Dijkstra's Algorithm Implementation in C

September 23, 2024

1 Introduction

This document provides the implementation of Dijkstra's algorithm in C, including a detailed explanation of each part of the code. The program calculates the shortest path from a source node to all other nodes in a weighted, directed graph.

2 Problem Breakdown

The program takes an adjacency list representation of a graph as input and outputs the shortest path for each node from the source. The result is sorted by distance in increasing order.

2.1 Input

- The first line contains an integer `n`, the number of nodes in the graph.
- The next `n` lines represent the adjacency list for each node. Each line contains pairs of integers indicating child nodes and weights, followed by `-1`.
- The last line contains the source node.

2.2 Output

The output consists of `n` pairs of integers: each pair shows a node and its shortest path distance from the source. The output is sorted by distance in increasing order.

3 Code Implementation

The following C code implements Dijkstra's algorithm and sorts the nodes by distance:

```c
#include <stdio.h>
#include <limits.h>

#define MAX_NODES 100  // Maximum number of nodes in the graph

int minDistance(int dist[], int processed[], int n) {
    int min = INT_MAX, min_index = -1;
    for (int i = 0; i < n; i++) {
        if (!processed[i] && dist[i] < min) {
            min = dist[i];
            min_index = i;
        }
    }
    return min_index;
}

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void sortByDistance(int nodes[], int dist[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (dist[j] > dist[j + 1]) {
                swap(&dist[j], &dist[j + 1]);
                swap(&nodes[j], &nodes[j + 1]);
            }
        }
    }
}

void dijkstra(int graph[MAX_NODES][MAX_NODES], int n, int src) {
    int dist[MAX_NODES];        // dist[i] will hold the shortest distance from src to i
    int processed[MAX_NODES]; // processed[i] will be 1 if node i's shortest path has be
    int nodes[MAX_NODES];       // store node numbers for output sorting

    for (int i = 0; i < n; i++) {
        dist[i] = INT_MAX;
        processed[i] = 0;
        nodes[i] = i;
    }
    dist[src] = 0;

    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, processed, n);
        if (u == -1) break;
        processed[u] = 1;
        for (int v = 0; v < n; v++) {
            if (!processed[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u]
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    sortByDistance(nodes, dist, n);

    for (int i = 0; i < n; i++) {
        printf("%d-%d-", nodes[i], dist[i]);
    }
}

int main() {
    int n;
    scanf("%d", &n);
    int graph[MAX_NODES][MAX_NODES];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            graph[i][j] = 0;
        }
    }
    for (int i = 0; i < n; i++) {
        int j, weight;
        while (scanf("%d", &j) && j != -1) {
            scanf("%d", &weight);
            graph[i][j] = weight;
        }
    }
    int src;
    scanf(”%d”, &src);
    dijkstra(graph, n, src);
    return 0;
}
```