Radical-Based Similarity Analysis of 日本語 Kanji Characters Using Tree Edit Distance with Dynamic Programming

Junior Nara Situmraong - 13524055
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: junioroppo64@gmail.com , 13524055@std.stei.itb.ac.id

***zAbstract*— Kanji is one of the most challenging aspects of learning Japanese due to the large number of characters and the high degree of visual similarity among them. Such similarities often lead to difficulties in character recognition and memorization for language learners. This study aims to analyze the similarity between Japanese kanji characters based on their constituent radicals. Each kanji character is represented as a tree structure that captures the hierarchical relationships among its radicals. Tree Edit Distance (TED) with a Dynamic Programming approach is then employed to calculate the minimum edit cost required to transform one tree into another, serving as a measure of similarity between kanji characters. A smaller edit distance indicates a higher degree of structural similarity. The proposed approach is expected to provide a more accurate representation of kanji similarity than methods that rely solely on visual appearance. The findings of this study may contribute to the development of Japanese language learning systems, similar-kanji recommendation tools, and radical-based character recognition applications.**

Keywords*— Kanji, Radical, Tree Edit Distance, Dynamic Programming, Similarity Analysis.

I. Introduction

Learning Japanese involves mastering three writing systems: Hiragana, Katakana, and Kanji. Among these, Kanji is often considered the most challenging for learners due to the large number of characters and their complex structures. Many kanji characters are composed of multiple radicals, which serve as the fundamental building blocks of the writing system. As a result, different kanji characters may share similar radicals and exhibit visually similar appearances, making them difficult to distinguish and memorize.

In Japanese language education, understanding the relationships between radicals is often considered an effective strategy for learning and remembering kanji. Radicals not only contribute to the visual structure of a character but can also provide clues about its meaning or classification. By recognizing common radicals across different characters, learners can develop associations that facilitate memorization and improve their ability to identify unfamiliar kanji.

The structural similarity between kanji characters can cause confusion, especially for beginner and intermediate learners. Therefore, analyzing kanji based on their radical composition may provide a more meaningful representation of character similarity than relying solely on visual appearance. This study proposes a radical-based similarity analysis of Japanese kanji characters by representing each character as a tree structure and measuring the similarity between characters using Tree Edit Distance (TED) with Dynamic Programming..

II. Theoritical Framework

A. Kanji (漢字)

Kanji (漢字) are logographic characters adopted from ancient Chinese writing and have been used in Japan since approximately the 5th century. Along with Hiragana and Katakana, Kanji forms one of the three primary writing systems used in the Japanese language. Kanji plays an important role in conveying the core meaning of words and is commonly used to write nouns, verb stems, adjective stems, and certain adverbs.

Each Kanji character possesses its own meaning and may represent a specific concept or idea. This characteristic allows written Japanese to convey information efficiently, as a single character can often express meaning that would otherwise require multiple phonetic symbols. However, the complexity of Kanji also presents significant challenges for learners.

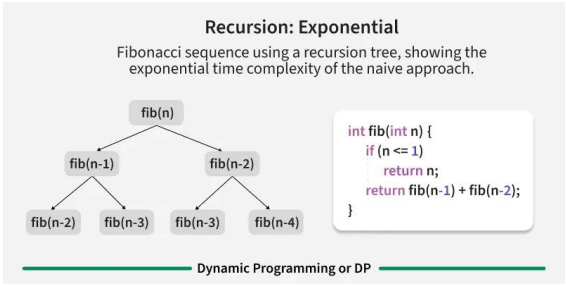
Source : https://crunchynihongo.com/introduction-to-kanji/ (accessed Jun 16, 2025).

One of the main difficulties in learning Kanji is that a single character may have multiple pronunciations depending on its context and usage. In addition, many Kanji characters are composed of smaller components known as radicals (部首). Radicals serve as the fundamental building blocks of Kanji and often provide clues regarding a character's meaning, category, or origin. Since different Kanji characters may share common radicals, they frequently exhibit structural similarities that can make them difficult to distinguish, especially for beginner learners.

Understanding the relationships between radicals and their arrangement within a Kanji character is therefore an important aspect of Kanji learning. In this study, radicals are utilized as the basis for representing Kanji structures and analyzing similarities between characters.

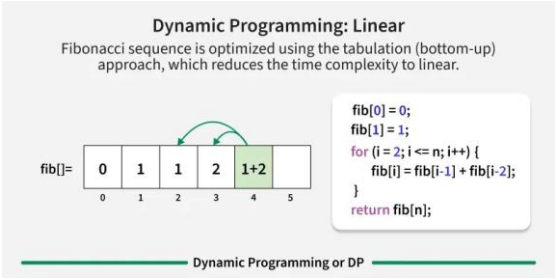
B. *Dynamic Programming*

Dynamic Programming (DP) is an algorithmic technique used to solve complex problems by decomposing them into smaller and overlapping subproblems. Instead of solving the same subproblem repeatedly, Dynamic Programming stores the results of previously computed subproblems and reuses them whenever needed. This approach significantly reduces redundant computations and improves overall efficiency.



Source : https://www.geeksforgeeks.org/dsa/dynamic-programming/ (accessed Jun 16, 2025).

Dynamic Programming is particularly effective for problems that exhibit two key properties: overlapping subproblems and optimal substructure. Overlapping subproblems occur when the same computations are performed multiple times during recursive execution, while optimal substructure means that the optimal solution to a problem can be constructed from the optimal solutions of its subproblems.



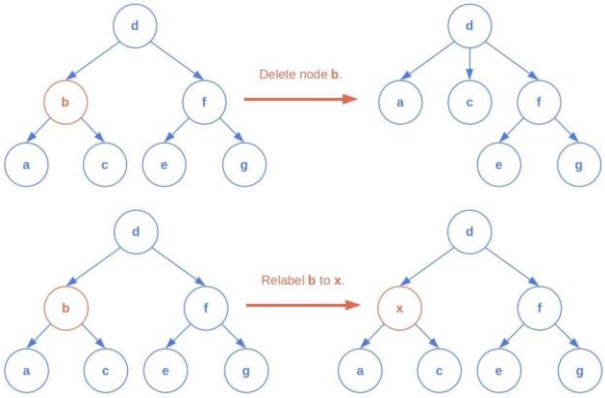
Source : https://www.geeksforgeeks.org/dsa/dynamic-programming/ (accessed Jun 16, 2025).

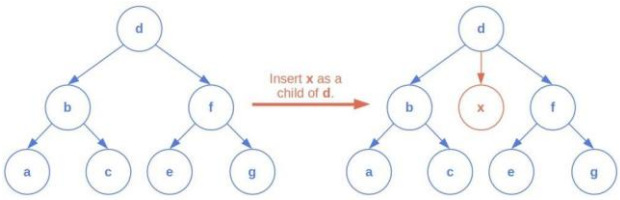
The technique is commonly implemented using either memoization (top-down approach) or tabulation (bottom-up approach). By storing intermediate results, Dynamic Programming can substantially reduce the time complexity of many algorithms. In this study, Dynamic Programming is employed to optimize the computation of Tree Edit Distance by avoiding repeated calculations of identical subtree comparisons, thereby improving the efficiency of similarity analysis between Kanji character structures.

C. *Tree Edit Distance*

Tree Edit Distance (TED) is an extension of the Edit Distance concept that measures the similarity between two tree structures. While Edit Distance calculates the minimum number of operations required to transform one string into another, Tree Edit Distance determines the minimum cost needed to transform one tree into another through a sequence of edit operations.

The common edit operations in Tree Edit Distance include node insertion, node deletion, and node substitution. The distance between two trees is defined as the minimum total cost of these operations required to make the trees identical. A smaller Tree Edit Distance indicates a higher degree of similarity between the compared tree structures.





Source : https://www.baeldung.com/cs/tree-edit-distance (accessed Jun 16, 2025).

Tree Edit Distance is commonly computed using Dynamic Programming to efficiently evaluate the costs of transforming subtrees. By storing intermediate results, the algorithm avoids redundant computations and significantly improves performance when comparing complex hierarchical structures.

In this study, each Kanji character is represented as a tree based on its radical composition. Tree Edit Distance is then used to measure the structural similarity between Kanji characters by comparing their corresponding radical trees.

D. KANJIDIC2

KANJIDIC2 is a comprehensive digital dictionary of Japanese Kanji maintained by the Electronic Dictionary Research and Development Group (EDRDG). The dataset is distributed in XML format and contains detailed information about Japanese Kanji characters, including their meanings, readings (On-yomi and Kun-yomi), stroke counts, grade levels, and radical classifications.

Due to its structured and machine-readable format, KANJIDIC2 is widely used in Japanese language processing and Kanji-related research. The dataset provides a reliable source of Kanji metadata that can be easily parsed and analyzed programmatically.

In this study, KANJIDIC2 is utilized as the primary source of Kanji characters. The information obtained from the dataset is combined with radical decomposition data from KRADFILE to construct tree representations of Kanji characters for similarity analysis using Tree Edit Distance.

E. KRADFILE

KRADFILE is a dataset developed by the Electronic Dictionary Research and Development Group (EDRDG) that provides radical decomposition information for Japanese Kanji characters. Unlike KANJIDIC2, which primarily contains Kanji metadata such as meanings and readings, KRADFILE focuses on identifying the component radicals that constitute each Kanji character.

Each entry in KRADFILE maps a Kanji character to a set of radicals or graphical components. These radicals represent the fundamental elements used in constructing more complex Kanji characters. By decomposing Kanji into their constituent radicals, KRADFILE enables structural analysis of character composition and relationships among different Kanji.

The dataset is particularly useful for studies involving Kanji similarity, classification, and character recognition. In this research, KRADFILE is used to obtain the radical composition of each Kanji character. The extracted radicals are then organized into tree representations, which serve as the input for Tree Edit Distance computation. This approach allows the structural similarity between Kanji characters to be measured based on their radical compositions rather than solely on their visual appearance.

III. METHOD

A. Load Data

The first stage of the proposed method involves loading Kanji data from the KANJIDIC2 and KRADFILE datasets. KANJIDIC2 is parsed from its XML format and indexed using a map data structure, where each Kanji character serves as the key and its corresponding XML node serves as the value. This indexing process enables efficient retrieval of Kanji information during subsequent processing stages.

Next, the KRADFILE dataset is loaded and parsed to obtain the radical decomposition of each Kanji character. The extracted data are stored in a map structure that associates each Kanji character with a collection of its constituent radicals. These radicals serve as the fundamental components for constructing the structural representation of Kanji characters.

```
def _load_kanjidic2(self):
   print("Memuat dan mengindeks
kanjidic2.xml...")
   tree = ET.parse(self.xml_path)
   root = tree.getroot()
   for character in root.findall("character"):
   literal_node = character.find("literal")
   if literal_node is not None and
literal_node.text:
      self.kanji_xml_map[literal_node.text] =
character
      print(f"-> Berhasil mengindeks
{len(self.kanji_xml_map)} karakter dari XML.")

def _load_kradfile(self):
   print("Memuat dan mengurai kradfile
(Dekomposisi Radikal)...")
   try:
      with open(self.krad_path, "r",
encoding="euc-jp") as f:
         self._parse_krad_lines(f)
   except UnicodeDecodeError:
      with open(self.krad_path, "r",
encoding="utf-8") as f:
         self._parse_krad_lines(f)
    print(f"-> Berhasil memuat dekomposisi
untuk {len(self.kanji_radical_map)}
karakter.")
```

After both datasets have been successfully loaded, the information from KANJIDIC2 and KRADFILE is combined to create a unified representation of each Kanji character. The resulting radical decomposition data are then used to construct tree structures, which form the basis for similarity computation using the Tree Edit Distance algorithm.

B. Tree Construction

After the KANJIDIC2 and KRADFILE datasets have been loaded into the system, the user can choose between two search modes: comparing two kanji characters or finding the most similar kanji characters to a given input kanji. The user then enters the kanji to be processed, and the system constructs a tree representation of the kanji based on the available radical decomposition information.

```
def build_tree(self, kanji, visited=None):
   radicals =
self.kanji_radical_map.get(kanji, [])
   if visited is None:
   visited = set()
      if kanji in visited:
   return None
   if not radicals:
      return None
   if len(visited) == 0 :
      root = TreeNode(kanji, "KANJI")
   else:
      root = TreeNode(kanji, "RADICAL")
   visited.add(kanji)
   for rad in radicals:
      if rad == kanji and (len(radicals) > 1
or len(visited) > 1):
         continue
      child = self.build_tree(rad, visited)
      if not child:
         child = TreeNode(rad, "RADICAL")
      root.children.append(child)
   return root
```

Each node in the tree represents either a kanji character or a radical. If a node has radical components according to the KRADFILE dataset, child nodes are created to represent those components. This decomposition process is performed recursively until leaf nodes are reached, which are kanji characters or radicals that have no further decomposition information available in the dataset. The resulting tree structure is then used as input for similarity computation using the Tree Edit Distance (TED) algorithm.

C. Tree Similarity Computation

After the tree structures have been constructed, the similarity between two Kanji characters is computed using the Tree Edit Distance (TED) algorithm. TED measures the minimum number of edit operations required to transform one tree into another. The supported operations consist of node insertion, node deletion, and node substitution.

The algorithm recursively compares the root nodes of both trees and computes the edit cost based on whether the node labels are identical. If the labels differ, a substitution cost of one is assigned; otherwise, the cost is zero. The algorithm then computes the distance between the child forests of both nodes using dynamic programming. Forest distance is calculated by considering the minimum cost among deletion, insertion, and substitution operations for each pair of child nodes.

To improve computational efficiency, memoization is employed to store previously computed tree and forest distances. This prevents redundant calculations when the same subtree pairs are encountered multiple times during recursion.

The final Tree Edit Distance value represents the structural difference between the two Kanji decomposition trees. A lower TED value indicates a higher degree of similarity between the Kanji characters.

The similarity score is then calculated using the following equation:

$$\text{Similarity} = (1 - \text{TED} / \text{MaxSize}) \times 100\%$$

where TED denotes the computed Tree Edit Distance and MaxSize represents the normalization factor based on the sizes of the compared trees. The resulting score ranges from 0% to 100%, where higher values indicate greater structural similarity.

For the Kanji search mode, the input Kanji tree is compared against all Kanji trees stored in the dataset. The similarity scores are calculated and ranked in descending order, and the Kanji characters with the highest similarity scores are returned as the search results.

D. Tree Similarity Computation

In the Kanji search mode, the objective is to identify Kanji characters that are structurally similar to a given input Kanji. After the decomposition tree of the input Kanji has been constructed, the system compares it against the decomposition trees of all Kanji characters available in the dataset using the Tree Edit Distance (TED) algorithm.

For each comparison, a similarity score is calculated based on the computed TED value. The similarity scores are stored together with their corresponding Kanji characters. Once all comparisons have been completed, the

results are sorted in descending order according to their similarity scores.

The ranked list enables the system to retrieve the Kanji characters that are most structurally similar to the input Kanji. The top-ranked Kanji characters are then presented to the user along with their corresponding similarity scores. This ranking process allows users to efficiently discover Kanji characters that share similar radical compositions and structural characteristics.

In the Kanji comparison mode, the system directly computes and displays the Tree Edit Distance and similarity score between the two input Kanji characters without performing a ranking process.

IV. Result

A. Test Case

The following table presents the test cases performed:

TABLE I. Test Case For 1 Kanji

NO	Test Case
1	木
2	林
3	明
4	森
5	架

TABLE II. Test Case For 2 Kanji

NO	Test Case	
	Kanji 1	Kanji 2
1	木	木
2	林	木
3	明	日
4	森	人
5	架	加

From the test cases presented above, it is expected to identify kanji that share similar radicals or overlapping structural components, allowing the Tree Edit Distance algorithm to calculate the similarity score based on the cost of transformation operations between the radical trees.

B. Result

The results of the structure tree presented in Table I are shown below:

TABLE III. Tree Structure For 1 Kanji

NO	Tree Structure
1	木 └──木
2	林 └──木
3	明 └──月 └──日
4	森 └──木
5	架 └──口 └──口 └──口 └──木 └──力

The results of the test cases presented in Table I are shown below:

TABLE IV. RESULT TABLE 1 TEST CASE

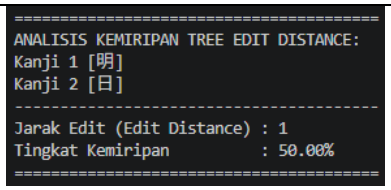
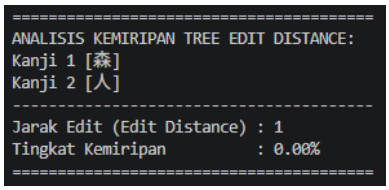
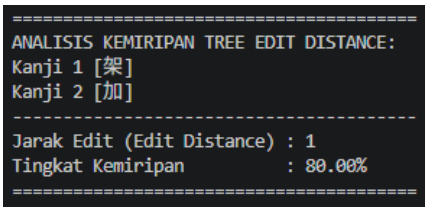
NO	Result	Explanation
1	1. 森 - Similarity: 100.00% 2. 林 - Similarity: 100.00% 3. 杰 - Similarity: 100.00% 4. 禾 - Similarity: 50.00% 5. 株 - Similarity: 50.00% 6. 閑 - Similarity: 50.00% 7. 机 - Similarity: 50.00% 8. 栗 - Similarity: 50.00% 9. 桑 - Similarity: 50.00% 10. 桂 - Similarity: 50.00%	The experimental results for the similarity analysis are summarized in Table III. The output identifies various Kanji that share the radical 木 (tree), with their respective similarity percentages.
2	1. 森 - Similarity: 100.00% 2. 木 - Similarity: 100.00% 3. 杰 - Similarity: 100.00% 4. 禾 - Similarity: 50.00% 5. 株 - Similarity: 50.00% 6. 閑 - Similarity: 50.00% 7. 机 - Similarity: 50.00% 8. 栗 - Similarity: 50.00% 9. 桑 - Similarity: 50.00% 10. 桂 - Similarity: 50.00%	The experimental results for the similarity analysis are summarized in Table III. The output identifies various Kanji that share the radical 木 (tree), with their respective similarity percentages.
3	1. 脂 - Similarity: 66.67% 2. 胆 - Similarity: 66.67% 3. 朝 - Similarity: 66.67% 4. 盟 - Similarity: 66.67% 5. 腥 - Similarity: 66.67% 6. 膃 - Similarity: 66.67% 7. 胃 - Similarity: 50.00% 8. 曳 - Similarity: 50.00% 9. 旺 - Similarity: 50.00% 10. 旧 - Similarity: 50.00%	The experimental results for the similarity analysis of Kanji 明 are summarized here. The algorithm identified various Kanji that share structural components or common radical characteristics, providing similarity scores based on their tree-based structural composition.
4	1. 木 - Similarity: 100.00% 2. 林 - Similarity: 100.00% 3. 杰 - Similarity: 100.00% 4. 禾 - Similarity: 50.00% 5. 株 - Similarity: 50.00% 6. 閑 - Similarity: 50.00% 7. 机 - Similarity: 50.00% 8. 栗 - Similarity: 50.00% 9. 桑 - Similarity: 50.00% 10. 桂 - Similarity: 50.00%	The experimental results for the similarity analysis of Kanji 森 are summarized here. The algorithm successfully identified characters that share the 木 radical component, evaluating the structural similarity between the forest-related Kanji and other characters within the dataset.
5	1. 枷 - Similarity: 100.00% 2. 杏 - Similarity: 80.00% 3. 加 - Similarity: 80.00% 4. 嘉 - Similarity: 80.00% 5. 梧 - Similarity: 80.00% 6. 呆 - Similarity: 80.00% 7. 劭 - Similarity: 80.00% 8. 劼 - Similarity: 80.00% 9. 枸 - Similarity: 80.00% 10. 梱 - Similarity: 80.00%	The experimental results for the similarity analysis of Kanji 架 are summarized here. The algorithm identified characters that share structural components, providing similarity scores based on their tree-based structural composition using Tree Edit Distance.

Based on the test cases presented in Table I, it is evident that for each target Kanji, the algorithm successfully recommends characters that share similar radical compositions. By utilizing the Tree Edit Distance (TED) algorithm, we have quantified the structural resemblance between Kanji, allowing the system to identify and categorize characters with corresponding tree structures.

The results demonstrate that characters with identical or highly overlapping radical hierarchies yield higher similarity scores, confirming that the proposed approach is effective in mapping the structural topology of Japanese Kanji.

TABLE V. RESULT TABLE 2 TEST CASE

NO	Result	Explanation
1	========== ANALISIS KEMIRIPAN TREE EDIT DISTANCE: Kanji 1 [木] Kanji 2 [木] ---------- Jarak Edit (Edit Distance) : 0 Tingkat Kemiripan : 100.00% ==========	The trees are isomorphic. Since both nodes are identical, the edit cost is 0, confirming a 100% similarity score.
2	========== ANALISIS KEMIRIPAN TREE EDIT DISTANCE: Kanji 1 [林] Kanji 2 [木] ---------- Jarak Edit (Edit Distance) : 0 Tingkat Kemiripan : 100.00% ==========	The algorithm recognizes '木' as the core constituent, resulting in a 0 edit cost.
3		'日' is identified as the shared

		root node; the Edit Distance of 1 accounts for the additional '月' radical.
4		No common radical nodes identified. The edit distance of 1 indicates no structural overlap despite the character count.
5		Both share the '加' component; the edit distance of 1 represents the addition of the '木' radical.

Based on the test results, the Tree Edit Distance (TED) algorithm successfully identifies Kanji with similar radical compositions. Kanji sharing common radicals demonstrate higher similarity scores. Furthermore, the algorithm effectively handles differences in structural complexity; any variation in radical placement or addition is captured by the Edit Distance, providing a more precise and accurate evaluation of Kanji similarity compared to conventional character-matching methods.

V. Conclusion

The implementation of the Tree Edit Distance (TED) algorithm, optimized via Dynamic Programming (DP), has proven effective in identifying Kanji with similar radical structures. This capability enhances the understandability of Kanji by allowing them to be grouped based on their radical composition. Furthermore, this similarity-based approach provides a robust framework for generating intelligent recommendations within Kanji search systems.

However, it is important to note that Kanji are defined not only by their radicals but also by their stroke composition. Therefore, future iterations of this algorithm could be improved by integrating a stroke-based search mechanism. By incorporating stroke-related data into the model, the system could perform more nuanced calculations, allowing it to determine and suggest characters that are the most visually similar, even when radical structures differ or stroke counts vary.

VI. Appendix

To complement this research paper, several external resources are provided for readers to explore the implementation and processes involved in this study:

1. Source Code Repository All source code used in this study is publicly available on the following GitHub repository: https://github.com/Jerannn24/Radical-Based-Kanji-Similarity

2. Research Presentation Video A brief explanation of the background, methodology, and results of this research is also presented in a video format, which can be accessed via the following link: https://youtu.be/S-qA4W77-kc

Acknowledgment *(Heading 5)*

First of all, I would like to thank God Almighty for His guidance and blessings, which allowed me to complete this paper even under a tight schedule.

I would also like to express my sincere gratitude to Mr. Rinaldi Munir, my lecturer, for his guidance, patience, and support throughout the past four semesters.

My heartfelt thanks go to my friends and family, who have always encouraged and supported me through both difficult and easy times.

References

[1] humas, "Perbedaan Kanji, Hiragana, dan Katakana dalam Bahasa Jepang - Universitas Al Azhar Indonesia," *Universitas Al Azhar Indonesia*, May 14, 2025. Available: https://uai.ac.id/perbedaan-kanji-hiragana-katakana-bahasa-jepang/. [Accessed: Jun. 16, 2026]

[2] GeeksforGeeks, "Dynamic Programming or DP," *GeeksforGeeks*, Feb. 09, 2024. Available: https://www.geeksforgeeks.org/dsa/dynamic-programming/

[3] GeeksforGeeks, "Edit Distance," *GeeksforGeeks*, Jul. 06, 2011. Available: https://www.geeksforgeeks.org/dsa/edit-distance-dp-5/

[4] M. Simic, "Baeldung," *Baeldung on Computer Science*, Jan. 11, 2022. Available: https://www.baeldung.com/cs/tree-edit-distance. [Accessed: Jun. 16, 2026]

[5] R. Munir, "Strategi Algoritma ," *Itb.ac.id*, 2025. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Juni 2026

Junior Natra Situmorang dan 13524055