Dijkstra's Algorithm Cheatsheet

Dijkstra's algorithm is a shortest-path algorithm for graphs. It finds the shortest path from a source vertex to all other vertices in a weighted graph. Here is an overview of the algorithm and its basic syntax.

Algorithm

1. Initialize a set of visited vertices `visited` and a set of tentative distances `dist` to all vertices to infinity, except for the source vertex, which has distance 0.
2. While there are unvisited vertices:
    1. Choose the unvisited vertex with the smallest tentative distance, call it `current`.
    2. For each neighbor `v` of `current` that is still unvisited:
        1. Calculate the tentative distance from the source vertex to `v` via `current`: `dist[current] + weight(current, v)`.
        2. If this tentative distance is less than the current distance stored in `dist[v]`, update `dist[v]` to the new, lower value.
    3. Mark `current` as visited.
3. Return `dist`.

Syntax

Python

```python
import heapq

def dijkstra(graph, source):
    visited = set()
    dist = {v: float('inf') for v in graph}
    dist[source] = 0
    heap = [(0, source)]

    while heap:
        (d, current) = heapq.heappop(heap)
        if current in visited:
            continue
        visited.add(current)
        for v, w in graph[current].items():
            if v in visited:
                continue
            if dist[current] + w < dist[v]:
                dist[v] = dist[current] + w
                heapq.heappush(heap, (dist[v], v))

    return dist
```

C++

```cpp
#include <queue>
#include <unordered_map>
#include <vector>

using namespace std;

typedef unordered_map<int, unordered_map<int, int>> Graph;

vector<int> dijkstra(const Graph& graph, int source) {
    vector<int> dist(graph.size(), INT_MAX);
    dist[source] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
pq;
    pq.push({0, source});

    while (!pq.empty()) {
        int current = pq.top().second;
        pq.pop();
        for (auto neighbor : graph.at(current)) {
            int v = neighbor.first;
            int w = neighbor.second;
            if (dist[current] + w < dist[v]) {
                dist[v] = dist[current] + w;
                pq.push({dist[v], v});
            }
        }
    }

    return dist;
}
```

Resources

- Dijkstra's Algorithm on Wikipedia
- Dijkstra's Algorithm Visualization