Algorithm for Extracting a Graph from Wikipedia Links

1 Abstract

This document describes an algorithm for extracting a graph from Wikipedia links, starting from a root article. The algorithm uses a breadth-first search approach to crawl the Wikipedia links and build the graph, where nodes represent Wikipedia pages and edges represent links between them. The algorithm also includes error handling for cases such as non-existent pages and disambiguation errors.

2 Introduction

The extraction of knowledge graphs from Wikipedia is a crucial task in various domains, including natural language processing, information retrieval, and data mining. This document details a specific algorithm designed for this purpose. It leverages the `wikipedia` and `networkx` Python libraries to retrieve page content and manage the graph structure, respectively. The algorithm performs a breadth-first search, starting from a specified root article, to explore linked Wikipedia pages and construct a graph representation of their relationships.

3 Algorithm Description

The algorithm, named `extract_graph`, takes the title of a root Wikipedia article and an optional maximum number of nodes as input. It returns a NetworkX graph object representing the extracted knowledge graph. The algorithm utilizes a helper function `__crawl` which implements the breadth-first search.

3.1 Pseudo-code

```
procedure EXTRACT_GRAPH(root_article_title, max_nodes)
    __check_title(root_article_title)                    ▷ Check if root article exists
    graph ← nx.Graph()
    graph.add_node(root_article_title)
    visited ← {root_article_title}
    queue ← [root_article_title]
    __crawl(graph, queue, visited, max_nodes)
    if not nx.is_connected(graph) then
        raise Exception("Graph is not connected")
    end if
    return graph
end procedure
procedure __CRAWL(graph, queue, visited, max_nodes)
    if not queue then
        return
    end if
    if len(graph.nodes) ≥ max_nodes then
        return
    end if
    page_title ← queue.pop(0)
    Try
    page ← __check_title(page_title)
    Catch(Exception)
    return                                               ▷ Skip this branch
```

```
    EndTry
    for link_title in page.links do
        if len(graph.nodes) ≥ max_nodes then
            return
        end if
        if link_title in visited then
            __add_edge(graph, page_title, link_title)
        else
            visited.add(link_title)
            queue.append(link_title)
            graph.add_node(link_title)
            __add_edge(graph, page_title, link_title)
        end if
    end for
    __crawl(graph, queue, visited, max_nodes)
end procedure
procedure __ADD_EDGE(graph, page_src, page_dst)
    if page_src == page_dst then
        return                                              ▷ Avoid self-links
    end if
    if graph.has_edge(page_src, page_dst) or graph.has_edge(page_dst, page_src) then
        return                                              ▷ Edge already present
    end if
    graph.add_edge(page_src, page_dst)
end procedure
procedure __CHECK_TITLE(page_title)
    Try
    page ← wikipedia.page(page_title)
    Catch(wikipedia.exceptions.PageError)
    raise Exception("Page not found")
    Catch(wikipedia.exceptions.DisambiguationError)
    raise Exception("Disambiguation Error")
    EndTry
    return page
end procedure
```

3.2 Implementation Details

The implementation uses the `wikipedia` library for fetching page content and the `networkx` library for graph manipulation. The `__check_title` function handles potential exceptions during page retrieval. The `__add_edge` function ensures no duplicate or self-referential edges are added. The `json_dump` function provides a way to serialize the graph into JSON format. The example usage demonstrates how to extract and visualize the graph using `matplotlib`.

4 Conclusion

This document presented a detailed algorithm for extracting knowledge graphs from Wikipedia. The algorithm uses a breadth-first search strategy and incorporates error handling to ensure robustness. The generated graph can be further analyzed and visualized for various applications.