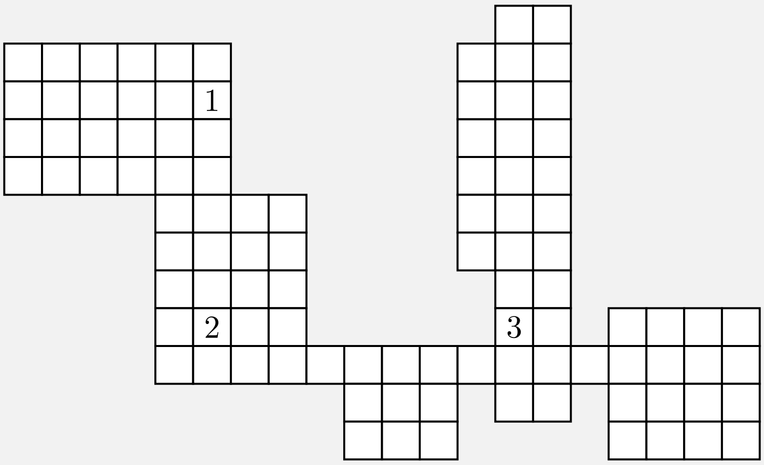
Riešenia 2. sady domácich úloh

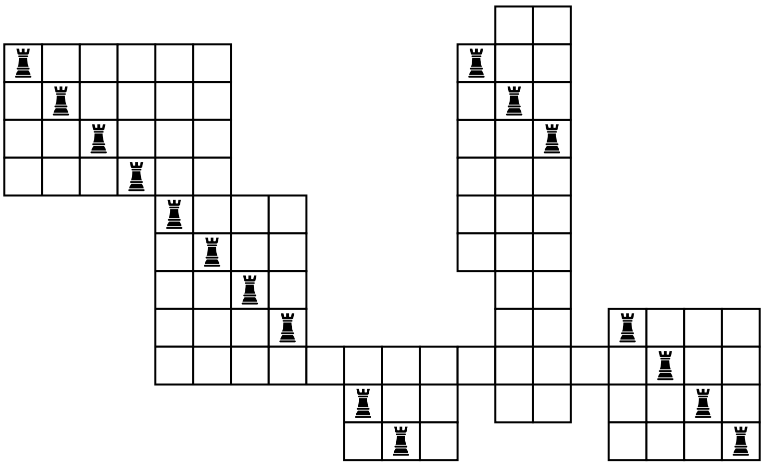
Úloha 1

Uvažujme plán MatFyzu ako na obrázku nižšie. Koľko najviac veží vieme umiestniť na políčka tohto plánu tak, aby sa žiadne dve neohrozovali? Vaše tvrdenie zdôvodnite. Veža ohrozuje políčka, na ktoré sa vie pohnúť tak, že ostane vo svojom riadku alebo stĺpci a pritom nevyjde mimo MatFyz (teda veže 1 a 2 sa ohrozujú, ale 2 a 3 nie).

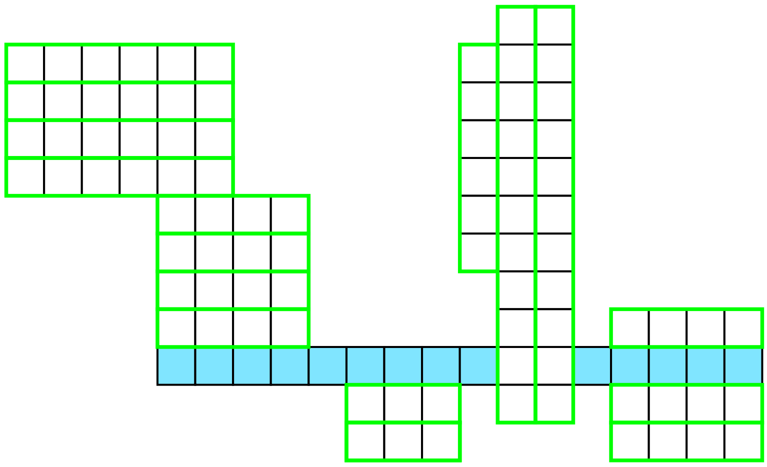


Ukážeme, že najviac možno umiestniť 17 veží.

Konštrukcia Na obrázku je umiestnených na pláne 17 veží, o ktorých ľahko skontrolujeme, že žiadne dve z nich sa neohrozujú.



Odhad MatFyz si rozdelíme na 17 oblastí ako na obrázku – 16 z nich tvorí zelený obdĺžnik a jedna je tvorená modrými políčkami. O každej z oblastí platí, že ak do nej ľubovoľne umiestnime dve veže, tak sa budú ohrozovať – oblasti sú totiž tvorená riadkami alebo stĺpcami, resp. ich časťami. Ak umiestnime aspoň 18 veží, tak z Dirichletovho princípu existuje oblasť, ktorá obsahuje aspoň dve veže – a tie sa ohrozujú. Preto nemôžeme mať viac ako 17 veží.



Úloha 2

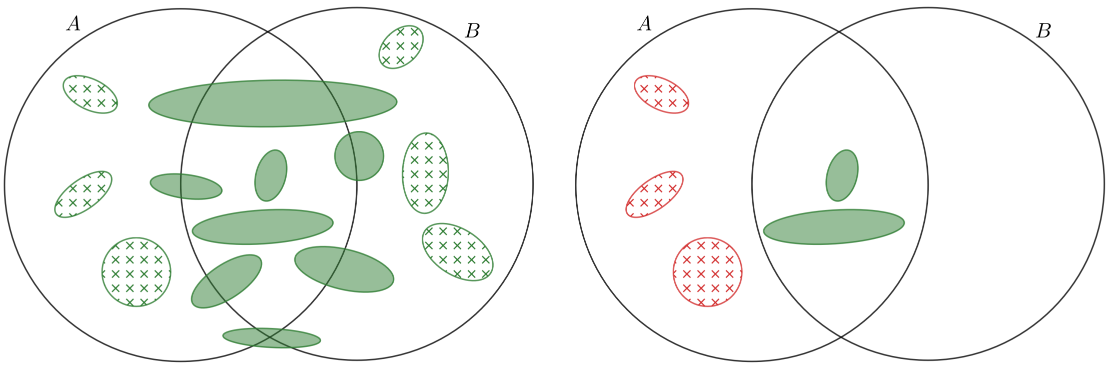
Rozhodnite a následne dokážte, či pre ľubovoľné dve množiny A, B platí:

a) $\mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A)) \subseteq \mathcal{P}(A \cap B) - \mathcal{P}(A - B)$

b) $\mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A)) \supseteq \mathcal{P}(A \cap B) - \mathcal{P}(A - B)$

Intuícia

Najskôr sa intuitívne zamyslime, ako vyzerajú podmnožiny, ktoré sa môžu vyskytnúť na ľavej a pravej strane.



Na ľavej strane máme na začiatku podmnožiny $A \cup B$ (označené zelenou), pričom odstránime také, ktoré sa celé zmestia do $A - B$ alebo $B - A$ (vyplnené krížikmi). Vyzerá to preto tak, že zostanú len také, ktoré majú prvky v prieniku $A \cap B$ alebo v aspoň dvoch „regiónoch“ diagramu.

Podobne vieme urobiť úvahy pre pravú stranu. Pôvodne sú v nej podmnožiny $A \cap B$ (označené zelenou), a potom odstránime také, ktoré sa celé zmestia do $A - B$ (vyplnené krížikmi). Vyzerá to tak, že sme žiadne podmnožiny neodstránili, lebo sme odstraňovali z „regiónu“, v ktorom sme pôvodne žiadne podmnožiny nemali.

Preto to vyzerá tak, že a) vo všeobecnosti nebude platiť a b) bude. Vieme teda, čo máme ísť dokazovať.

To, čo je uvedené v predošlých odsekoch, však nie je dôkaz. Dôvodom je, že vysvetlenia sú veľmi vágne naformulované, preto ľahko môžeme prehliadnuť nejaké podmnožiny. Napríklad by sme sa mohli dovtípiť, že $\mathcal{P}(A \cap B) - \mathcal{P}(A - B) = \mathcal{P}(A \cap B)$. To však neplatí, lebo každá potenčná množina obsahuje aj prázdnu množinu – a teda prázdna množina nebude prvkom ľavej strany, no bude prvkom pravej. Dôvod, prečo tvrdenia dokazujeme formálne, je, že nás to dosť často uchráni od takýchto chýb.

Podúloha a)

Keď chceme ukázať, že nejaká inklúzia neplatí vo všeobecnosti, stačí nájsť dve množiny A, B, pre ktoré neplatí. Z obrázkov v intuitívnych úvahách sa javí, že ak budeme mať nejaké prvky v $A \cap B$ aj v $A - B$, tak vzniknú nejaké podmnožiny, ktoré budú iba na ľavej strane. Skúsme si teda zobrať $A = \{1, 2\}$, $B = \{2\}$. Potom

$$\mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A)) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} - (\{\emptyset, \{1\}\} \cup \{\emptyset\}) = \{\{2\}, \{1, 2\}\},$$
$$\mathcal{P}(A \cap B) - \mathcal{P}(A - B) = \{\emptyset, \{2\}\} - \{\emptyset, \{1\}\} = \{\{2\}\}.$$

Vidíme, že pre tieto dve množiny $\mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A)) \not\subseteq \mathcal{P}(A \cap B) - \mathcal{P}(A - B)$, preto táto inklúzia neplatí. Všimnime si ešte, že s druhou inklúziou zatiaľ problém nemáme.

Podúloha b)

Túto inklúziu by sme chceli dokázať. Na to by sme pre všetky množiny chceli vedieť, že ak $X \in \mathcal{P}(A \cap B) - \mathcal{P}(A - B)$, tak aj $X \in \mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A))$. Pri takýchto úlohách je fajn si najskôr z definícií (t. j. ekvivalentnými úpravami) prepísať tieto podmienky. Poďme teda na to.

$$\begin{aligned}
&X \in \mathcal{P}(A \cap B) - \mathcal{P}(A - B), && \\
&X \in \mathcal{P}(A \cap B) \wedge \neg(X \in \mathcal{P}(A - B)), && \text{(definícia množinového rozdielu)} \\
&X \subseteq A \cap B \wedge \neg(X \subseteq A - B), && \text{(definícia potenčnej množiny)} \\
&(\forall x \in X : x \in A \cap B) \wedge \neg(\forall x \in X : x \in A - B), && \text{(definícia podmnožiny)} \\
&(\forall x \in X : x \in A \cap B) \wedge (\exists x \in X : \neg(x \in A - B)), && \text{(negácia kvantifikovaného výroku)} \\
&(\forall x \in X : (x \in A \wedge x \in B)) \wedge (\exists x \in X : \neg(x \in A \wedge x \notin B)), && \text{(definície množinových operácií)} \\
&(\forall x \in X : (x \in A \wedge x \in B)) \wedge (\exists x \in X : (x \notin A \vee x \in B)). && \text{(De Morganov zákon)}
\end{aligned}$$

Všimnite si, že sme nerobili žiadne myšlienkové úvahy, išlo iba o priamočiaru aplikáciu definícií a pravidiel na zjednodušovanie negácií. Keď budeme podobne upravovať druhú podmienku (vyskúšajte si, je to dobré cvičenie), dostaneme, že

$$X \in \mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A))$$
$$\Leftrightarrow$$
$$(\forall x \in X : (x \in A \vee x \in B)) \wedge (\exists x \in X : (x \notin A \vee x \in B)) \wedge (\exists x \in X : (x \notin B \vee x \in A)).$$

Chceme teda z predpokladu

$$\underbrace{(\forall x \in X : (x \in A \wedge x \in B))}_{\text{(P1)}} \wedge \underbrace{(\exists x \in X : (x \notin A \vee x \in B)))}_{\text{(P2)}}$$

dokázať záver

$$\underbrace{(\forall x \in X : (x \in A \vee x \in B))}_{\text{(Z1)}} \wedge \underbrace{(\exists x \in X : (x \notin A \vee x \in B))}_{\text{(Z2)}} \wedge \underbrace{(\exists x \in X : (x \notin B \vee x \in A))}_{\text{(Z3)}}.$$

Všimnime si, že (Z1) priamo vyplýva z (P1), lebo keďže všetky prvky musia patriť do A aj B, tak máme splnené pre ľubovoľný prvok obe strany disjunkcie (Z1) a na jej pravdivosť stačí splnenie jednej z nich.

Ďalej, (Z2) je to isté ako (P2), tým tento záver máme dokázaný priamo v predpoklade.

A nakoniec treba dokázať (Z3). Z (P2) vieme, že X nemôže byť prázdna. Z (P1) vieme, že všetky prvky X musia patriť do A. Preto určite máme prvok z X, ktorý patrí do A. A tým už máme pre nejaký prvok splnenú pravú stranu disjunkcie (Z3), čím je pravdivá celá.

Podarilo sa nám teda dokázať implikáciu

$$(X \in \mathcal{P}(A \cap B) - \mathcal{P}(A - B)) \Rightarrow (X \in \mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A))),$$

pričom tento dôkaz bol nezávislý od voľby X, a preto máme aj

$$\mathcal{P}(A \cap B) - \mathcal{P}(A - B) \subseteq \mathcal{P}(A \cup B) - (\mathcal{P}(A - B) \cup \mathcal{P}(B - A)),$$

ako sme chceli.

Úloha 3

Zistite, či pre ľubovoľné množiny A, B, C platí: Určte, koľko je všetkých

a) 3-ciferných čísel, ktorých prvá cifra (na mieste stoviek) je deliteľná tromi, druhá cifra je deliteľná štyrmi a posledná cifra je deliteľná piatimi;

b) 4-ciferných čísel zložených z cifier 2, 4, 7, 9, v ktorých sa cifry neopakujú a cifra 4 sa nachádza v nich skôr ako cifra 7;

c) 6-ciferných čísel zložených z cifier 1 a 7, ktoré neobsahujú tri jednotky po sebe.

Ku každej podúlohe vypíšte všetky možnosti. Zdôvodnite (stačí neformálne), prečo sú vaše riešenia správne (teda že ste každú možnosť započítali práve raz).

Bonus 1. *(1 bod)* V úlohe 3 možno získať bonusový bod, keď ručné vypísanie možností nahradíte vypísaním možností pomocou počítačového programu. Pre získanie celého bodu treba:

- zdôvodniť, že program vypísal možnosti správne (pokiaľ je to priamočiare, tak stačí stručné zdôvodnenie);
- zdôvodniť počet možností, ktoré program vypíše, bez toho, aby ste ho spustili.

V programe nesmiete využívať funkcie na generovanie možností (ako napr. knižnica `itertools` v Pythone). Body za samotný program budeme udeľovať podľa toho, ako náročné je odôvodniť potrebné veci (napr. zvyčajne je lepšie nepoužívať `if`).

Na začiatok poznamenáme, že výpis možností (či už ručne alebo programovaním) ani tak nebol samostatnou úlohou, hoci niektorí z vás to tak brali, ako bolo vidno z vašich riešení. Tento výpis mohol slúžiť na kontrolu vášho riešenia, nielen výsledku, ale aj postupu. Za účelom takejto kontroly je vhodné zvoliť spôsob výpisu, ktorý čo najviac zodpovedá nášmu „matematickému“ riešeniu. Tak aj spravíme v týchto riešeniach (hoci nie úplne všetko vo všetkých). Niektoré výpisy riešení budú prehnane detailné, to však slúži na to, aby ste lepšie na nich pochopili použité kombinatorické koncepty.

Bodovanie. Body za jednotlivé časti boli: a) 0,4b, b) 0,8b, c) 0,8b. Pri bonusoch 1 a 3: a) 0,2b, b) 0,4b, c) 0,4b.

Časť a)

Na prvé miesto si vyberáme z 3 možností: 3, 6, 9. Na druhé miesto z 3 možností: 0, 4, 8. Na tretie miesto z 2 možností: 0, 5. Celkovo máme tak $3 \cdot 3 \cdot 2 = 18$ možností.

Program. Tento spôsob riešenia zodpovedá nasledujúcemu programu.

```
for a in ['3', '6', '9']:
  for b in ['0', '4', '8']:
     for c in ['0', '5']:
        print(a + b + c)
```

Ľahko vidíme, že prvý cyklus sa zopakuje 3-krát, druhý 3-krát a tretí 2-kráte, teda príkaz `print` sa vykoná $3 \cdot 3 \cdot 2 = 18$-krát. Toľko program vypíše možností

Časť b) cez pozíciu štvorky

Možnosti rozdelíme na prípady podľa pozície štvorky:

1. 4 na 1. mieste: potom na druhé miesto máme 3 možnosti (okrem 4), na tretie miesto 2 možnosti (okrem 4 a prvého čísla) a na štvrté miesto 1 možnosť (číslo, čo nám ostalo). Máme teda $3 \cdot 2 \cdot 1 = 6$ možností.

2. 4 na 2. mieste: Na prvom mieste máme 2 možnosti (2 a 9), na treťom 2 možnosti (2, 7, 9 okrem prvého použitého čísla) a na posledné nám ostane 1 možnosť. Teda $2 \cdot 2 \cdot 1 = 4$ možnosti.

3. 4 na 3. mieste: na prvé miesto máme 2 možnosti (2 a 9), na druhé len jednu (to, čo ostane z 2 a 9) a na posledné musí ísť cifra 7. Teda $2 \cdot 1 \cdot 1 = 2$ možnosti.

Keďže jednotlivé prípady sú disjunktné, tak spolu máme $6 + 4 + 2 = 12$ možností.

Vypísanie možností

4279						
4297		2479				
4729		2497		2947		
4792		9427		9247		
4927		9472				
4972						
6	+	4	+	2	= 12 možností	

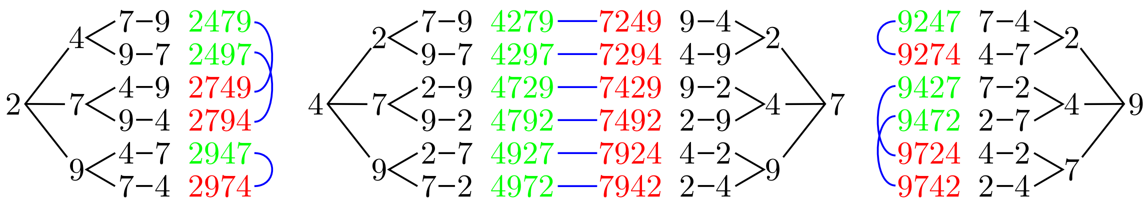
Program Tento spôsob vypísania priamočiaro zodpovedá programu:

```
cifry = {2, 7, 9}
# 4 je na 1. mieste
for b in cifry: # 3-krat sa zopakuje
   for c in cifry - {b}: # 2-krat sa zopakuje
      for d in cifry - {b, c}: # raz sa zopakuje
         # Vypise 3 * 2 * 1 = 6 moznosti
         print(4, b, c, d, sep='')

# 4 je 2. mieste
for a in cifry - {2}: # 2-krat sa zopakuje
   for c in cifry - {a}: # 2-krat sa zopakuje
      for d in cifry - {a, c}: # 1-krat sa zopakuje
         # Vypise 2 * 2 * 1 = 4 moznosti
         print(a, 4 , c, d, sep='')

# 4 je 3. mieste
for a in cifry - {2}: # 2-krat sa zopakuje
   for b in cifry - {a, 2}: # 1-krat sa zopakuje
      for d in cifry - {a, b}: # 1-krat sa zopakuje
         # Vypise 2 * 1 * 1 = 2 moznosti
         print(a, b, 4, d, sep='')
# Celkovo program vypise 6 + 4 + 2 = 12 monznosti
```

Časť b) cez delenie dvomi

Vynechajme podmienku, že 4 musí byť pred 7. Na prvé miesto máme 4 možnosti, na druhé len 3 (nemôžeme použiť prvú cifru), potom 2 (bez prvých dvoch cifier) a potom ostane 1 možnosť. Teda celkom máme $4 \cdot 3 \cdot 2 \cdot 1 = 24$ možností (ide vlastne o permutácie 4 prvkov). Tieto možnosti vieme rozdeliť do dvojíc, ktoré sa budú líšiť len poradím čísel 4 a 7 (keď v nejakom čísle vymeníme 4 a 7 dostaneme jeho pár a po opätovnej výmene máme zas pôvodné číslo, takže ide naozaj o dvojice). V každej dvojici máme jednu dobrú možnosť, kde 4 je pred 7, a jednu zlú možnosť, kde 4 je za 7. Preto dobrých možností je polovica: $24/2 = 12$.

Výpis možností Vypíšeme najprv všetky možnosti a potom vyznačíme dvojice dobrá (zelená) – zlá (červená) možnosť.



Riešenie c) cez zlé možnosti

Najprv spočítame počet 6-ciferných čísel zložených z cifier 1 a 7. Na každé zo 6 miest máme 2 možnosti, teda máme $2^6 = 64$ možností. Teraz spočítame zlé možnosti, teda také čísla, ktoré obsahujú tri jednotky za sebou. Tieto možnosti rozdelíme podľa toho, na ktorej možnosti sa začína **prvý** úsek troch jednotiek:

1. Ak sa začína na prvej pozícii (číslo vyzerá 111***), tak na každej zo zvyšných troch miest môže byť 1 alebo 7, teda máme $2^3 = 8$ možností.
2. Ak na druhej pozícii, tak na prvej pozícii musí byť 7 (teda číslo má tvar 7111**). Ostali nám posledné dve miesta, na každé dve možnosti, teda máme $2^2 = 4$ možnosti.
3. Teraz je číslo tvaru *7111*, na prvom mieste môže byť 1 aj 7, rovnako aj na poslednom. Máme teda opäť $2^2 = 4$ možnosti.
4. Teraz je číslo tvaru **71111, na prvom a druhom mieste máme 2 možnosti, teda opäť máme $2^2 = 4$ možnosti.

Vďaka tomu, že uvažujeme prvý úsek 111, tak tieto možnosti sú navzájom disjunktné. Preto ich je celkovo $8 + 4 + 4 + 4 = 20$. Preto tých možností, ktoré neobsahujú 111, je $64 - 20 = 44$.

Vypísanie možností Zlé možnosti sme vypísali nasledovným systémom:

111111			
111117			
111171	711111	171111	117111
111177	711117	171117	177111
111711	711171	771111	717111
111717	711177	771117	777111
111771			
111777			
111777			

Dobré možnosti tak vieme potom vypísať tak, že vypíšeme všetky možnosti a vyškrtáme tie, ktoré sme vypísali vyššie.

~~111111~~	~~117111~~	~~171111~~	~~177111~~	~~711111~~	~~717111~~	~~771111~~	~~777111~~
~~111117~~	117117	~~171117~~	177117	~~711117~~	717117	~~771117~~	777117
~~111171~~	117171	171171	177171	~~711171~~	717171	771171	777171
~~111177~~	117177	171177	177177	~~711177~~	717177	771177	777177
~~111711~~	117711	171711	177711	711711	717711	771711	777711
~~111717~~	117717	171717	177717	711717	717717	771717	777717
~~111771~~	117771	171771	177771	711771	717771	771771	777771
~~111777~~	117777	171777	177777	711777	717777	771777	777777

```
vsetky = [] # 2^6 = 64 moznosti
cifry = {'1', '7'}
for a in cifry:
```

```
    for b in cifry:
        for c in cifry:
            for d in cifry:
                for e in cifry:
                    for f in cifry:
                        vsetky.append(a + b + c + d + e + f)

zle = []
# 111... (2 * 2 * 2 = 8 moznosti)
for d in cifry:
    for e in cifry:
        for f in cifry:
            zle.append('111' + d + e + f)
# 7111.. (2 * 2 = 4 moznosti)
for e in cifry:
    for f in cifry:
        zle.append('7111' + e + f)
# .7111. (2 * 2 = 4 moznosti)
for a in cifry:
    for f in cifry:
        zle.append(a + '7111' + f)
# ..7111 (2 * 2 = 4 moznosti)
for a in cifry:
    for b in cifry:
        zle.append(a + b + '7111')
# Spolu 8 + 4 + 4 + 4 = 20 zlych moznosti

# Vypiseme moznosti: bude ich 64 - 20 = 44
zle_mnozina = set(zle)
for moznost in vsetky:
    if moznost not in zle_mnozina:
        print(moznost)
```

Časť c) cez menšie prípady

Úlohu vyriešime rovno vypísaním možností. Možnosti budeme vypisovať postupne od 1-ciferných čísel. Po 3-ciferné čísla bez problémov zvládneme všetky možnosti vypísať ručne (postupne na každom miestne skúšame najprv 1, potom 7, akurát pri trojmiestnych nechceme mať číslo 111). Štvormiestne čísla vypíšeme nasledovne:

1. Najprv vypíšeme čísla, ktoré sa začínajú na 7 (vyznačená modrou). Na zvyšných troch pozíciách môže byť ľubovoľné 3-ciferné číslo, ktoré neobsahuje tri jednotky. Takže k 7-ke len prepíšeme tieto možnosti.

2. Ostávajú čísla, ktoré sa začínajú na 1. Z nich vypíšeme tie, ktoré majú na druhom mieste cifru 7 (vyznečené zelenou). Ďalej môže byť ľubovoľné 2-ciferné číslo, čiže opäť vieme využiť výsledok z predošlého prípadu.

3. Ostali nám čísla, ktoré sa začínajú dvomi jednotkami (vyznačené červenou). Na treťom mieste musia mať 7-ku, aby sme nedostali tri jednotky. Na zvyšných miestach môžu mať ľubovoľné 1-ciferné číslo.

Tento spôsob vypysovania zopakujeme ešte dvakrát:

1. 5-ciferné čísla vypíšeme ako: 117 + 2-ciferné čísla, 17 + 3-ciferné a 7 + 4-ciferné čísla.

2. 6-ciferné vypíšeme ako: 117 + 3-ciferné čísla, 17 + 4-ciferné a 7 + 5-ciferné čísla.

Celý postup vypisovania je na obrázku 1. Tento postup podáme všeobecnejšie v ďalšom riešení.

1-miestne:	2-miestne:	3-miestne:	4-miestne:	5-miestne:	6-miestne:
1	11	117	1171	11711	117117
7	17	171	1177	11717	117171
	71	177	1711	11771	117177
	77	711	1717	11777	117711
		717	1771	17117	117717
		771	1777	17171	117771
		777	7117	17177	117777
			7171	17711	171171
			7177	17717	171177
			7711	17771	171711
			7717	17777	171717
			7771	71171	171771
			7777	71177	171777
				71711	177117
				71717	177171
				71771	177177
				71777	177711
				77117	177717
				77171	177771
				77177	177777
				77711	711711
				77717	711717
				77771	711771
				77777	711777
					717117
					717171
					717177
					717711
					717717
					717771
					717777
					771171
					771177
					771711
					771717
					771771
					771777
					777117
					777171
					777177
					777711
					777717
					777771
					777777

Obr. 1:

Program. Tento spôsob možno tiež priamočiaro previesť na program. Moznosti pre jednotlivé počty cifier si budeme ukladať do poľa `moznosti`.

```
moznosti = [
    [],
    ['1', '7'],
    ['11','17','71','77'],
    ['117','171','177','711','717','771','777']
]
for n in range(4, 7):
    # zoznam pre nove moznosti
    nove = []

    # pridame zacinajuce na 7
    for stara in moznosti[n - 1]:
        nove.append('7' + stara)

    # pridame zacinajuce na 17
    for stara in moznosti[n - 2]:
        nove.append('17' + stara)

    # pridame zacinajuce na 117
    for stara in moznosti[n - 3]:
        nove.append('117' + stara)

    moznosti.append(nove)

for moznost in moznosti[6]:
    print(moznost)
```

Časť c) všeobecne

Nech a_n označuje počet n-ciferných čísel zložených z cifier 1 a 7, ktoré neobsahujú tri jednotky za sebou (ďalej o nich hovoríme iba ako o n-ciferných číslach). Ľahko zistíme, že $a_1 = 2$, $a_2 = 4$, $a_3 = 7$. Pre $n \geq 4$ vypočítame a_n tak, že si všetky n-ciferné čísla rozdelíme na:

- tie, čo sa začínajú na 7 – tie môžu pokračovať ľubovoľným $(n-1)$-ciferným číslom, ktorých je a_{n-1};
- tie, čo sa začínajú na 17 – tie môžu pokračovať ľubovoľným $(n-2)$-ciferným číslom, ktorých je a_{n-2};
- tie, čo sa začínajú na 11 (a teda aj na 117) – tie môžu pokračovať ľubovoľným $(n-3)$-ciferným číslom, ktorých je a_{n-3}.

Tým sme ukázali, že pre všetky $n \geq 4$ platí

$$a_n = a_{n-1} + a_{n-2} + a_{n-3}. \tag{1}$$

Teda

- $a_4 = a_3 + a_2 + a_1 = 7 + 4 + 2 = 13$,
- $a_5 = a_4 + a_3 + a_2 = 13 + 7 + 4 = 24$,
- $a_6 = a_5 + a_4 + a_3 = 24 + 13 + 7 = 44$.

Teda hľadaný počet možností je 44.

Bonus 2. *(1 bod)* Napíšte program, ktorý načíta číslo n a vypíše počet všetkých n-ciferných čísel zložených z cifier 1 a 7, ktoré neobsahujú tri jednotky po sebe. Počet udelených bodov závisí od efektivity vášho programu (teda pre ako veľké n program vypíše výsledok do pár sekúnd). Nakoľko počty možností sú pre veľké n dosť veľké, tak stačí, keď vypíšete zvyšok výsledku po delení číslom $10^9 + 7$. Správnosť vášho programu odôvodnite (náročnosť odôvodnenia záleží od konkrétneho programu).

Programy môžeme založiť na riešeniach z úlohy 3c). Najjednoduchšie je presjť všetky možnosti a o každej z nich overiť, či obsahuje 111. Tým zvládneme dať správny odpoveď efektívne zhruba pre $n \leq 20$. Ten by sa dal o niečo zefektívniť prehľadávaním s návratom.

Efektívne riešenie však dostaneme tak, že využijeme vzťah (1), pomocou ktorého budeme postupne počítať hodnoty a_n.

```
n = int(input())
a = [None, 2, 4, 7]
for i in range(n - 3):
    a.append(a[-1] + a[-2] + a[-3]) # Nove a je sucet troch poslednych
print(a[n])
```

Zaujámavosťou je, že na toto riešenie nepotrebujeme zoznam. Stačí nám pamätať si v troch premenných posledné tri hodnoty.

```
n = int(input())
a, b, c = 2, 4, 7
    for i in range(n - 3):
        a, b, c = b, c, a + b + c
    print(c)
```

Tento program je efektívny pre asi $n \leq 300\,000$. Jediným problémom je, že pracuje s príliš veľkými číslami. Keďže nám však záleží len na zvyšku po delení číslom $p = 10^9 + 7$, tak nám stačí pamätať si iba tieto zvyšky. Pri tom využívame pravidlo $(a + b) \bmod d = (a \bmod d + b \bmod d) \bmod d$, teda ak chceme zistiť zvyšok súčtu, tak nám stačí spočítať zvyšky sčítancov (a prípadne z neho ešte spraviť zvyšok).

```
n = int(input())
a, b, c = 2, 4, 7
    for i in range(n - 3):
        a, b, c = b, c, (a + b + c) % (10**9 + 7)
    print(c)
```

Tento spôsob riešenia kombinatorických úloh sa volá *rekuretný* – spočíva v tom, že výsledok na úlohu pre n (prípadne aj viac premenných) vyjadríme pomocou výsledkov pre menšie hodnoty ako n. Na rovnakom princípe je založené aj počítanie kombinačných čísel z pravidiel Pascalovho trojuholníka.

Zaujímavosťou ešte je, že existuje efektívnejšie riešenie. Náš algoritmus spočíva v tom, že trojicu (a, b, c) nahradíme trojicou $(b, c, a + b + c)$. Na toto sa môžeme pozrieť ako na istý typ funkcie (ktorá každej trojici priradí práve jednu inú trojicu). Dokonca ide o lineárnu funkciu. Budúci semester sa naučíte, že takéto lineárne funkcie možno vyjadriť pomocou násobenia matíc. Keď chceme túto funkciu viackrát zopakovať, tak z toho dostaneme umocňovanie matice, ktoré sa dá spraviť výrazne efektívnejšie.