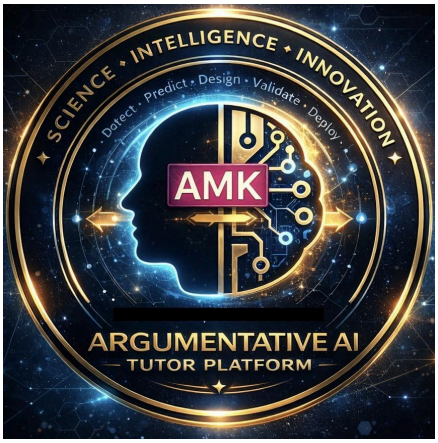


# Benchmark Lecture Note 14

Algorithms, Shortest-Path Methods, Natural Language Processing, Python Dictionaries as a Research Method, and Farthest-First Clustering

AI Certification Program • AMK Research Lab Program

**Purpose.** This benchmark note gives learners one integrated view of algorithmic thinking in AI: graph search for routing, natural language processing for text understanding, dictionary-based coding for lightweight research, and farthest-first clustering for interpretable unsupervised learning. The note also aligns each topic to governed and trustworthy deployment.

## Learning outcomes

| Topic | What the learner should master | Applied AMK direction |
|---|---|---|
| Algorithms | Explain step-by-step procedures, complexity, and correctness intuition. | Build disciplined AI reasoning and reproducible workflows. |
| Shortest path | Compare BFS, Dijkstra, Bellman–Ford, and A* for routing problems. | Use graphs in logistics, cyber paths, or knowledge navigation. |
| NLP | Tokenize, normalize, count, and interpret text with Python tools. | Support tutors, safety filters, and research analytics. |
| Python dictionaries | Use key–value coding to organize themes, labels, and counts. | Apply low-cost research coding before advanced ML. |
| Farthest-first clustering | Explain center selection and why it is useful for diverse seeds. | Support interpretable clustering and capstone prototypes. |

## AMK governance context

This lecture note sits inside the AMK S²I and HACE benchmark philosophy: science for rigor, intelligence for decision quality, and innovation for governed deployment with human authority in the loop.

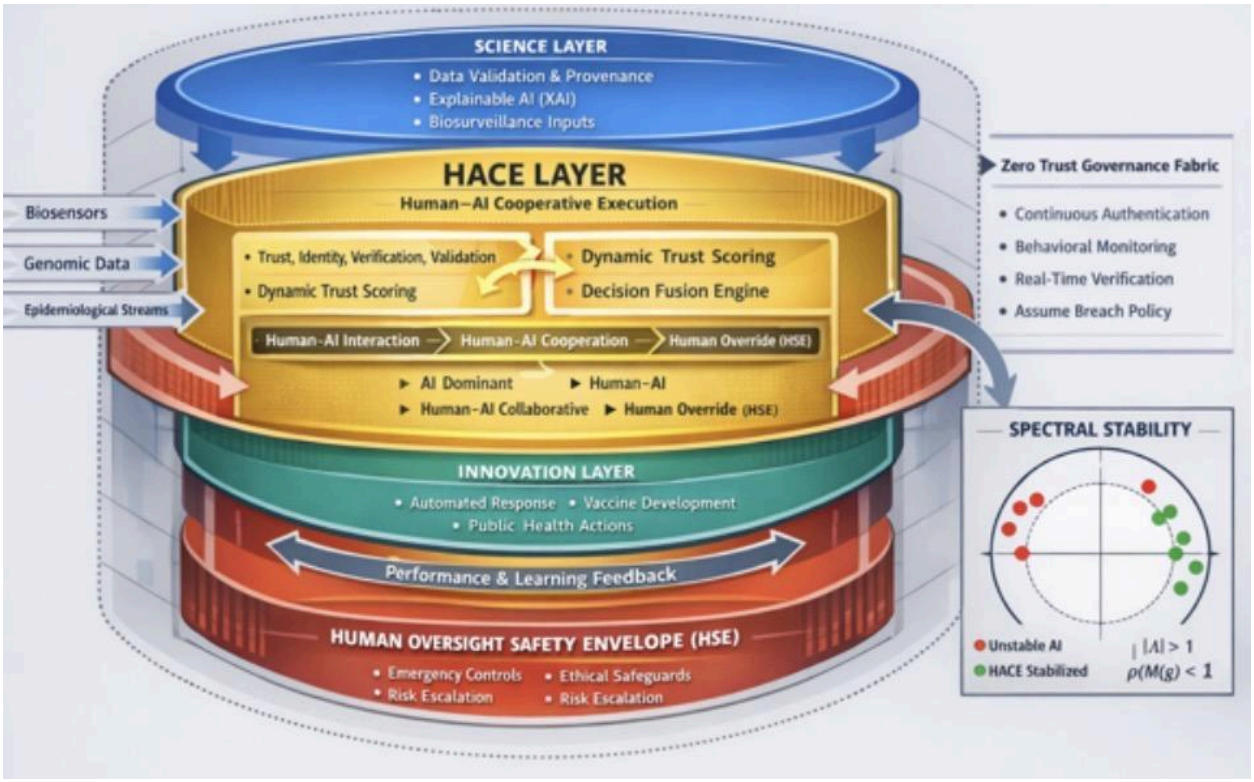


Figure 1. AMK HACE governance visual used as the benchmark context for safe AI learning and deployment.

# 1. Algorithm foundations for AI

An algorithm is a finite, ordered, and reproducible procedure for solving a problem. In AI education, algorithms matter because they connect theory to implementation: they define what steps are executed, what information is updated, and how performance is measured.

Three guiding questions help learners evaluate any algorithm:

- What input does it require, and what output does it produce?
- How does the state change after each step?
- How much time and memory does it consume as the problem grows?

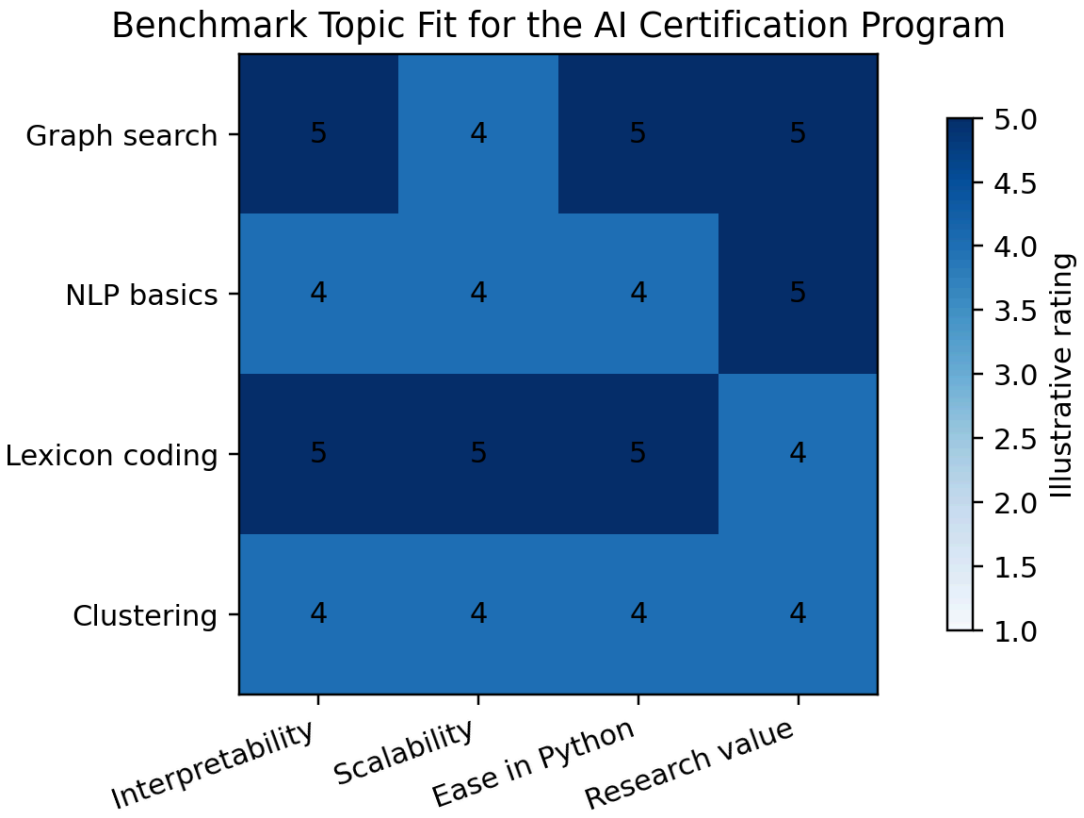


Figure 2. Illustrative benchmark fit of the lecture-note topics for certification and research use.

## 2. Shortest-path algorithms

Shortest-path algorithms compute the least-cost route through a graph. In AI and data science, they support route optimization, recommendation paths, knowledge-graph traversal, network defense, and workflow planning.

### Key intuition:

A graph contains nodes and edges. If edges have no weights, breadth-first search (BFS) often suffices. If edges carry costs, Dijkstra's algorithm is a standard choice when all weights are nonnegative. Bellman–Ford is more flexible because it can handle negative weights, although it is typically slower. A* introduces a heuristic so that search can focus on promising directions toward a target.

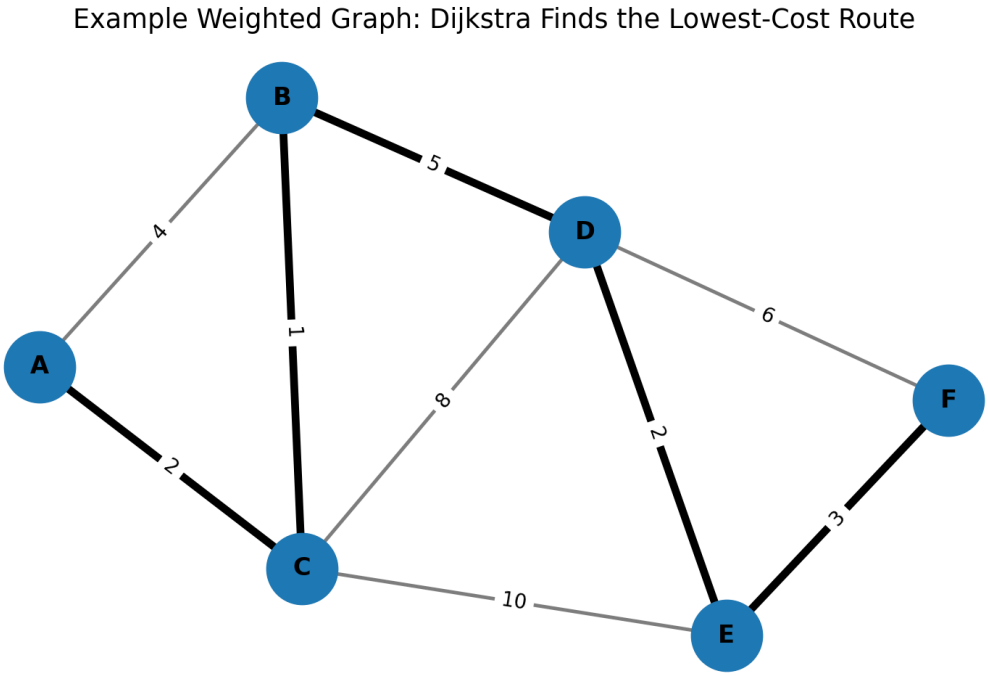


Figure 3. Weighted graph example for route-finding demonstrations.

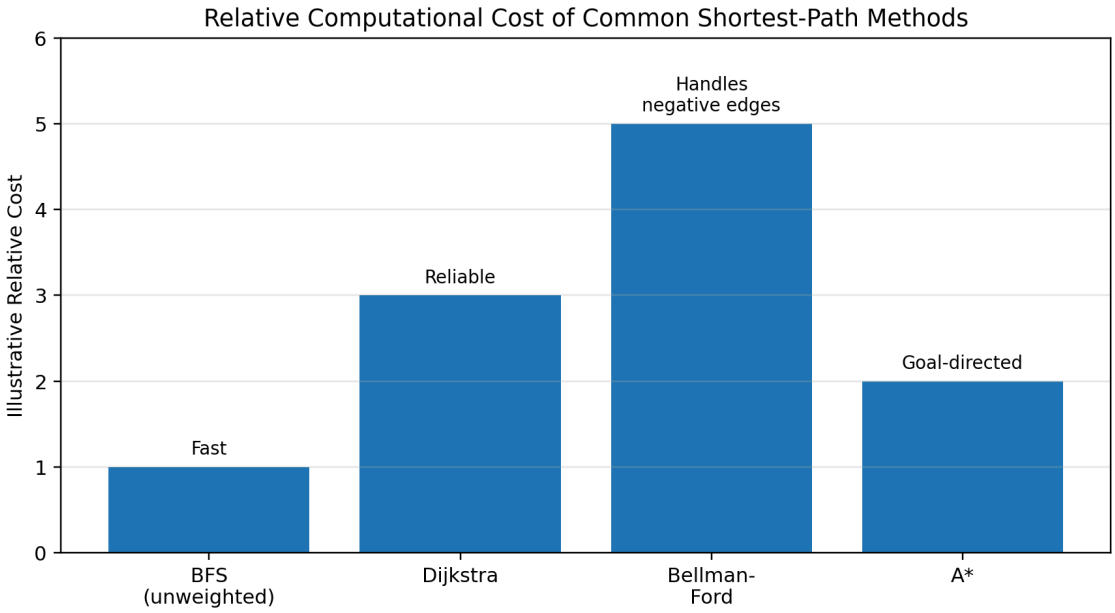


Figure 4. Relative instructional comparison of common shortest-path methods.

## Comparison table

| Method | Best use case | Strength | Caution |
|---|---|---|---|
| BFS | Unweighted graphs | Simple and fast | Ignores weighted costs |
| Dijkstra | Weighted routing with nonnegative edges | Reliable baseline | Not suited to negative weights |
| Bellman–Ford | Graphs with possible negative weights | Flexible and explicit | Higher computational cost |

| Method | Best use case | Strength | Caution |
|---|---|---|---|
| A* | Targeted route search with a heuristic | Often faster in practice | Depends on heuristic quality |

### Sample Python: Dijkstra with NetworkX

```
import networkx as nx

G = nx.Graph()
G.add_weighted_edges_from([
    ("A", "B", 4), ("A", "C", 2), ("B", "C", 1),
    ("B", "D", 5), ("C", "D", 8), ("D", "E", 2), ("E", "F", 3)
])

path = nx.shortest_path(G, source="A", target="F", weight="weight")
cost = nx.shortest_path_length(G, source="A", target="F", weight="weight")
print(path, cost)
```

## 3. Natural language processing (NLP)

NLP teaches machines to work with human language. In practical benchmarking, students should understand tokenization, normalization, lexicon or dictionary coding, part-of-speech tagging, entity extraction, and simple classification. These steps are enough to build useful prototypes before jumping into large language models.

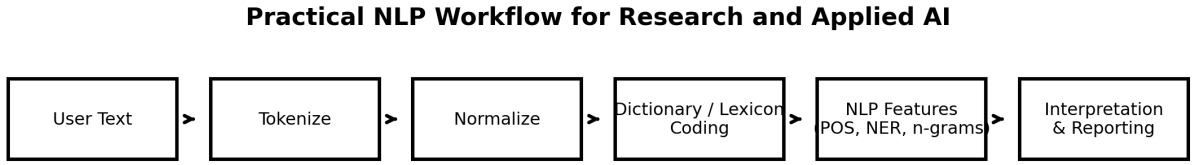


Figure 5. A practical NLP workflow from raw text to interpretable reporting.

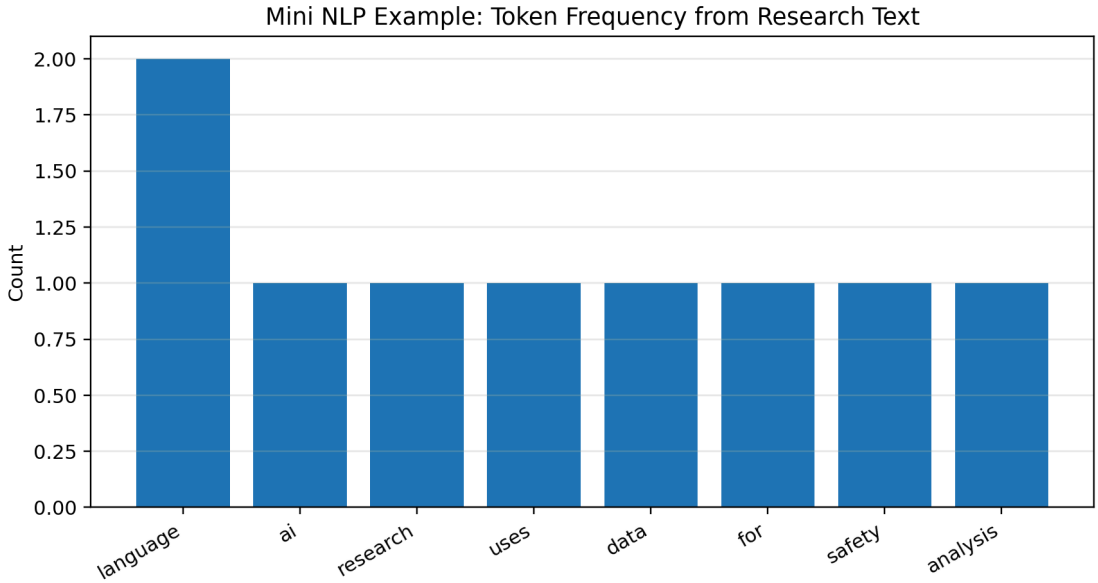

Figure 6. Simple token-frequency chart for an introductory research corpus.

### What students should practice

- Cleaning and normalizing text without losing meaning.
- Building interpretable features such as counts, keywords, and n-grams.
- Testing small lexicon-based methods before training heavier models.
- Checking outputs for bias, ambiguity, and unsupported claims.

### Sample Python: token counting with a dictionary

```
from collections import Counter
text = "AI safety and AI governance support responsible innovation"
tokens = [t.lower() for t in text.split()]
counts = Counter(tokens)
print(counts)
```

## 4. Python dictionaries as a lightweight research method

A Python dictionary stores key value pairs. In research, this becomes a practical coding structure: a researcher can map a theme to counts, a participant ID to labels, or a concept to notes. That makes dictionaries useful for transparent, low-cost exploratory analysis before advanced machine learning.

**Research method idea.** When a project is still exploratory, a dictionary can act as an interpretable coding frame. For example, the researcher can map themes such as “safety,” “ethics,” and “security” to frequencies or excerpts from user prompts.

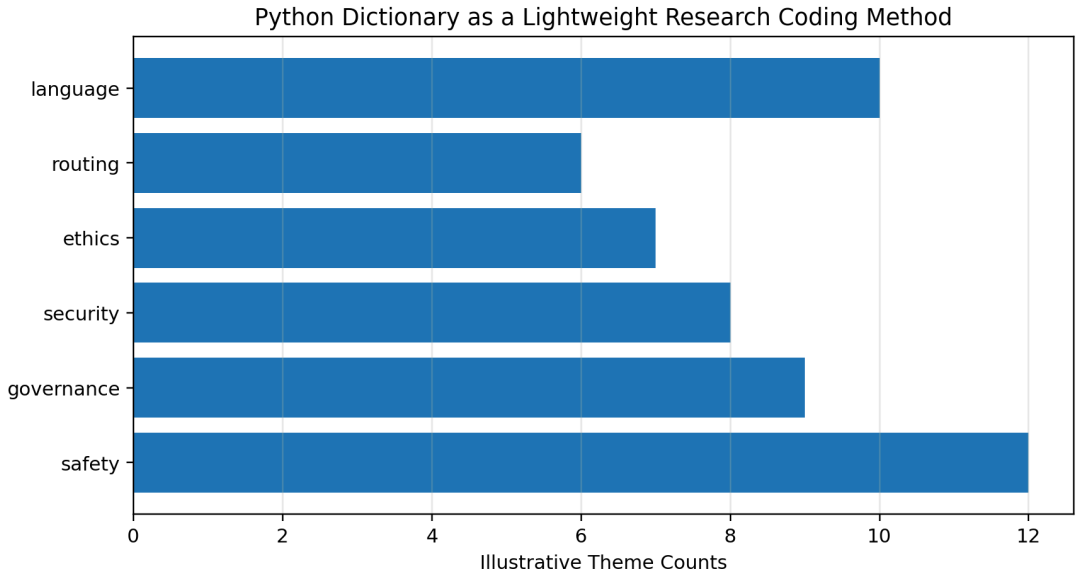


Figure 7. Dictionaries can support transparent and auditable theme coding.

| Dictionary key | Example value | Research use |
|---|---|---|
| theme | {"safety": 12, "ethics": 7} | Count concepts across prompt sets |
| participant_id | {"P01": "high concern"} | Store labels for interviews or surveys |
| document | {"doc3": ["risk","bias"]} | Track coded excerpts by source |

### Sample Python: dictionary-based coding

```
prompts = [
    "Safety matters in healthcare AI",
    "Governance and ethics reduce deployment risk",
    "Security and safety must be monitored"
]

lexicon = {"safety": 0, "governance": 0, "ethics": 0, "security": 0}
for prompt in prompts:
    lower = prompt.lower()
    for key in lexicon:
        if key in lower:
            lexicon[key] += 1

print(lexicon)
```

## 5. Farthest-first clustering

Farthest-first clustering chooses a first center, then repeatedly selects the point farthest from the centers already chosen. The idea is attractive in education because learners can see why the selected centers are diverse. It is closely associated with the k-center problem and is often taught as an interpretable greedy strategy and as a useful seeding idea for later clustering stages.

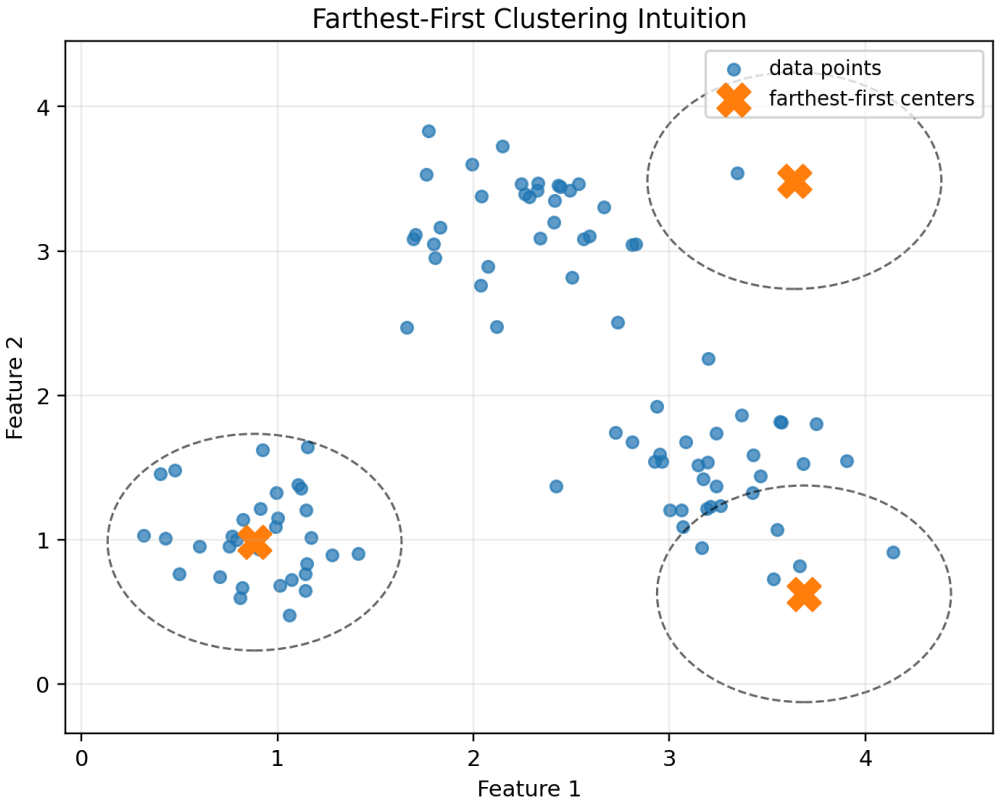


Figure 8. Farthest-first center selection illustrated on a toy dataset.

### Why it matters in AI training

- It gives learners an intuitive bridge from greedy algorithms to clustering.
- It encourages coverage and diversity instead of selecting overly similar seeds.
- It can be used to discuss approximation, interpretability, and initialization quality.

### Sample Python: simple farthest-first seeding

```
import numpy as np

points = np.array([[0,0],[1,0],[0,1],[5,5],[6,5],[5,6]])
centers = [points[0]]

for _ in range(2):
    d = np.min([np.sum((points - c)**2, axis=1) for c in centers], axis=0)
    centers.append(points[np.argmax(d)])

print(np.array(centers))
```

## 6. Benchmark alignment and capstone pathways

This lecture note is especially useful because it joins algorithmic depth with research readiness. Students can move from path-finding and clustering to text analytics and small-scale coding methods without losing interpretability.

**Responsible-AI reminder.** Even classical algorithms should be taught with governance in mind: document assumptions, validate outputs, keep human review for high-impact decisions, and preserve transparent audit trails for research and deployment.

## References


Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38, 293–306.

NIST. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0).

NIST. (2024). Generative AI Profile (AI 600-1) companion resource to the AI RMF.

NetworkX Developers. (2026). Shortest paths documentation.

Python Software Foundation. (2026). Python documentation: Data structures and mapping type dict.

Bird, S., Klein, E., & Loper, E. Natural Language Processing with Python / NLTK Book.

Explosion AI. (2025). spaCy usage documentation: linguistic features and models.

scikit-learn developers. (2026). Clustering and KMeans documentation.