

Úloha č. 4

Teleportace

10 b

V úloze máme rozhodnout, zda daný hráč má jistou výhru za podmínky, že oba hráči používají optimální strategii. Optimální strategie znamená, že nedělají chyby a rozhodují se tak, jako by znali celý průběh hry pro každou z možností, kterou mohou zvolit.

Zaklínadla hráče posunují pouze na východ a na jih. Pokud se tedy posunou z políčka a na políčko b nějakým zaklínadlem, pak se již v této hře nikdy nemohou na políčko a dostat zpět. Díky tomuto a faktu, že hra je optimální, můžeme pro každé políčko rozhodnout zda hráč, který na něm stojí má šanci vyhrát, nebo zda použití libovolného zaklínadla pro něj znamená prohru. Označíme tedy políčka, kdy má hráč šanci vyhrát (jde o optimální hru a tedy pokud má hráč šanci vyhrát tak ji využije), jako výherní a políčka, pro která žádnou šanci na výhru nemá, jako proherní.

Začínáme v rohu a Martien neví, jak udělat první tah. K tomu, aby se rozhodl potřebuje vědět výsledek hry pro všechna zaklínadla, která v tu chvíli může použít. Zkusí se tedy zeptat postupně pro každé zaklínadlo, jak hra dopadne. Při použití každého ze zaklínadel se hráči přesunou na nějaké další políčko. Všechna políčka mimo hrací pole označíme jako výherní.

Pro začátek zkusíme uvažovat, že pouze zkoušíme všechny permutace zaklínadel. Rekursivně se ptáme na každém políčku jak hra dopadne pro toto políčko.

```
def reseni(x, y):
    for each zaklinadlo:
        if reseni(x+zaklinadlo.x, y+zaklinadlo.y) == VYHRA:
            return VYHRA
    return PROHRA
```

Časová složitost bude exponenciální ($O(z^{max(n_{jih}, n_{vychod})})$), ale paměťová náročnost konstantní (není závislá na číslech z , n_{jih} ani n_{vychod}). Tedy algoritmus poběží pro větší čísla hodně dlouho, ale zabere pouze minimum paměti. Časová složitost a paměťová náročnost spolu v mnoha případech, včetně tohoto, úzce souvisí. Můžeme tak vylepšit časovou složitost tím, že si budeme ukládat více informací. U této úlohy si například můžeme všimnout, že budeme často opakovaně počítat to samé. Například pokud bychom měli zaklínadlo (0, 1) a zaklínadlo (1, 0), tak při použití zaklínadel v pořadí (0, 1), (1, 0) i (1, 0), (0, 1) se dostaneme na políčko [1, 1] a ptáme se na stejnou otázku – zda je výherní nebo proherní. Protože informace o tom, zda je políčko výherní nebo proherní, je spjatá pouze se souřadnicemi na mřížce nádvoří, stačí nám použít tabulku s rozměry stejnými jako nádvoří ($n_{jih} \times n_{vychod}$). Do této tabulky si uložíme výsledek našich výpočtů a při následujícím dotazu na již spočítané políčko pouze vrátíme výsledek rovnou z tabulky a nemusíme tak už počítat dál.

```
tabulka[n_jih][n_vychod]

def reseni(x, y):
    if tabulka[x][y] == NEVYPLNENE:
        vysledek = PROHRA
        for each zaklinadlo:
            if x+zaklinadlo.x >= n_jih or y+zaklinadlo.y >= n_vychod or \
                reseni(x+zaklinadlo.x, y+zaklinadlo.y) == VYHRA:
                vysledek = VYHRA
                break
```

```
    tabulka[x][y] = vysledek
    return tabulka[x][y]
```

Tato jednoduchá obměna přináší výrazné zlepšení časové složitosti – díky tomu, že tabulka je omezeně velká – $O(n_{jih} \cdot n_{vyhod})$. Paměťová náročnost bude stejná (také se odvíjí od velikosti tabulky) $O(n_{jih} \cdot n_{vyhod})$.

Alternativně můžeme stejný postup udělat i iterativně – začneme v opačném rohu, než začínají hráči. Poté postupně vyplňujeme tabulku. Díky tomu, že se pohybujeme v opačném směru než hráči, jsou všechna políčka, na která se dotazujeme, již vyplněná.

```
tabulka[n_jih][n_vychod]

for x in range(n_jih):
    for y in range(n_vychod):
        tabulka[x][y] = PROHRA
        for each zaklinadlo:
            if x+zaklinadlo.x >= n_jih or y+zaklinadlo.y >= n_vychod or \
                tabulka[x+zaklinadlo.x][y+zaklinadlo.y] == VYHRA:
                tabulka[x][y] = VYHRA
                break
```