi':
        for c in 'mfi':
            for d in 'mfi':
                for e in 'mfi':
                    for f in 'mfi':
                        print(a + b + c + d + e + f)
```

Riešenie 1.3.

```
M = {0, 2, 4, 7}
for a in M - {0}:
    for b in M:
        for c in M:
            print(a, b, c, sep='')
for a in M - {0}:
    for b in M:
        for c in M:
            for d in M:
                print(a, b, c, d, sep='')
for a in M - {0}:
    for b in M:
        for c in M:
            for d in M:
                for e in M:
                    print(a, b, c, d, e, sep='')
```

Riešenie 1.4. Vyberieme tri písmená, ktoré nie sú `b`. Potom máme 4 možnosti, ako medzi ne vieme vsunúť `bb`.

```
M = {'a', 'c', 'd'}
for x in M:
    for y in M:
        for z in M:
            print('bb' + x + y + z)
            print(x + 'bb' + y + z)
            print(x + y + 'bb' + z)
            print(x + y + z + 'bb')
```

Alebo najskôr vyberieme pozíciu `bb`, ktorá môže byť od 0 do 3 (4 možnosti) a potom vyberieme zvyšné písmená.

```
M = {'a', 'c', 'd'}
for i in range(4):
    for x in M:
        for y in M:
            for z in M:
                slovo = x + y + z
                print(slovo[:i] + 'bb' + slovo[i:])
```

Riešenie 1.5. Podobne ako predošlá úloha, len tu máme 10 možností, ako môžu byť rozmiestnené dve *b*-čka. Nie je náročné tieto možnosti ručne vygenerovať. Neskôr si ukážeme, ako to spraviť všeobecne.

```
M = {'a', 'c', 'd'}
for x in M:
    for y in M:
        for z in M:
            print('bb' + x + y + z)
            print('b' + x + 'b' + y + z)
            print('b' + x + y + 'b' + z)
            print('b' + x + y + z + 'b')
            print(x + 'bb' + y + z)
            print(x + 'b' + y + 'b' + z)
            print(x + 'b' + y + z + 'b')
```

```
            print(x + y + 'bb' + z)
            print(x + y + 'b' + z + 'b')
            print(x + y + z + 'bb')
```

Riešenie 1.6.

```
M = {1, 2, 3, 5, 8, 9}
for a in M:
    for c in M:
        for d in M:
            for e in {2, 8}:
                print(a, a, c, d, e, sep='')
```

Riešenie 1.7.

```
from itertools import product
P = {0, 2, 4, 6, 8}
N = {1, 3, 5, 7, 9}
# Ak je parna cifra prva
for p in P - {0}:
    for a, b, c, d in product(N, repeat=4):
        print(p, a, b, c, d, sep='')
# Ak parna cifra nie je prva
for p in P:
    for a, b, c, d in product(N, repeat=4):
        print(a, p, b, c, d, sep='')
        print(a, b, p, c, d, sep='')
        print(a, b, c, p, d, sep='')
        print(a, b, c, d, p, sep='')
```

Riešenie 1.8.

```
for a in range(4):
    for b in range(2):
        for c in range(2):
            print(2**a * 3**b * 5**c)
```

Riešenie 1.9.

```
for a in range(6):
    for b in range(5):
        for c in range(3):
            for d in range(9):
                print(3**a * 5**b * 7**c * 11**d)
```

Riešenie 1.10. Tu si treba dať pozor na prvočíselný rozklad

$$3^4 \cdot 4^5 \cdot 6^2 \cdot 7^6 = 2^{12} \cdot 3^6 \cdot 7^6.$$

```
for a in range(13):
    for b in range(7):
        for c in range(7):
                print(2**a * 3**b * 7**c)
```

Riešenie 1.11.

```
M = {1, 2, 3, 4}
for a in M:
    for b in M - {a}:
        for c in M - {b}:
            for d in M - {c}:
                print(a, b, c, d, sep='')
```

Riešenie 1.12.

```
M = {'a', 'b', 'c', 'd'}
# Vyberiem, ktore pismeno bude prave dvakrat (4 moznosti)
for d in M:
    # Vyberiem, na ktorych poziciah bude (10 moznosti)
    for i, j in (0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4):
        # Vyberiem zvysne tri pismena, bez opakovania
        for x in M - {d}: # 3 moznosti
            for y in M - {d, x}: # 2 moznosti
```

```
            for z in M - {d, x, y}: # 1 moznost
                pozicie = [0, 1, 2, 3, 4]
                pozicie.remove(i)
                pozicie.remove(j)
                slovo = [d] * 5
                slovo[pozicie[0]] = x
                slovo[pozicie[1]] = y
                slovo[pozicie[2]] = z
                print(''.join(slovo))
```

Riešenie 1.6.

```
M = {2, 3, 4, 5}
# 1 je na 1. mieste (4*3*2*1 moznosti)
for a in M:
    for b in M - {a}:
        for c in M - {a, b}:
            for d in M - {a, b, c}:
                print(1, a, b, c, d, sep='')
# 1 je na 2. mieste (3*3*2*1 moznosti)
for a in M - {2}:
    for b in M - {a}:
        for c in M - {a, b}:
            for d in M - {a, b, c}:
                print(a, 1, b, c, d, sep='')
# 1 je na 3. mieste (3*2*2*1 moznosti)
for a in M - {2}:
    for b in M - {2, a}:
        for c in M - {a, b}:
            for d in M - {a, b, c}:
                print(a, b, 1, c, d, sep='')
# 1 je na 4. mieste (3*2*1*1 moznosti)
for a in M - {2}:
    for b in M - {2, a}:
        for c in M - {2, a, b}:
            for d in M - {a, b, c}:
                print(a, b, c, 1, d, sep='')
# 1 je na 5. mieste - vtedy nemame kam dat 2-ku (0 moznosti)
```