KSP 33-Z1-3: Petrův zmatený výlet

Metodický materiál pro budoucí učitele informatiky

Seminář z algoritmizace

17. února 2026

Co úloha žáky naučí (Didaktické cíle)

- **Znalosti:**
  - Reprezentace 2D mřížky (pole polí).
  - Pohyb v mřížce pomocí souřadnic.
  - Práce s orientovaným grafem.
- **Dovednosti:**
  - Algoritmus prohledávání do šířky (BFS).
  - Změna paradigmatu: Schopnost optimalizovat řešení inverzním (zpětným) přístupem.
- **Návyky:**
  - Správné ošetření okrajových podmínek (ochrana proti chybě *Index Out of Bounds*).

Rozbor úlohy: Naivní přístup

Princip: Simulace cesty od začátku. Zkusíme se z každého políčka vydat po šipkách a uvidíme, jestli dojdeme do cíle `C`.

Problémy k diskusi s žáky:

1. **Zacyklení:** Co když dvě šipky ukazují na sebe navzájem? Program spadne do nekonečné smyčky, pokud si neukládáme historii.
2. **Zbytečná práce:** Řada cest se brzy spojí, ale zkoumáme je z každého políčka znovu.

Časová složitost

V nejhorším případě prohledáváme opakovaně tytéž cesty. Časová složitost takového přístupu by byla $\mathcal{O}((R \times S)^2)$.

Rozbor úlohy: Optimální řešení (Aha-moment)

Heuristická otázka: *„Kam všechny cesty vedou? Nešlo by postupovat od konce?“*

Princip (BFS na reverzním grafu):

- Místo zkoumání směru *od nás*, stoupneme si do cíle `C`.
- Ptáme se: *„Ze kterých sousedních políček vede šipka k nám?“*
- Zpracovaná políčka označíme # a vložíme do fronty.
- Tento postup opakujeme (Prohledávání do šířky - BFS).

Časová složitost

Každé políčko navštívíme maximálně jednou. Složitost je lineární vzhledem k velikosti mapy, tedy $\mathcal{O}(R \times S)$.

Pseudokód (Postup od cíle)

```
vysledek = vytvor_mrizku(R, S, '.')
fronta = prazdna_fronta()

# 1. Najdi cil C a pridej do fronty
pro r, c v mape:
    pokud mapa[r][c] == 'C':
        fronta.pridej((r, c))
        vysledek[r][c] = 'C'

# 2. Samotne prohledavani z fronty
dokud fronta neni prazdna:
    (r, c) = fronta.odeber_prvni()

    # Soused dole (Pokud ma 'S', ukazuje na nas nahoru)
    pokud r+1 < R a mapa[r+1][c] == 'S' a vysledek[r+1][c] == '.':
        vysledek[r+1][c] = '#'
        fronta.pridej((r+1, c))

    # Podobne zkontrolujeme sousedy: nahore ('J'), vlevo ('V'), vpravo ('Z')
```

Prohledávání proti proudu šipek

Vzorový kód v Pythonu (Část 1: Načítání a cíl)

```python
from collections import deque

def vyres_vylet():
    R, S = map(int, input().strip().split())

    mapa = []
    for _ in range(R):
        mapa.append(input().strip())

    vysledek = [['.' for _ in range(S)] for _ in range(R)]
    fronta = deque()

    # Nalezeni cile a jeho zarazeni do fronty
    for r in range(R):
        for c in range(S):
            if mapa[r][c] == 'C':
                fronta.append((r, c))
                vysledek[r][c] = 'C'

    # Zde navazuje smycka (viz dalsi snimek)
```

Vzorový kód v Pythonu (Část 2: Smyčka BFS)

```
    # ...racovani funkce vyres_vylet()
    while fronta:
        r, c = fronta.popleft()

        # Soused dole (smeruje na sever 'S')
        if r + 1 < R and mapa[r+1][c] == 'S' and vysledek[r+1][c] == '.':
            vysledek[r+1][c] = '#'
            fronta.append((r + 1, c))

        # Soused nahore (smeruje na jih 'J')
        if r - 1 >= 0 and mapa[r-1][c] == 'J' and vysledek[r-1][c] == '.':
            vysledek[r-1][c] = '#'
            fronta.append((r - 1, c))

        # Obdobne pro vlevo ('V') a vpravo ('Z')...

    # Vypis vysledku
    for r in range(R):
        print("".join(vysledek[r]))
```

Metodické tipy do výuky

- **Zrádnost souřadnic `(r, c)` vs. `(x, y)`:**
  Žáci mají z matematiky zažito, že první souřadnice je horizontální (X) a druhá vertikální (Y). V polích polí je to ale naopak (řádek, sloupec). Doporučte pojmenovávat proměnné jasně `r, c` (row, column).

- `collections.deque` **místo** `list`**:**
  Pokud by žáci použili klasické pole a odebírali ze začátku přes `pop(0)`, složitost by klesla, protože posun všech prvků trvá $\mathcal{O}(N)$. S `deque` je odebrání prvního prvku konstantní $\mathcal{O}(1)$.

- **Podmínka proti zacyklení** `vysledek[...] == '.'`**:**
  Zamezuje zacyklení v samotném BFS a brání zbytečnému opakovanému zpracování stejných políček.