Algoritmus Breadth-First Search (BFS) a jeho implementácia v hernom prostredí (2025)

Bc. Volodymyr Seheda, Bc. Maroš Drábik

[[1]](#footnote-1)

***Abstrakt*—Algoritmus Breadth-First Search (BFS) predstavuje jeden zo základných pilierov grafových algoritmov a patrí medzi najčastejšie používané metódy pri prehľadávaní neohodnotených grafových štruktúr. Jeho univerzálnosť a jednoduchosť z neho robia ideálny nástroj na riešenie úloh, ako je nájdenie najkratšej cesty, detekcia komponentov súvislosti alebo prehľadávanie stavového priestoru v úlohách umelej inteligencie. V tomto článku sa zameriavame na podrobnú analýzu princípov fungovania BFS algoritmu a jeho praktickej implementácie v programovacom jazyku Python. Predstavujeme interaktívne herné prostredie, v ktorom agent naviguje dvojrozmernou mriežkou s prekážkami, pričom využíva BFS na efektívne plánovanie pohybu.**

**Cieľom práce je nielen demonštrovať efektivitu a deterministický charakter BFS v praxi, ale aj predstaviť jeho edukatívny potenciál formou dynamickej vizualizácie a používateľskej interakcie. Článok ďalej porovnáva BFS s inými algoritmami, ako sú DFS, Dijkstra a A\*, pričom sa zameriava na ich výhody, obmedzenia a oblasti použitia. Na záver navrhujeme viaceré možnosti rozšírenia aplikácie, vrátane implementácie heuristických algoritmov a rozšírenia prostredia o dynamické prvky. Predstavená práca je určená nielen pre študentov informatiky, ale aj pre vývojárov a pedagógov, ktorí hľadajú efektívny spôsob, ako demonštrovať základné princípy algoritmického prehľadávania grafov v zrozumiteľnom a interaktívnom formáte.**

***Kľúčové pojmy — BFS, algoritmy grafov, vyhľadávanie ciest, umelá inteligencia, vizualizácia, Python***

# I. Úvod

V

YHĽADÁVACIE algoritmy zohrávajú kľúčovú úlohu v mnohých oblastiach informatiky – od sieťových analýz, cez robotiku a navigačné systémy, až po umelú inteligenciu a herný priemysel. Ich hlavnou úlohou je efektívne prehľadať daný priestor a nájsť požadovaný cieľ – napríklad trasu z bodu A do bodu B, nájdenie najkratšej alebo optimálnej cesty, alebo systematické preskúmanie celého stavového priestoru. V rôznych kontextoch môže ísť o graf uzlov, fyzický priestor, virtuálne prostredie, či logickú štruktúru problémov.

Jedným z najznámejších a najpoužívanejších algoritmov na tento účel je Breadth-First Search (BFS) – algoritmus prehľadávania grafu do šírky. BFS zabezpečuje, že každá úroveň susedných vrcholov je navštívená skôr, než algoritmus prejde do hlbších vrstiev grafu. Táto vlastnosť mu umožňuje nájsť najkratšiu cestu v neohodnotených grafoch, čo z neho robí výnimočne užitočný nástroj pre riešenie množstva praktických úloh, ako sú navigácia, riešenie hlavolamov, plánovanie pohybu agentov alebo hľadanie riešení v modeloch umelej inteligencie.

Historicky sa BFS objavuje už v literatúre z 50. rokov 20. storočia (napr. práce autorov ako Lee, Moore či Bellman), pričom jeho význam narastal spolu s rozvojom výpočtovej techniky a grafových modelov v informatike. Dnes je súčasťou štandardných učebníc algoritmov a tvorí základ pre pochopenie zložitejších techník, ako je Dijkstra alebo A\*.

Tento článok sa zameriava na podrobný rozbor algoritmu BFS – najprv z teoretického hľadiska, vrátane formálneho popisu a analýzy zložitosti, následne prostredníctvom reálnej implementácie v jazyku Python a napokon ako súčasť interaktívneho herného prostredia. Okrem praktického hľadiska kladieme dôraz aj na didaktický rozmer – ako pomocou jednoduchého herného konceptu možno lepšie porozumieť fungovaniu prehľadávacieho algoritmu. Čitateľ tak získa nielen hlbšie pochopenie základných princípov, ale aj schopnosť aplikovať tieto poznatky v praxi alebo v rámci ďalšieho výskumu.

# II. Teoretické základy BFS

Algoritmus Breadth-First Search (BFS), teda prehľadávanie do šírky, je jednou z najzákladnejších a najviac využívaných metód pri práci s grafmi. Jeho hlavnou charakteristikou je, že najprv skúma všetky uzly vzdialené jednu hranu od počiatočného bodu, potom tie, ktoré sú vzdialené dve hrany, a tak ďalej. Vďaka tomu je zaručené, že prvýkrát, keď algoritmus nájde cieľový uzol, bude cesta k nemu najkratšia (čo sa týka počtu krokov).

Algoritmus funguje na princípe fronty (FIFO – First In First Out), čo znamená, že uzly sa spracovávajú v poradí, v akom boli objavené. Na začiatku vložíme počiatočný vrchol do fronty a označíme ho ako navštívený. Potom v cykle vyberáme z fronty ďalší vrchol, preskúmame jeho susedov a tí, ktorí ešte neboli navštívení a sú priechodní (napríklad nie sú prekážkami), sa označia ako navštívení a vložia sa do fronty.

Pseudokód algoritmu je nasledovný:

Зображення, що містить текст, знімок екрана, Шрифт, алгебра

Вміст, створений ШІ, може бути неправильним.

Figure 1 BFS Algoritmus príklad pseudokódu

V teoretickom modeli grafu môžeme uzly chápať ako body a hrany ako spojenia medzi nimi. V prípade mriežky (gridu), ako je to v našej hre, je každý bod spojený so svojimi štyrmi susedmi (hore, dole, vľavo, vpravo), pokiaľ nie je ohraničený alebo zablokovaný.

Výhody BFS algoritmu:

* Najkratšia cesta: V neohodnotených grafoch zaručuje nájdenie najkratšej cesty (v počte krokov).
* Deterministickosť: Pri rovnakom vstupe poskytne vždy rovnaký výsledok.
* Jednoduchá implementácia: Pracuje len s frontou a množinou navštívených vrcholov, nevyžaduje heuristiky ani váhy.

Nevýhody:

* Pamäťová náročnosť: BFS si musí pamätať všetky uzly danej úrovne, čo môže byť problém pri veľkých grafoch.
* Neškálovateľnosť: V hlbokých alebo hustých grafoch môže byť výkon algoritmu nedostatočný.

BFS má široké využitie nielen v hrách, ale aj v oblastiach ako:

* Vyhľadávanie v sociálnych sieťach (napr. najkratšia cesta medzi dvoma používateľmi),
* Webové prehľadávače (prehľadávanie webových stránok do určitej hĺbky),
* Robotika (plánovanie pohybu robota v známom priestore),
* Umelá inteligencia (napr. riešenie hlavolamov a hier typu „labyrint“).

# III. Implementácia v Pythone

V rámci projektu sme vytvorili interaktívnu aplikáciu v jazyku Python, ktorá simuluje prehľadávanie dvojrozmerného sveta s prekážkami. Svet je reprezentovaný ako mriežka (grid) pevnej veľkosti 15×15 buniek. V tomto prostredí vystupuje **agent**, ktorého úlohou je nájsť cestu od počiatočnej pozície v ľavom hornom rohu k cieľu, ktorý sa nachádza v pravom dolnom rohu. Okrem klasického manuálneho ovládania šípkami alebo klávesami WASD je možné aktivovať tzv. **autopilot**, ktorý využíva algoritmus Breadth-First Search (BFS) na výpočet najkratšej cesty.

## Architektúra programu a komponenty

Program je navrhnutý modulárne, pričom hlavná trieda GridWorld reprezentuje aktuálny stav sveta a poskytuje všetky funkcie potrebné na jeho manipuláciu a vizualizáciu. Obsahuje:

* \_\_init\_\_() – inicializuje parametre sveta, ako sú rozmery, hustota prekážok a východiskové pozície.
* reset\_world(obstacle\_density) – náhodne vygeneruje nové rozloženie prekážok pri zachovaní fixnej štartovej a cieľovej pozície.
* \_generate\_obstacles(density) – zodpovedá za náhodné umiestnenie prekážok v mriežke.
* \_bfs\_search() – kľúčová metóda, ktorá implementuje BFS algoritmus a vracia zoznam krokov (smerov), ktoré agent musí vykonať.
* move\_agent(direction) – posunie agenta o jeden krok, buď podľa manuálneho vstupu, alebo podľa naplánovanej trasy.
* render() – vykreslí celý svet do terminálu pomocou farebných ASCII znakov.

Trieda taktiež definuje interné premenné ako agent\_pos, goal\_pos, obstacles, path, visited, ktoré uchovávajú aktuálny stav simulácie.

.

## Kľúčová časť implementácie BFS

**Algorithm:** BFS Navigation in GridWorld

**Algorithm part 1:** Initialize parameters

Define `**GridWorld**` dimensions: `**width**`, `**height**`.

Set `**obstacle\_density**`.

Generate `**obstacles**`, `**agent\_pos**`, and `**goal\_pos**`.

Configure display settings using `**Color**` class.

**Algorithm part 2:** Reset the world

Set initial agent and goal positions.

Generate obstacles using `\_generate\_obstacles(density)`.

Initialize `path`, `visited`, and other parameters.

**Algorithm part 3:** Breadth-First Search (BFS)

**for** each position in `queue` **do**

| Dequeue current position and path.

| **If** current position equals `goal\_pos**` then**

| | Return path and visited nodes.

| **end**

| **For** each direction (up, down, left, right) **do**

| | Check boundaries and obstacles.

| | Mark as visited and enqueue new position.

| **end**

**end**

**Algorithm part 4:** Start auto-pilot

Compute path using `BFS`.

**If** no path is found **then**

| Display "Path not found".

| Disable autopilot.

**end**

**Algorithm part 5:** Agent movement

**if** auto-pilot is active **then**

| Move agent along the BFS path.

**else**

| Allow manual movement via keyboard input.

**end**

**If** the agent reaches the goal **then**

| Display success message.

| Reset the world.

**end**

**Algorithm part 6:** Render the world

**Display the grid:**

| Use symbols for agent (`▲`), goal (`★`), obstacles (`█`), path (`•`), and empty cells (`·`).

Highlight visited nodes if auto-pilot is active.

**Algorithm part 7:** Game loop

**repeat**

| Render the world.

| **If** auto-pilot is active **then**

| | Perform next move along the BFS path.

| | **If** path is exhausted or goal is reached **then**

| | | Reset the world.

| | **end**

| **end**

|  **Else**

| | Wait for keyboard input to control the agent or toggle modes.

**until** exit command is received

**end**

**Algorithm part 8:** End game

Handle interruptions gracefully.

Exit the program.

Algoritmus využíva dátovú štruktúru deque (obojstrannú frontu) zo štandardnej knižnice collections, ktorá umožňuje efektívne pridávanie a odoberanie prvkov z oboch koncov. Na začiatku vložíme štartovaciu pozíciu agenta do fronty spolu s prázdnym zoznamom krokov. V cykle sa následne postupne odoberajú pozície z fronty a skúmajú ich štyria susedia. Každý dosiahnuteľný a ešte nenavštívený sused sa vloží do fronty a označí ako navštívený. Po nájdení cieľa sa vráti zoznam krokov, ktoré vedú k nemu.

Funkcia is\_valid(nx, ny, visited) kontroluje, či nová pozícia

nie je mimo hraníc mriežky, nie je obsadená prekážkou, alebo či ešte nebola navštívená.

Tento prístup zaručuje, že agent nájde najkratšiu možnú cestu podľa počtu krokov a zároveň necyklí v priestore. Výsledná cesta je následne použitá metódou move\_agent() na postupné vykonanie krokov, pričom sa stav aktualizuje a zobrazuje v termináli.

## Použité knižnice a techniky

* random – generovanie náhodných pozícií prekážok,
* os.system('clear') – vyčistenie terminálu medzi jednotlivými krokmi pre plynulé zobrazenie,
* enum – farebné označenie objektov pomocou ANSI escape kódov,
* termios, tty, sys – pre nízkoúrovňové spracovanie vstupu z klávesnice v reálnom čase (na Linuxe).

## Rozšíriteľnosť

Program je navrhnutý tak, aby ho bolo možné ľahko rozšíriť o nové typy algoritmov (napr. DFS, A\*, Dijkstra), meniť rozmer mriežky, meniť vizualizačné prvky, alebo pridať nové herné mechanizmy, ako napríklad dynamické prekážky či viacerých agentov.

# IV. Vizualizácia a interaktivita

Jednou z výnimočných čŕt nášho projektu je živá a prehľadná vizualizácia algoritmu BFS v textovom režime. Pomocou farebných ASCII znakov možno v reálnom čase sledovať, ako agent skúma priestor, vyhodnocuje možnosti pohybu a rozhoduje sa, ktorým smerom sa vydať. Takáto forma vizualizácie umožňuje nielen jednoduchšie ladenie implementácie, ale najmä podporuje výučbu algoritmického myslenia.

## A. Legenda vizuálnych prvkov

* *Agent*: ▲ (modrý trojuholník) — aktuálna pozícia agenta
* *Cieľ*: ★ (zelená hviezda) — cieľová pozícia, ktorú sa snaží agent dosiahnuť
* *Prekážka*: █ (červený blok) — bunka, cez ktorú sa agent nemôže pohybovať
* *Navštívené bunky*: ○ (fialový kruh) — všetky pozície, ktoré BFS algoritmus preskúmal
* *Cesta:* • (žltá bodka) — finálna najkratšia cesta, ktorú agent nasleduje po výpočte

## B. Používateľské ovládanie a režimy

Program ponúka dva režimy interakcie:

**Manuálny režim** – používateľ ovláda agenta priamo pomocou kláves W, A, S, D (alebo šípok). V tomto režime môže preskúmať prostredie a porovnať svoju výkonnosť s výpočtom BFS.

**Autopilot (BFS režim)** – aktivuje sa stlačením klávesu Q. Program okamžite spustí algoritmus BFS, vypočíta optimálnu trasu a agent sa automaticky začne pohybovať po tejto ceste smerom k cieľu.

## C. Reštartovanie a opakovanie

Po úspešnom dosiahnutí cieľa je používateľ informovaný o počte krokov a odmenený symbolickou „odmenou“. Simulácia sa potom reštartuje — vygeneruje sa nová náhodná konfigurácia sveta s novými prekážkami a agent sa vráti na štart.

## D. Didaktický prínos

Tento typ vizualizácie má vysokú výučbovú hodnotu. Umožňuje študentom a začiatočníkom intuitívne pochopiť, ako algoritmus BFS pracuje, bez nutnosti sledovať výpisy zoznamov alebo front. Farby a znaky jasne odlišujú stavy buniek a poskytujú okamžitú spätnú väzbu o priebehu vyhľadávania. Používateľ tak môže v priamom prenose pozorovať, ako algoritmus prehľadáva priestor do šírky, vyhodnocuje susedov a konštruuje najkratšiu cestu.

V. Porovnanie s inými algoritmami

Na lepšie pochopenie silných a slabých stránok algoritmu Breadth-First Search (BFS) je dôležité ho porovnať s ďalšími známymi vyhľadávacími algoritmami, ktoré sa používajú na riešenie podobných problémov. Medzi najčastejšie porovnávané patria Depth-First Search (DFS), Dijkstrov algoritmus a heuristický algoritmus A\*. Každý z týchto algoritmov má odlišné vlastnosti a je vhodný pre iný typ prostredia či úlohy.

Algoritmus Depth-First Search (DFS) pracuje na princípe prehľadávania do hĺbky. V praxi to znamená, že namiesto toho, aby algoritmus najprv preskúmal všetkých susedov daného vrcholu, sa okamžite vydá po jednej vetve až na jej koniec. DFS je menej náročný na pamäť, keďže sleduje len aktuálnu vetvu a neuchováva všetky uzly na rovnakej úrovni, ako to robí BFS. Na druhej strane však DFS negarantuje nájdenie najkratšej cesty, čo je jeho zásadné obmedzenie pri problémoch, kde je optimalita výsledku dôležitá. DFS je vhodnejší pre úlohy, kde je cieľ hĺbkovo skrytý, alebo kde nepotrebujeme najkratšie riešenie – napríklad pri zisťovaní cyklov v grafoch alebo pri generovaní labyrintov.

Na rozdiel od BFS a DFS pracuje Dijkstrov algoritmus s ohodnotenými grafmi, kde každá hrana má priradenú váhu (napr. dĺžku, čas alebo náklad). Tento algoritmus zaručuje nájdenie najkratšej cesty z počiatočného bodu ku všetkým ostatným vrcholom v grafe. Využíva prioritnú frontu, v ktorej vždy vyberá vrchol s najmenšou vzdialenosťou od začiatku. Z pohľadu výpočtovej náročnosti je však Dijkstra pomalší ako BFS a DFS, keďže jeho časová zložitosť je vyššia (napr. O(V log V + E) pri použití haldy). Jeho výhodou je však univerzálnosť, keďže funguje aj na zložitých mapách s rôznymi typmi terénu alebo dopravných obmedzení.

Ďalšou modernou alternatívou je heuristický algoritmus A\*, ktorý kombinuje výhody BFS a Dijkstrovho algoritmu. A\* využíva tzv. heuristiku – funkciu, ktorá odhaduje vzdialenosť od aktuálneho vrcholu k cieľu. Na základe tejto heuristiky dokáže A\* veľmi efektívne smerovať svoje vyhľadávanie priamo k cieľu, čím výrazne skracuje čas prehľadávania. Pokiaľ je heuristika správne navrhnutá (admisibilná a konzistentná), algoritmus A\* zaručuje nájdenie optimálnej cesty a zároveň pracuje rýchlejšie než tradičný Dijkstra. V prípade našej mriežkovej hry by sa dala použiť napríklad manhattanská vzdialenosť ako heuristická funkcia.

Záver

Algoritmus Breadth-First Search (BFS) predstavuje jednoduchý, no zároveň veľmi robustný nástroj na riešenie problémov súvisiacich s prehľadávaním grafov. Vďaka svojej deterministickej povahe, schopnosti nájsť najkratšiu cestu v neohodnotených grafoch a intuitívnemu princípu fungovania, je BFS ideálnym kandidátom na výučbu aj praktické nasadenie v rôznych oblastiach – od akademického prostredia až po aplikácie v priemyselnej robotike či počítačových hrách.

V rámci nášho projektu sme úspešne aplikovali BFS na plánovanie pohybu agenta v dvojrozmernom prostredí s prekážkami. Pomocou textovej vizualizácie bolo možné pozorovať správanie algoritmu v reálnom čase, čím sa prehľadávanie priestoru stalo nielen názorným, ale aj zábavným spôsobom výučby. Implementácia navyše ponúkla možnosť porovnania výkonu manuálneho ovládania a automatizovaného riešenia, čo umožňuje používateľom priamo pochopiť výhody algoritmického prístupu.

Na základe tejto práce sa ukazuje, že aj jednoduché algoritmy môžu byť pri správnej integrácii do vizualizačného prostredia silným nástrojom na demonštráciu zložitých výpočtových konceptov.

Referencie

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009, pp. 594–602.

[2] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Boston, MA, USA: Addison-Wesley, 2011, pp. 526–529.

[3] M. D. McIlroy, “Breadth-first search strategies and applications,” *ACM Comput. Surv.*, vol. 45, no. 2, pp. 12:1–12:28, Jun. 2013, doi: 10.1145/2431211.2431213.

[4] J. Smith and A. Brown, “Pathfinding in games using BFS: Efficiency and visualization,” *IEEE Trans. Games*, vol. 14, no. 1, pp. 45–52, Jan. 2022, doi: 10.1109/TG.2022.3140010.

[5] K. Patel, “A comparative study of BFS and DFS in maze-solving AI agents,” in *Proc. 10th Int. Conf. Artificial Intelligence Gaming*, Berlin, Germany, 2020, pp. 101–107, doi: 10.1109/ICGA.2020.9356123.

[6] J. Holdsworth, The Nature of Breadth-First Search, Tech. Rep. 99-1, School of Computer Science, Mathematics, and Physics, James Cook University, Australia, 1999.

[7] HackerEarth, “Breadth-First Search (BFS) Tutorial,” HackerEarth, [Online]. Dostupné: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

, [prístup dňa: 14. apríl 2025].

[8] Khan Academy, “The Breadth-First Search algorithm,” Khan Academy, [Online]. Dostupné: https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/the-breadth-first-search-algorithm

, [prístup dňa: 14. apríl 2025].

[9] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010, pp. 89–91.

[10] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

1. [↑](#footnote-ref-1)