Univerzita Karlova

Přírodovědecká fakulta

Úvod do programování

Eva Flášarová

2. ročník studijního programu Geografie a kartografie

Praha 2026

1. Bludiště

1.1 Zadání úlohy

Na rastru tvořeném 50 x 50 pixelů vygenerujte bludiště. Vyberte startovní buňku a označte ji jako navštívenou. Dokud existují nenavštívené sousední buňky, vyberte náhodného souseda, který ještě nebyl navštíven, odstraňte zeď mezi aktuální buňkou a sousedem, přesuňte se na souseda a opakujte (rekurze). Cestu v takto vygenerovaném bludišti vyhledejte metodami BFS (nejkratší) nebo DFS (libovolnou).

1.2 Rozbor problému

Nejprve se vygeneruje bludiště, které se skládá z jednotlivých buněk ohraničených zdmi. Následně se vybere libovolná buňka, kterou použijeme jako startovní místo, a označíme ji jako navštívenou. Náhodně si zvolíme některou ze sousedních buněk, která ještě nebyla navštívená, odstraníme zeď mezi těmito dvěma buňkami a přesuneme se do nové buňky. Takto budeme pokračovat, dokud se nedostaneme do takové buňky, ze které není možné se přesunout na dosud nenavštíveného souseda. Zde cesta skončí. Pro nalezení cesty v bludišti lze využít buď metodu BFS, nebo DFS.

1.3 Popis algoritmu

Základem úlohy je vygenerovat si vlastní bludiště na rastru o velikosti 50 x 50 pixelů. Bludiště vygenerujeme pomocí dvou tříd – *class Policko* a *class Bludiste*. Začínáme třídou *Policko*, kde si nadefinujeme existenci čtyř zdí pro každou buňku (nahoru, dolu, doleva a doprava).

Třídě *Bludiste* nejprve určíme výškové a šířkové rozměry, tedy 50 x 50 pixelů. Následně určíme „mapu“, která se bude skládat z jednotlivých buněk o souřadnicích x, y.

Bludiště jako takové se bude generovat pomocí funkce *gen_bludiste*. V rámci této funkce také naimplementuje způsob, jakým se budou odstraňovat zdi mezi buňkami. Pro každý ze čtyř možných směrů vytvoříme vlastní podmínku, která zeď v tomto směru odstraní a posune nás na nové políčko o příslušných souřadnicích. Zeď, kterou budeme odstraňovat, se vybírá náhodně (v případě, že je víc než jedna možnost).

Vytvoříme si novou funkci, která již bude hledat samotnou cestu. Neboť jsem se rozhodla hledat cestu metodou BFS, i tuto funkci pojmenujeme *bfs*. Zvolíme si políčko, ze kterého budeme startovat, a označíme jej jako navštívené. Následně vytvoříme frontu, do které vložíme naše startovní políčko. Poté už budeme za pomoci while cyklu postupně procházet bludiště, odstraňovat zdi a každé další navštívené políčko včetně jeho souřadnic uložíme do naší fronty. Takto pokračujeme, dokud se lze přesouvat na doposud nenavštívená sousední políčka.

Na závěr vytiskneme souřadnice cílového políčka (*bludiste.end*) a také souřadnice celé cesty.

1.4 Rozbor známých algoritmů

1.4.1 Metoda BFS

Prohledávání do šířky neboli breadth-first search (BFS) je nejjednoduším způsobem prohledávání grafu. Postupně hledáme všechny vrcholy, do kterých lze z výchozího uzlu dojít po hranách. Na počátku prohlásíme vstupní uzel za otevřený a v momentě, kdy otevřeme jeho následovníka, uzavřeme původní vstup. Otevřené vrcholy máme zapsané ve frontě, odkud je po uzavření zase odebíráme.

Pseudokód metody BFS:

Vstup: graf G = (V, E), počáteční vrchol $v_0 \in V$
Pro všechny vrcholy v:
    stav(v) <– nenalezený
stav(v_0*) <– otevřený*
Založíme frontu Q, do které vložíme vrchol v_0*.*
Dokud je fronta Q neprázdná:
    Odebereme první vrchol z Q a označíme ho v.
    Pro všechny následníky w vrcholu v:
        Je-li stav(w) = nenalezený:
            stav(w) <– otevřený
            Přidáme w do fronty Q.

stav(v) <– uzavřený

1.4.2 Metoda DFS

Prohledávání do hloubky neboli depth-first search (DFS) je dalším z řady grafových algoritmů. Danou větev grafu či stromu prozkoumá do maximální hloubky a v momentě, kdy již není možné pokračovat na dalšího nenavštíveného souseda, se vrátí zpět do výchozí pozice (tzv. backtracking). Poté začne prozkoumávat další cestu. Obvykle se implementuje buď za využití rekurze, nebo pomocí zásobníku.

Pseudokód metody DFS:

```
Vstup: graf G = (V, E)
    vytvoř prázdný zásobník S
    vlož start na zásobník
    dokud S není prázdný:
        v = odeber vrchol ze zásobníku
        pokud v není navštívený:
            označ v jako navštívený
            pro každého souseda w vrcholu v:
                pokud w není navštívený:
                    vlož w na zásobník
```

V této úloze jsme metodu DFS mohli použít, ale přišlo mi praktičtější zvolit metodu BFS.

1.5 Problematická místa

Prvním výrazným problémem úlohy bylo správné pochopení zadání – cílem totiž nebylo nalézt cestu v náhodně postaveném bludišti (tedy v takovém, kde stěny někde jsou a někde ne), ale procházení pravidelné čtverečkované mřížky, kdy si stěny odstraňuje sám program. Na to rovnou navázal další problém – vymyslet samotný algoritmus, který by zvládl takové bludiště vygenerovat a pak jej správně procházet.

1.6 Možná vylepšení

Program by určitě šel podstatně vylepšit, zejména části s podmínkami pro jednotlivé zdi by si zasloužily přepsat do nějaké „uhlazenější podoby“.

Pro lepší přehlednost a pochopení úlohy by také šla dodělat vykreslovací část, na které by bylo znázorněno bludiště a možné varianty cest v něm. Také by se pochopitelně mohla dodělat již zmiňovaná metoda DFS.

2. Zlatý řez

2.1 Zadání úlohy

Pro zadanou funkci y = f (x), která je na intervalu <a, b> unimodální, naleznete se zadanou chybou eps (vstupní parametr) její minimum. Pro ilustraci použijte alespoň 5 různých funkcí.

2.2 Princip zlatého řezu

Hledání minima funkce metodou zlatého řezu pracuje s následující konstantou:

$$\varphi = \frac{\sqrt{5}-1}{2} \approx 0{,}618$$

V každém kroku jsou definovány dva vnitřní body:

x1 = b – ϕ (b−a)

x2 = a + ϕ (b−a)

Následně se porovnávají funkční hodnoty: jestliže f(x1) < f(x2), minimum leží v intervalu $\langle a, x_2 \rangle$. Jinak leží v intervalu $\langle x_1, b \rangle$. Hledání skončí v momentě, kdy |b – a| < ε.

2.3 Popis algoritmu

Neboť chceme úlohu řešit za využití objektově orientovaného programování, vytvoříme si nejprve třídu *ZlatyRez*. Jejími atributy budou jednotlivé funkce, krajní body intervalů těchto funkcí (označeny jako *a, b*), a požadovaná přesnost (označena jako *eps*). Dále si spočítáme konstantu zlatého řezu (*self.konst*).

Vytvoříme si funkci, která bude vyhledávat minima jednotlivých funkcí, a nazveme ji *hledej_minimum*. Následujícím výpočtem rozdělíme interval ve zlatém poměru:

*x1 = b - self.konst * (b - a)*

*x2 = a + self.konst * (b - a)*

A následně spočítáme hodnoty funkce v těchto bodech.

f1 = self.f(x1)

f2 = self.f(x2)

Následně provedeme while cyklus, kterým budeme postupně zmenšovat interval a porovnávat hodnoty funkce. Interval zmenšujeme vždy pouze z jedné strany, druhý výchozí bod ponecháváme. Pokračujeme v iteraci, dokud nenalezneme minimum funkce v zadaném intervalu.

Pokračujeme vytvořením pěti funkcí, na které budeme metodu Zlatého řezu aplikovat. Tyto funkce již definujeme mimo třídu *ZlatyRez*, každou zvlášť jako samostatnou funkci. Pro ilustraci jsem použila následující funkce:

1) funkce f1 – kvadratická funkce ve tvaru $x^2 + 2x + 1$

2) funkce f2 – kubická funkce ve tvaru $x^3 + 2x^2 – x + 1$

3) funkce f3 – funkce sinus ve tvaru sin(x)

4) funkce f4 – funkce cosinus ve tvaru cos(x)

5) funkce f5 – exponenciální funkce ve tvaru e^x

Dále si založíme dva seznamy. Do seznamu *funkce* uložíme všech pět funkcí, do seznamu *intervaly* pak jednotlivé hraniční body, pro které budeme u každé funkce zvlášť počítat minimum. Nakonec už jen zavoláme funkci pro hledání minima a vytiskneme výsledné hodnoty minim pro všech pět funkcí.

2.4 Vstupy a výstupy

Na vstupu potřebujeme mít funkci f(x), posuzovaný interval <a, b> a přesnost ε. Výstupní data nám pak tvoří nalezené minimum této funkce.

2.5 Problematická místa

Aby tato úloha správně fungovala, musí být splněno hned několik podmínek. Problém může nastat v následujících případech:

1) Pokud funkce není na zvoleném intervalu unimodální, tedy nemá pouze jedno minimum. Je nutné zvolit správný interval, a pokud by jej měl zadávat uživatel, je nutné ošetřit případ, kdy funkce nebude unimodální.

2) Pokud nebude zadán rozumný interval. Krajní body a, b musí být reálná čísla a současně musí platit, že $a < b$.

3) Pokud nebude vhodně zvolené ε. Jeho hodnota musí být kladná, jinak dojde k nekonečnému výpočtu či jeho okamžitému konci.

2.6 Možná vylepšení

Kód by šel upravit např. přidáním uživatelského rozhraní, kdy si sám uživatel zvolí jednotlivé parametry, případně i funkci, kterou chce vypočítat. Také by se úloha dala přepsat tak, aby v ní nebylo použito objektově orientované programování, které zde není úplně nutno použít.

Zdroje

MAREŠ, M., VALLA, T.: Průvodce labyrintem algoritmů, 2022, CZ NIC.