

# Výběr projektů

## Zrna a šachovnice

### *Zadání a popis:*

Představte si, že umístíte jedno zrnko rýže na první pole šachovnice, dvě na druhé, čtyři na třetí a tak dále, přičemž množství na každém poli se zdvojnásobí oproti předchozímu. Kolik zrn rýže bude na celé šachovnici?

Problém počtu zrn na šachovnici se zabývá otázkou, kolik zrn rýže by se nacházelo na šachovnici, pokud by se na první pole položilo jedno zrnko rýže, na druhé pole dvě zrnka, na třetí čtyři zrnka, a tak dále, přičemž počet zrn na každém poli se zdvojnásobuje oproti předchozímu poli. Tento problém vychází z matematického principu exponenciálního růstu a nabízí možnost výpočtu celkového počtu zrn rýže na šachovnici.

### *Požadavky:*

- Napište program v C#, který vypočítá celkový počet zrn rýže na šachovnici.
- Použijte smyčku k iteraci přes všechna pole šachovnice.
- Uchovávejte aktuální počet zrn rýže ve proměnné.
- V každé iteraci zdvojnásobte aktuální počet zrn rýže.
- Po dokončení iterací vypište na konzolu celkový počet zrn rýže.

### *Nápověda k řešení:*

- Šachovnice má standardně 8 řádků a 8 sloupců.
- Můžete použít cyklus for se dvěma vnořenými smyčkami k iteraci přes jednotlivá pole šachovnice.

### *Očekávaný výstup:*

Pro standartní šachovnici 8x8 by program měl vypsát dvacetiferné číslo.

## Problém osmi královen

### *Zadání a popis:*

Problém 8 královen spočívá v umístění ( $n$ ) královen na šachovnici ( $n \times n$ ) tak, aby žádné dvě královny nebyly na stejném řádku, sloupci ani diagonále.

Úloha Problém osmi královen je známá svou kombinatorickou a algoritmickou náročností, neboť vyžaduje nalezení všech možných řešení umístění 8 (nebo  $n$ ) královen na šachovnici tak, aby se navzájem neohrožovaly. Cílem této úlohy je tedy vytvořit program, který systematicky prohledá všechny kombinace umístění královen a vypíše je, přičemž musí respektovat pravidla šachové hry a zajistit, aby žádné dvě královny nebyly na stejném řádku, sloupci ani diagonále.

### *Požadavky:*

- Napište program, který řeší problém osmi královen pro standardní šachovnici ( $n=8$ ) a pro uživatelem zadanou velikost šachovnice ( $n>0$ ).
- Využijte rekurzi k hledání všech možných řešení.
- Pro každé nalezené řešení vypište umístění jednotlivých královen na šachovnici.
- Změřte a vypište celkový počet nalezených řešení.

### *Nápověda k řešení:*

- Můžete reprezentovat umístění královen pomocí jednorozměrného pole, kde index reprezentuje sloupec a hodnota řádek.
- V každé rekurzivní volání zkontrolujte, zda umístění dané královny neohrožuje ostatní.
- Pokud je umístění bezpečné, pokračujte s umístěním dalších královen rekurzivně.
- V případě neplatného umístění se vraťte zpět z rekurze a zkuste jiné umístění pro předchozí královnu.

### *Očekávaný výstup:*

Pro standardní šachovnici by program měl vypsat celkový počet řešení (92) a uvést několik příkladů umístění královen.

## **Jezdcova procházka**

### *Zadání a popis:*

Problém jezdcovy procházky, známý také jako Rytířova cesta, představuje výzvu jak pro kombinatorickou, tak algoritmicou analýzu. Úloha spočívá v nalezení cesty, kterou by šachový jezdec mohl projet všemi poli na šachovnici, přičemž každé pole navštíví právě jednou.

Jezdcova procházka spočívá v nalezení posloupnosti tahů pro koně na šachovnici, tak aby navštívil každé pole přesně jednou. Řešte tento problém pomocí backtrackingu.

### *Požadavky:*

- Napište program, který řeší Jezdcovu procházku pro standardní šachovnici.
- Implementujte algoritmus pro hledání cesty koně.
- Reprezentujte šachovnici jako dvourozměrné pole, kde hodnota na daném poli udává, kolikrát bylo pole navštíveno (0 - nenavštíveno, 1 - navštíveno).
- V každé pozici vyzkoušejte všech osm možných tahů koně.
- Pokud tah je platný (nevede mimo šachovnici a pole nebylo navštíveno), zkuste pokračovat cestou od tohoto nového pole rekurzivně.
- Pokud se dostanete na poslední pole a všechny pole byla navštívena, našli jste řešení.
- Jinak se vraťte z rekurze a zkuste další možný tah z předchozí pozice.
- Po nalezení řešení, vypište posloupnost tahů (např. A1-C2-...) pro toto řešení.
- Pokud existuje více řešení, zkuste najít všechna z nich.

### *Nápověda k řešení:*

- Využijte datovou strukturu pro ukládání pozice koně a informací o navštívených polích.

### *Očekávaný výstup:*

Program by měl najít a zobrazit jedno z řešení pro Jezdcovu procházku na standardní šachovnici.

## Fisher-Yatesův algoritmus

### *Zadání a popis:*

Implementujte Fisher-Yates algoritmus a použijte ho k náhodnému promíchání pole prvků, které využijete k tvorbě jednoho z následujících: míchání slov v křížovkách, rozmisťování lodí v lodní bitvě, losování otázek v kvízu.

Úloha Fisher-Yatesův algoritmus spočívá v implementaci algoritmu pro náhodné promíchání pole prvků. Tento algoritmus je efektivní a používaný zejména tam, kde je potřeba generovat náhodné sekvence. Princip fungování spočívá v procházení pole od konce k začátku a postupném výběru náhodného prvku, který je následně prohozen s prvkem na aktuální pozici. Tímto způsobem se každý prvek pole postupně dostane na náhodnou pozici.

### *Požadavky:*

- Metoda by měla přijímat jako vstup pole prvků a náhodně je promíchat.
- Algoritmus využívá náhodná čísla k výběru prvku a jeho výměny se zástupcem na konci pole.
- Opakujte tento proces pro každý prvek od konce pole směrem k jeho začátku.
- Nepoužívejte vestavěnou funkci pro míchání pole jako Shuffle nebo OrderByRandom.

### *Nápověda k řešení:*

- Algoritmus prochází pole od konce k začátku.
- V každém kroku vybere náhodný index  $i$  v rozmezí od aktuální pozice až po konec pole.
- Vymění prvek na aktuální pozici s prvkem na pozici  $i$ .
- Můžete využít funkci Random.Next pro generování náhodných čísel.

### *Očekávaný výstup:*

Metoda by měla vracet pole promíchaných prvků odlišné od originálního pořadí.

## Magické čtverce

### *Zadání a popis:*

Úloha Magické čtverce se zabývá vytvářením čtvercových matic, ve kterých je součet čísel v každém řádku, sloupci a diagonále stejný. Zadání obsahuje algoritmy pro generování těchto magických čtverců, konkrétně metodu Středového bodu. Dále je úlohu možné řešit i Kiahanovým schématem nebo Lo Shu metodou. Cílem může být implementovat tyto algoritmy a porovnat jejich výstupy a rychlost generování.

### *Požadavky:*

Magické čtverce jsou čtvercové matice, ve kterých je součet čísel v každém řádku, sloupci a diagonále stejný. V tomto zadání se zaměřte na generování magických čtverců lichého řádu (např. 3x3, 5x5, 7x7 atd.).

### *Nápověda k řešení:*

Řešení pro metodu středového bodu:

- Umístěte číslo 1 do středu.
- Pro další čísla (od 2 do  $n^2$ ):
- Pohněte se o jednu pozici vpravo nahoru.
- Pokud jste mimo matici, jděte dolů a doprava.
- Pokud na nové pozici je již číslo, jděte dolů o dvě a pak vpravo.
- Vložte číslo na aktuální pozici.

### *Očekávaný výstup:*

Tabulka vyplněná čísly, případně ASCII art.

## Pascalův trojúhelník

### *Zadání a popis:*

Pascalův trojúhelník je matematická struktura, která obsahuje čísla uspořádaná do tvaru trojúhelníku tak, že každé číslo je součtem dvou čísel nad ním. Cílem této úlohy je napsat program, který generuje Pascalův trojúhelník pro zadaný počet řádků.

### *Požadavky:*

- Program by měl přijímat jako vstup celé číslo, které udává počet řádků trojúhelníku.
- Následně by měl vygenerovat a zobrazit Pascalův trojúhelník s daným počtem řádků.
- Použijte pro výpočet hodnot v trojúhelníku standardní rekurentní vzorec:

$$T(n,k) = T(n-1,k-1) + T(n-1,k).$$

- Ve výstupu zarovnejte čísla v každé řádce pro lepší přehlednost.

### *Nápověda k řešení:*

- Můžete využít dvourozměrné pole pro ukládání hodnot Pascalova trojúhelníku.
- Použijte cyklus pro iteraci přes řádky a sloupce trojúhelníku.

### *Očekávaný výstup:*

*Pro vstupní hodnotu 5 by program měl zobrazit následující Pascalův trojúhelník:*

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

## Převod římských čísel na arabská

### *Zadání a popis:*

Úkolem této úlohy je napsat funkci, která provádí konverzi římských číslic na celá arabská čísla. Mezi požadavky patří implementace základních symbolů římských číslic (I, V, X, L, C, D, M), kontrola platnosti vstupního řetězce včetně délky a správného použití odčítání. Výsledná funkce má být efektivní a čitelná.

### *Požadavky:*

- Napište funkci v C#, která převádí římské číslice na celé číslo.
- Funkce by měla přijímat jako vstup řetězec s římskými číslicemi (např. "III", "IV", "MCMXLIV").
- Výstupem funkce by mělo být odpovídající celé číslo (např. 3, 4, 1944).
- Implementujte podporu pro základní symboly I, V, X, L, C, D a M.
- Kontrolujte platnost vstupního řetězce (délka, platné symboly, správné použití odčítání).
- Výsledná funkce by měla být efektivní a čitelná.

### *Nápověda k řešení:*

- Můžete využít dictionary pro mapování symbolů na jejich hodnoty.
- Iterujte přes jednotlivé znaky vstupního řetězce.
- Kontrolujte platnost znaků a zvažte použití výjimky pro neplatné vstupy.
- Pro odčítání symbolů (IV, IX) zkontrolujte pořadí v řetězci a hodnoty.

### *Očekávaný výstup:*

Pro vstupní řetězec "III" by funkce měla vrátit číslo 3. Pro neplatný vstup by měla být zobrazena chybová zpráva s vysvětlením.

## Překládání papíru až k měsíci

### *Zadání a popis:*

Úloha se zaměřuje na výpočet počtu přeložení papíru na polovinu potřebných k dosažení tloušťky odpovídající vzdálenosti k Měsíci. S použitím základního předpokladu, že každým složením se tloušťka papíru zdvojnásobí, a znalosti tloušťky papíru a vzdálenosti k Měsíci, program vypočítá počet potřebných složení. Cílem je demonstrovat exponenciální růst tloušťky papíru a ilustrovat překvapivé výsledky, které mohou vzniknout z jednoduchého matematického principu. Tato úloha tak ilustruje, jak malé změny mohou vést k pozoruhodným výsledkům, které mohou být na první pohled nepředstavitelné.

### *Požadavky:*

- Napište program, který vypočítá, kolikrát byste museli složit papír na polovinu, abyste jeho tloušťkou dosáhli na Měsíc.
- Předpokládejte, že tloušťka papíru je 0.1 milimetru a vzdálenost k Měsíci je 384400 kilometrů.
- Každým složením se tloušťka papíru zdvojnásobí.

### *Nápověda k řešení:*

- Porovnejte aktuální tloušťku s požadovanou vzdáleností.
- Vyhledejte výšku Mount Everestu nebo jiného referenčního bodu.

### *Očekávaný výstup:*

Program by měl zobrazit výsledek jako celé číslo a uvést, zda počet složení je v rámci možností nebo zda je to nadlidský úkol.



## Ulamova spirála

### *Zadání a popis:*

Ulamova spirála je matematická vizualizace prvočísel pojmenovaná po matematikovi Stanisławu Ulamovi. Spirála začíná číslem 1 ve středu a každé další číslo je umístěno do spirálového vzoru.

Pointa této úlohy spočívá ve vytvoření programu, který generuje Ulamovu spirálu a identifikuje v ní prvočísla. Tím umožňuje vizualizovat distribuci prvočísel a lépe porozumět jejich matematickým vlastnostem. Ulamova spirála je fascinujícím vizuálním zobrazením prvočísel, které má aplikace v matematice, informatice a analýze dat. Implementace tohoto algoritmu umožňuje zkoumat vzorce a vlastnosti prvočíselného rozložení a experimentovat s různými formáty vizualizace.

### *Požadavky:*

- Napište program, který generuje Ulamovu spirálu zadané velikosti (počet prvků na stranu).
- Umožněte uživateli zadat velikost spirály jako celé číslo.
- Implementujte algoritmus pro umístění čísel do spirály a identifikaci prvočísel.
- Zobrazte spirálu pomocí znaků (např. ' ' pro prázdné pole, '#' pro prvočísla, '-' a '|' pro další čísla).
- Formátujte výstup spirály pro přehledné zobrazení.

### *Nápověda k řešení:*

- Můžete využít dvourozměrné pole pro ukládání čísel v spirále.
- Použijte funkci pro kontrolu prvočíselnosti (např. dělitelnost).
- Použijte různé znaky pro zobrazení prázdného pole, prvočísla a dalších čísel.
- Použijte cyklus a formátovací řetězce pro pěkné zobrazení spirály.

### *Očekávaný výstup:*

Pro vstupní velikost 5 by program měl zobrazit:

```
---- |
#--- |
|---#
| -# - |
| - -# |
```

## Hammingova vzdálenost

### *Zadání a popis:*

Tato úloha se zaměřuje na výpočet Hammingovy vzdálenosti mezi dvěma zadanými celými čísly. Hammingova vzdálenost je metrika používaná k měření rozdílu mezi dvěma binárními řetězci stejné délky. V našem případě jsou tato binární čísla reprezentována dvěma celými čísly. Úkolem je implementovat dvě různé metody pro výpočet této vzdálenosti. Tato metrika pro vzdálenost je užitečná například při porovnávání DNA sekvencí v bioinformatice, při klasifikaci dat v analýze dat, nebo při detekci a opravě chyb v komunikačních sítích.

### *Požadavky:*

- Napište program v C#, který vypočítá Hammingovu vzdálenost mezi dvěma zadanými celými čísly.
- Hammingova vzdálenost je počet bitů, ve kterých se dvě čísla liší.
- Program by měl přijímat jako vstup dvě celá čísla.
- Výstupem programu by měla být Hammingova vzdálenost mezi těmito čísly.
- Implementujte metodu s bitwise operací, rekurzí nebo použijte vestavěnou funkci.

### *Nápověda k řešení:*

- Můžete využít bitwise operátor XOR (^) pro porovnání jednotlivých bitů.
- Posouváním bitů vpravo můžete izolovat jednotlivé bity z celého čísla.
- Funkce BitConverter.ToString(long) konvertuje celé číslo na jeho binární reprezentaci.
- Cyklus se zbytkem po dělení 2 (% 2) umožňuje iterovat přes jednotlivé bity.

### *Očekávaný výstup:*

Řetězce "ACGT" a "ACGG" se liší v jediné pozici (2. znak), takže jejich Hammingova vzdálenost je 1.

Pro vstupní hodnoty 97 (01100001) a 133 (10000101) by program měl zobrazit:

Hammingova vzdálenost: 3 Číslo 1: 01100001 Číslo 2: 10000101