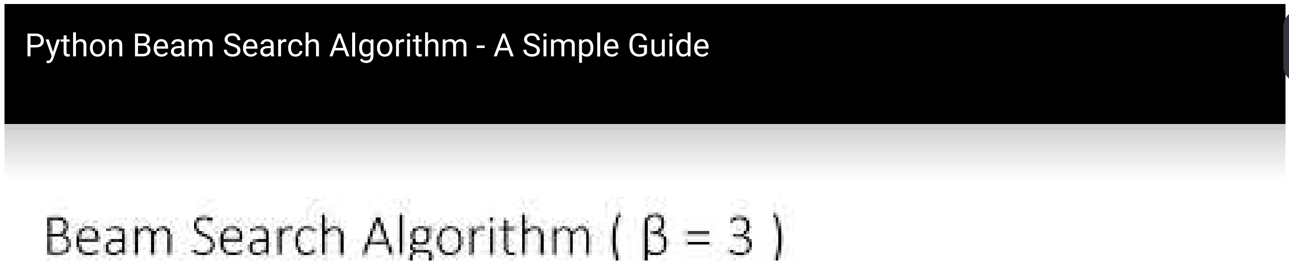
finxter

Menu

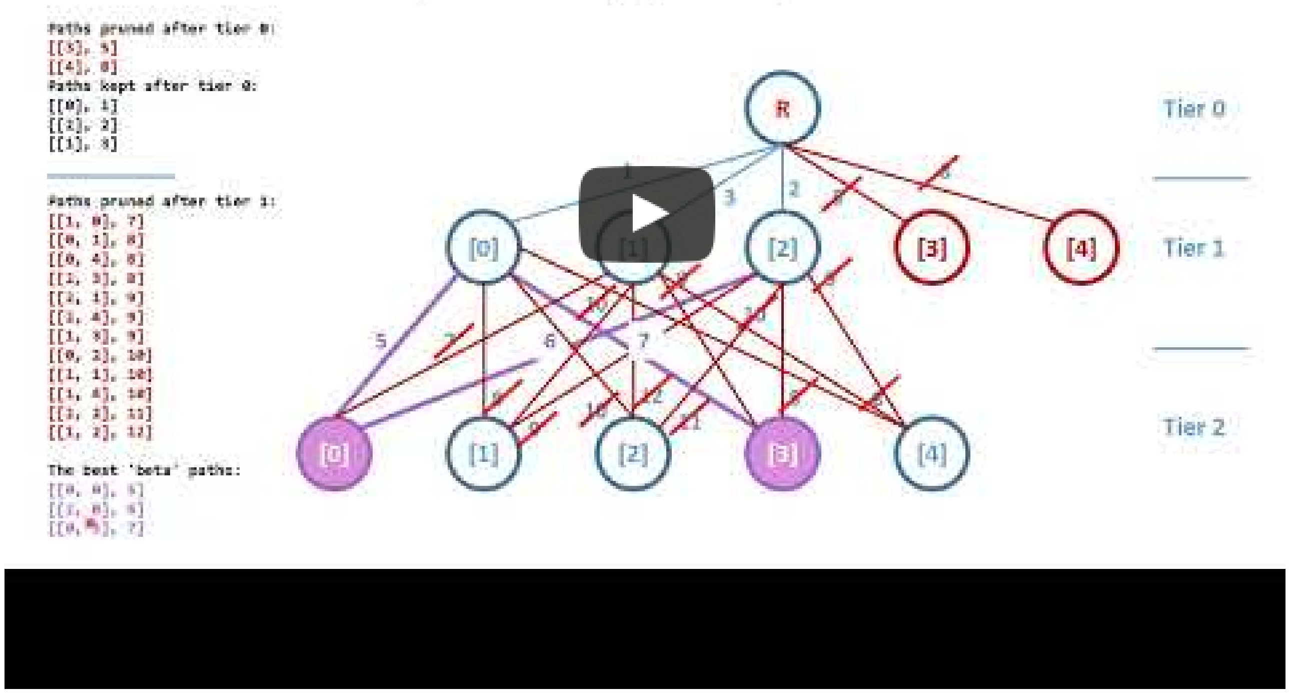
Python Beam Search Algorithm

February 22, 2022 by **Matija Horvat**

★★★★★ 5/5 - (1 vote)


Python Beam Search Algorithm - A Simple Guide
Beam Search Algorithm (β = 3)

Ask FinxterGPT (GPT-4)



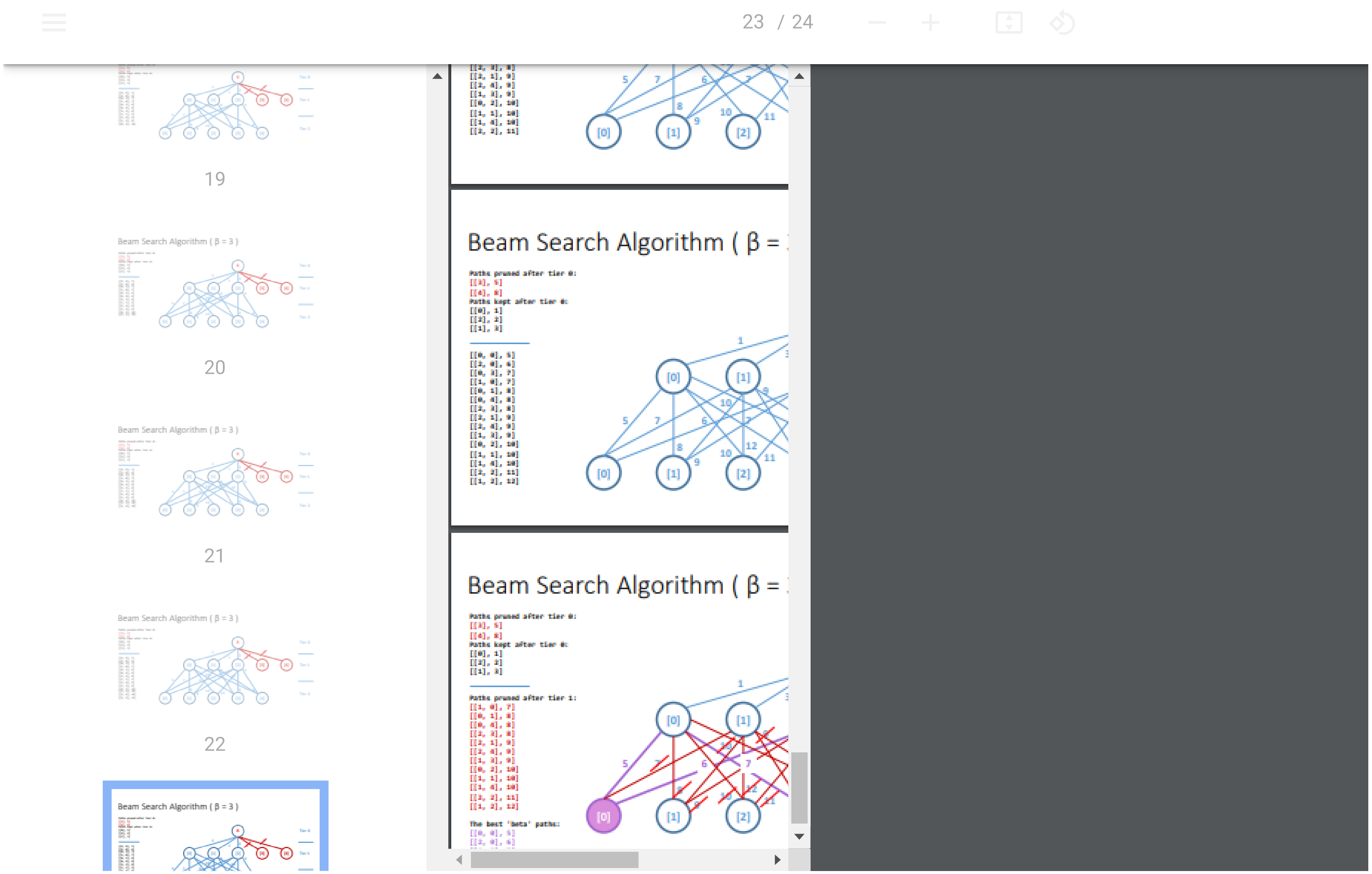
```
Paths pruned after tier 0:
[[3], 5]
[[4], 6]
Paths kept after tier 0:
[[0], 1]
[[2], 2]
[[1], 3]

Paths pruned after tier 1:
[[1, 0], 7]
[[0, 1], 8]
[[0, 4], 8]
[[2, 3], 8]
[[2, 1], 9]
[[2, 4], 9]
[[1, 3], 9]
[[0, 2], 10]
[[1, 1], 10]
[[1, 4], 10]
[[2, 2], 11]
[[1, 2], 12]

The best 'beta' paths:
[[0, 0], 5]
[[2, 0], 6]
[[0, 3], 7]
```

You can check out the slide deck here to get a first intuition on how the algorithm works:

Ask FinxterGPT (GPT-4)


23 / 24
Beam Search Algorithm (β = 3)
19
Beam Search Algorithm (β = 3)
20
Beam Search Algorithm (β = 3)
21
Beam Search Algorithm (β = 3)
22
Beam Search Algorithm (β = 3)
[0]
[1]
[2]

```
[[2, 3], 8]
[[2, 1], 9]
[[2, 4], 9]
[[1, 3], 9]
[[0, 2], 10]
[[1, 1], 10]
[[1, 4], 10]
[[2, 2], 11]
```

Beam Search Algorithm (β =

```
Paths pruned after tier 0:
[[3], 5]
[[4], 8]
Paths kept after tier 0:
[[0], 1]
[[2], 2]
[[1], 3]
```

```
[[0, 0], 5]
[[2, 0], 6]
[[0, 3], 7]
[[1, 0], 7]
[[0, 1], 8]
[[0, 4], 8]
[[2, 3], 8]
[[2, 1], 9]
[[2, 4], 9]
[[1, 3], 9]
[[0, 2], 10]
[[1, 1], 10]
[[1, 4], 10]
[[2, 2], 11]
[[1, 2], 12]
```

Beam Search Algorithm (β =

```
Paths pruned after tier 0:
[[3], 5]
[[4], 8]
Paths kept after tier 0:
[[0], 1]
[[2], 2]
[[1], 3]
```

```
Paths pruned after tier 1:
[[1, 0], 7]
[[0, 1], 8]
[[0, 4], 8]
[[2, 3], 8]
[[2, 1], 9]
[[2, 4], 9]
[[1, 3], 9]
[[0, 2], 10]
[[1, 1], 10]
[[1, 4], 10]
[[2, 2], 11]
[[1, 2], 12]

The best 'beta' paths:
[[0, 0], 5]
[[2, 0], 6]
```

Python-blog-Beam-Search-Algorithm Download

Ask FinxterGPT (GPT-4)

Before we'll dive into the algorithm and the Python implementation, let's first skim over some related graph tutorials you may enjoy and that may help your understanding!

Related Graph Tutorials

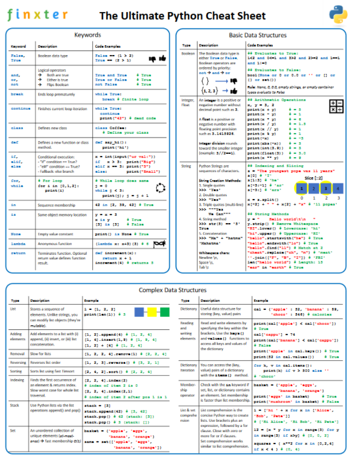
This algorithm is part of our graph algorithm tutorials:

- **Breadth-First Search (BFS) Algorithm in Python**
- **Python Depth-First Search (DFS) Algorithm**
- **Iterative Deepening Depth-First Search (DFS) Algorithm in Python**
- **Python Best-First Search Algorithm**
- **Python Dijkstra Algorithm**
- **Python A* Algorithm**
- **Jump Search Algorithm in Python**
- **Python Backtracking**

Ask FinxterGPT (GPT-4)

- [Python Beam Search Algorithm]

Each of these tutorial links opens in a new browser tab.


finxter The Ultimate Python Cheat Sheet

What is the Beam Search Algorithm?

The beam search algorithm is an informed search algorithm it is a more flexible variant of the previously explained best-first search algorithm. The beam search algorithm can take multiple paths into each iteration, ordered and selected by their path length.

Reminder: informed search algorithms use some kind of auxiliary information to guide their search strategy. Not being statically determined upfront makes them an interesting choice for a wide range of applications. However, their performance is greatly determined by the quality of auxiliary information, commonly known in computer science as *heuristic* function, *h(vertex)*.

The same as its base algorithm, the best-first search algorithm, the **beam search algorithm** uses a *greedy*, hence best-first approach, where the next β path choices are determined by their current length, rather than the overall quality of the path.

Ask FinxterGPT (GPT-4)

Symbol β (reads as “beta”) stands for the width of the beam, i.e. the number of the shortest (cheapest) paths to be taken into the next iteration of the algorithm, while all the other paths are being pruned.

What is the Purpose of Beam Search?

As a more flexible variant of the best-first search algorithm, the beam search algorithm inherits some of its predecessor’s fundamental properties. However, depending on the β, the algorithm can perform as both pure best-first search algorithm (β=1), pure breadth-first search algorithm (β=∞), and of course, anything in between.

Applications: The beam search algorithm is commonly used in applications such as machine translation, where there is possibly more than one good enough solution.

Ask FinxterGPT (GPT-4)

Except for its robustness, the most notable property of the beam search algorithm is its ability to maintain the manageability and usability of systems with limited resources in dealing with large and dense graphs.

How Does Beam Search Work?

The beam search algorithm starts the ***graph traversal*** by marking the start vertex as visited, i.e. putting it in the dictionary and placing it into the *priority queue* of candidate vertices.

We will use the term *explored*, which is synonymous with the terms *expanded* or *extended* in other literature.

Vertex priority determines the best β vertices to be kept for the next iteration. Howev Ask FinxterGPT (GPT-4) selection will be done first by expanding all the neighboring vertices for each vertex in the current tier.

Then, the best β paths will be kept and taken to the next iteration.

The cycle of choosing, exploring, and populating the priority queue continues, until the priority queue becomes exhausted. At that point, the beam search algorithm stops its execution.

Since the heuristic function greatly influences the algorithm performance, the function’s accuracy is crucial.

What are the Properties or Beam Search?

The main property of the beam search algorithm lies in its versatility, i.e. the fact that it can switch between the best-first search and breadth-first search approach of traversing the graph.

Ask FinxterGPT (GPT-4)

Its performance depends on the quality of the heuristic function, which in most cases represents the distance estimation from the goal vertex. The choice of heuristic function can influence the algorithm to find the shortest possible path to the goal vertex, to never complete the search — and everything in between these two extremes.

How is Beam Search Implemented in Python?

The implementation of our beam search algorithm is achieved by the function `beam_search()`.

For a more self-contained educational showcase, we will omit the commonly used graph data structure and introduce a few simplifications.

Ask FinxterGPT (GPT-4)

- First, we will assume densely connected vertices (with many-to-many connections).

- Second, we will define a fixed matrix representing distances, or weights between individual vertices in each tier.
- Third, each element of the distance matrix is composed of two parts: the first one is a list of distances from any previous vertex to its neighboring vertices, where vertices are determined by the indices of each distance, e.g. in a list `[12, 13, 14]`, distance to vertex 0 is 12, and distances to vertices 1 and 2 are 13 and 14.

The function `beam_search()` takes only two parameters:

- The `distances` parameter takes an initialized `numpy.array` object.
- The `beta` parameter takes a number representing the beam width, which we choose between integer values of 1 and ∞ (a large enough number for practical purposes).

For a better understanding of the algorithm and its implementation, each step is precisely described in the code below:

Ask FinxterGPT (GPT-4)

```
from numpy import array
```

```python
# Uses a beam to search through the tree
def beam_search(distances, beta):
    # Defines an initial, dummy record structure represented by
    # visited vertices and the path length (so far),
    # traversed along each path.
    paths_so_far = [[list(), 0]]

    # Propagates through the neighbouring vertices tier by tier
    # (row by row). A vertex position is indicated by its
    # index in each row (dists).
    for idx, tier in enumerate(distances):
        if idx > 0:
            print(f'Paths kept after tier {idx-1}:')
            print(*paths_so_far, sep='\n')
        paths_at_tier = list()

        # Follows each path.
        for i in range(len(paths_so_far)):
            # Reads the current path and its length (sum of distances).
            path, distance = paths_so_far[i]

            # Extends the path for every possible neighboring vertex
            # at the current tier.
            for j in range(len(tier)):
                path_extended = [path + [j], distance + tier[j]]
                paths_at_tier.append(path_extended)

        # Sorts the newly generated paths by their length.
        paths_ordered = sorted(paths_at_tier, key=lambda element: element[1])
```

Ask FinxterGPT (GPT-4)

```
        # The best 'beta' paths are preserved.
        paths_so_far = paths_ordered[:beta]
        print(f'\nPaths pruned after tier {idx}: ')
        print(*paths_ordered[beta:], sep='\n')

    return paths_so_far


# Define a distance matrix of 10 tiers
dists = [[1, 3, 2, 5, 8],
         [4, 7, 9, 6, 7]]
dists = array(dists)

# Calculates the best paths by applying a beam of width 'beta'.
best_beta_paths = beam_search(dists, 3)

# Prints the best 'beta' paths.
print('\nThe best \'beta\' paths:')
for beta_path in best_beta_paths:
    print(beta_path)
```

The test run gave us the output (for β = 3):

```
Paths pruned after tier 0:
[[3], 5]
[[4], 8]
Paths kept after tier 0:
[[0], 1]
[[2], 2]
```

Ask FinxterGPT (GPT-4)

```
[[1], 3]

Paths pruned after tier 1:
[[1, 0], 7]
[[0, 1], 8]
[[0, 4], 8]
[[2, 3], 8]
[[2, 1], 9]
[[2, 4], 9]
[[1, 3], 9]
[[0, 2], 10]
[[1, 1], 10]
[[1, 4], 10]
[[2, 2], 11]
[[1, 2], 12]

The best 'beta' paths:
[[0, 0], 5]
[[2, 0], 6]
[[0, 3], 7]
```

The resulting output shows the intermediate and final paths in a list of vertex indices and the total path length.

Based on the output, we can see that the search started from the root vertex and that in the first iteration (tier 0) the `beam_search()` has pruned two and kept three (shortest) paths, marked l

Ask FinxterGPT (GPT-4)

the vertex indices and the appropriate total path length so far.

In the second iteration, the `beam_search()` has pruned twelve and kept three (shortest) paths, marked by the vertex indices and the appropriate total path length so far. Since our `dists` matrix has only two tiers (I've kept it short for educational purposes), the algorithm ends here.

The final result is also displayed as The best 'beta' paths.

After setting the β to 1 and a re-run, we got a result matching a best-first search algorithm:

```
Paths pruned after tier 0:
[[2], 2]
[[1], 3]
[[3], 5]
[[4], 8]
Paths kept after tier 0:
[[0], 1]

Paths pruned after tier 1:
```

Ask FinxterGPT (GPT-4)

```
[[0, 3], 7]
[[0, 1], 8]
[[0, 4], 8]
[[0, 2], 10]

The best 'beta' paths:
[[0, 0], 5]
```

On the contrary, a large β (larger than number of distances in each tier multiplied, e.g. 5 x 5) would yield a breadth-first search algorithm behavior, where no pruning occurs:

```
Paths pruned after tier 0:

Paths kept after tier 0:
[[0], 1]
[[2], 2]
[[1], 3]
[[3], 5]
[[4], 8]

Paths pruned after tier 1:



The best 'beta' paths:
[[0, 0], 5]
[[2, 0], 6]
[[0, 3], 7]
[[1, 0], 7]
```

Ask FinxterGPT (GPT-4)

```
[[0, 1], 8]
[[0, 4], 8]
[[2, 3], 8]
[[2, 1], 9]
[[2, 4], 9]
[[1, 3], 9]
[[3, 0], 9]
[[0, 2], 10]
[[1, 1], 10]
[[1, 4], 10]
[[2, 2], 11]
[[3, 3], 11]
[[1, 2], 12]
[[3, 1], 12]
[[3, 4], 12]
[[4, 0], 12]
[[3, 2], 14]
[[4, 3], 14]
[[4, 1], 15]
[[4, 4], 15]
[[4, 2], 17]
```

Efficiency analysis

Ask FinxterGPT (GPT-4)

The algorithm's worst-case time complexity depends on β and lies between *O(bd)* (b

pure best-first search algorithm) and *O(|V| + |E|)* (behaves like a pure breadth-first search algorithm). It is determined by the heuristic function and the factor β.

The algorithm's worst-case space complexity also depends on β and lies between *O(bd)* (behaves like a pure best-first search algorithm) and *O(|V|)* (behaves like a pure breadth-first search algorithm).

Conclusion

In this article, we learned about the beam search algorithm.

- First, we explained what a beam search algorithm is.
- Second, we took a look at what are its common purposes and applications.
- Third, we went through an explanation of how the algorithm works.
- Fourth, we examined the algorithm's main properties.

Ask FinxterGPT (GPT-4)

- Fifth, we went through the implementation of the algorithm. We also tested the algorithm by calling its main function, beam_search(), and analyzed its steps of execution for the smallest, middle, and largest (“infinite”) β scenarios.
- Sixth, we analyzed the algorithm efficiency.

In the end, we concluded that the algorithm’s efficiency may be optimal if it behaves like a breadth-first search algorithm, although such mode of operation would defeat its initial purpose — to reduce the space complexity and data storage requirements.

In other modes of operation, the algorithm is not guaranteed to perform optimally and might also take a virtually infinite time in reaching the solution, especially for $\beta = 1$.

However, this behavior can be prevented by constructing the appropriate heuristic function using the relevant knowledge about the graph and vertice relationships.

Ask FinxterGPT (GPT-4)

FREE: Download Python Cheat Sheets (PDF)

Academy Course – Mastering the Top 10 Graph Algorithms

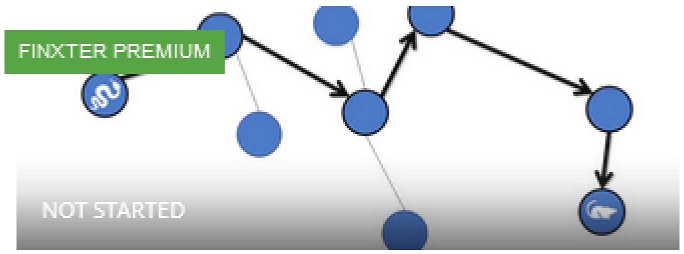
If you want to improve your fundamental computer science skills, there's nothing more effective than **studying algorithms**.

To help you master the **most important graph algorithms**, we've just launched the "Top 10 Algorithms" course at the Finxter Computer Science Academy. This great course from Finxter Star Creator Matija ⭐ teaches you the most important graph algorithms such as BFS, DFS, A*, and Dijkstra.

Ask FinxterGPT (GPT-4)

Understanding these algorithms will not only make you a better coder, but it’ll also lay a strong foundation on which you can build your whole career as a computer scientist.

Click the screenshot to find out more:



Top 10 Graph Algorithms in Python

This great course from Finxter Star Creator Matija ⭐ teaches you the most important graph algorithms such as BFS, DFS, A*, and Dijkstra. Understanding these algorithms will not only make you a better coder, it'll lay a strong foundation on which you can build your whole career as a computer scientist.

11 Lessons - Intermediate

Learn More

Ask FinxterGPT (GPT-4)

Matija Horvat

I'm an experienced computer science engineer and technology enthusiast dedicated to understanding how the world works and using my knowledge and ability to advance it. I'm focused on becoming an expert in Solidity and crypto technology, with a passion for coding, learning, and contributing to the Finxter mission of increasing the collective intelligence of humanity.

Ask FinxterGPT (GPT-4)

Algorithms, Computer Science, Graph Theory, Python

Be on the Right Side of Change 🚀

The world is changing exponentially. Disruptive technologies such as AI, crypto, and automation eliminate entire industries. 🤖

Do you feel uncertain and afraid of being replaced by machines, leaving you without money, purpose, or value? Fear not! There a way to not merely survive but *thrive* in this new world!

Finxter is here to help you stay ahead of the curve, so you can keep winning as paradigms shift.

Learning Resources 🧑‍💻

⭐ Boost your skills. Join our free email academy with daily emails teaching exponential with 1000+ tutorials on AI, data science, Python, freelancing, and Blockchain development!

Ask FinxterGPT (GPT-4)

Join the **Finxter Academy** and unlock access to premium courses 👑 to certify your skills in exponential technologies and programming.

Ask FinxterGPT (GPT-4)

New Finxter Tutorials:

Python Create Dictionary – The Ultimate Guide

Top 10 LLM Training Datasets – It's Money Laundering for Copyrighted Data!

Python Multiprocessing Pool [Ultimate Guide]

OpenAI API Functions & Embeddings Course (7/7): Sentiment Analysis using Embeddings

OpenAI API Functions & Embeddings Course (6/7): Similarity Comparison with Embeddings

OpenAI API Functions & Embeddings Course (5/7): Calling Functions That Do Not Exist to Extract Structured Data

OpenAI API Functions & Embeddings Course (4/7): Database Querying using ChatGPT

OpenAI API Functions & Embeddings Course (3/7): Multiple Functions and Multiple Calls

OpenAI API Functions & Embeddings Course (2/7): Function Calls with Parameters

OpenAI API Functions & Embeddings Course (1/7): Simple Function Request

Finxter Categories:

Select Category

About

Impressum

Privacy

Ask FinxterGPT (GPT-4)

- Terms
- Earnings Disclaimer
- Puzzles
- Academy
- Books & Courses
- Finxter Instagram
- Finxter Twitter
- Finxter Facebook
- Finxter YouTube
- Email Newsletter Sponsorship

- Finxter Exponential Academy 🚀
- Books
- Puzzles
- User Stories 🤎
  - What Our Users Say
  - About Finxter
  - Finxter Feedback from ~1000 Python Developers

Ask FinxterGPT (GPT-4)

- Video Reviews
- Course Review from Cristian
- Course Review from Adam
- About Chris

Start Here ⭐

Ask FinxterGPT (GPT-4)