/tikz/,/tikz/graphs/
conversions/canvas coordinate/.code=1 , conversions/coordinate/.code=1

trees,layered

THUẬT TOÁN DIJKSTRA (DIJKSTRA'S ALGORITHM)

Toán Tổ Hợp và Lý Thuyết Đồ Thị

1 Lý thuyết cơ bản về thuật toán Dijkstra

1.1 Định nghĩa và đặc điểm

Thuật toán Dijkstra là một thuật toán tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh khác trong đồ thị có trọng số không âm. Thuật toán có những đặc điểm sau:

- Sử dụng chiến lược **tham lam (Greedy)** - chọn đỉnh có khoảng cách nhỏ nhất
- Áp dụng nguyên lý **tối ưu con (Optimal Substructure)**
- Sử dụng cấu trúc dữ liệu **hàng đợi ưu tiên (Priority Queue)** hoặc **heap**
- Chỉ hoạt động đúng với trọng số không âm
- Độ phức tạp: $O((V + E) \log V)$ với heap, $O(V^2)$ với mảng

1.2 So sánh Dijkstra với các thuật toán khác

Tiêu chí	**Dijkstra**	**Bellman-Ford**	**Floyd-Warshall**	**BFS**
Đồ thị có trọng số	Có (0)	Có	Có	Không
Trọng số âm	Không	Có	Có	N/A
Từ một nguồn	Có	Có	Không	Có
Tất cả cặp đỉnh	Không	Không	Có	Không
Độ phức tạp	$O(V^2)$	$O(VE)$	$O(V^3)$	$O(V + E)$
Phát hiện chu trình âm	Không	Có	Có	N/A

1.3 Ứng dụng chính của Dijkstra

- Tìm đường đi ngắn nhất trong hệ thống GPS
- Định tuyến trong mạng máy tính (OSPF protocol)
- Tối ưu hóa trong game AI (pathfinding)
- Phân tích mạng xã hội

- Scheduling và resource allocation
- Network flow optimization

1.4 Nguyên lý hoạt động

Ý tưởng chính:

1. Khởi tạo khoảng cách từ nguồn đến tất cả đỉnh là vô cùng, trừ đỉnh nguồn = 0
2. Duy trì tập đỉnh chưa được xử lý (unvisited)
3. Lặp lại: chọn đỉnh chưa xử lý có khoảng cách nhỏ nhất
4. Cập nhật khoảng cách đến các đỉnh kề nếu tìm được đường ngắn hơn
5. Đánh dấu đỉnh đã xử lý và tiếp tục

2 Bài toán 14: Dijkstra trên đồ thị đơn

2.1 Mô tả bài toán

Đề bài: Cho đồ thị đơn có trọng số $G = (V, E, w)$ với n đỉnh và m cạnh. Mỗi cạnh (u, v) có trọng số $w(u, v) \geq 0$. Cài đặt thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh nguồn s đến tất cả các đỉnh khác.

Input:

- Đồ thị đơn G có trọng số không âm
- Đỉnh nguồn $s \in V$

Output:

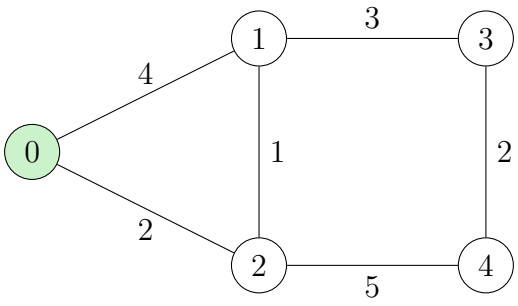
- Khoảng cách ngắn nhất từ s đến mọi đỉnh
- Đường đi ngắn nhất từ s đến mọi đỉnh
- Cây đường đi ngắn nhất (Shortest Path Tree)

2.2 Ý tưởng và giải pháp

Ý tưởng chính:

1. Sử dụng mảng $dist[]$ để lưu khoảng cách ngắn nhất hiện tại
2. Sử dụng mảng $parent[]$ để lưu đỉnh cha trong đường đi ngắn nhất
3. Sử dụng priority queue để luôn chọn đỉnh có khoảng cách nhỏ nhất
4. Áp dụng **edge relaxation**: nếu $dist[u] + w(u, v) < dist[v]$ thì cập nhật $dist[v]$

2.3 Minh họa thuật toán



Đồ thị có trọng số - Dijkstra từ đỉnh 0

Quá trình Dijkstra từ đỉnh 0:

- **Khởi tạo:** $dist = [0, \infty, \infty, \infty, \infty]$, $Q = \{0, 1, 2, 3, 4\}$
- **Bước 1:** Chọn đỉnh 0 ($dist[0] = 0$), cập nhật: $dist[1] = 4$, $dist[2] = 2$
- **Bước 2:** Chọn đỉnh 2 ($dist[2] = 2$), cập nhật: $dist[1] = 3$, $dist[4] = 7$
- **Bước 3:** Chọn đỉnh 1 ($dist[1] = 3$), cập nhật: $dist[3] = 6$
- **Bước 4:** Chọn đỉnh 3 ($dist[3] = 6$), cập nhật: $dist[4] = 8$ (không cải thiện)
- **Kết quả:** $dist = [0, 3, 2, 6, 7]$

2.4 Thuật toán chi tiết

Algorithm 1 Dijkstra's Algorithm

Require: Graph $G = (V, E, w)$, source vertex s
Ensure: Shortest distances and paths from s

Initialize $dist[v] = \infty$ for all $v \in V$
Initialize $parent[v] = -1$ for all $v \in V$
$dist[s] = 0$
Initialize priority queue Q with all vertices
Initialize $visited[v] = false$ for all $v \in V$
while Q is not empty **do**
  $u \leftarrow$ vertex in Q with minimum $dist[u]$
  Remove u from Q
  $visited[u] = true$
  for each vertex v adjacent to u **do**
    if $visited[v] = false$ AND $dist[u] + w(u, v) < dist[v]$ **then**
      $dist[v] = dist[u] + w(u, v)$
      $parent[v] = u$
      Update v in priority queue Q
    end if
  end for
end while
return $dist[]$, $parent[]$

3 Cài đặt cho đồ thị đơn

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <algorithm>
#include <iomanip>

using namespace std;

struct Edge {
    int to;
    int weight;

    Edge(int t, int w) : to(t), weight(w) {}
};

class SimpleDijkstra {
private:
    int numVertices;
    vector<vector<Edge>> adjList;
    vector<int> dist;
    vector<int> parent;
```

```
    vector<bool> visited;

public:
    SimpleDijkstra(int n) : numVertices(n) {
        adjList.resize(n);
        reset();
    }

    void reset() {
        dist.assign(numVertices, INT_MAX);
        parent.assign(numVertices, -1);
        visited.assign(numVertices, false);
    }

    void addEdge(int u, int v, int weight) {
        adjList[u].push_back(Edge(v, weight));
        adjList[v].push_back(Edge(u, weight)); //        t h   v
    h    ng
        cout << "Th m   c nh   " << u << " - " << v << "   v i   t r ng
    s    " << weight << endl;
    }

    // Dijkstra   s     d ng   priority queue
    void dijkstraWithPQ(int source) {
        reset();

        // Priority queue (min-heap): pair<distance, vertex>
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<
   pair<int, int>>> pq;

        dist[source] = 0;
        pq.push({0, source});

        cout << "\nDijkstra   v i   Priority Queue   t       nh     " <<
   source << ":" << endl;

        while (!pq.empty()) {
            int u = pq.top().second;
            int d = pq.top().first;
            pq.pop();

            if (visited[u]) continue;

            visited[u] = true;
            cout << " X    l      nh     " << u << "   v i   k h o ng   c  ch  "
    << d << endl;

            //  D u y t   c  c     nh      k
            for (const Edge& edge : adjList[u]) {
                int v = edge.to;
                int weight = edge.weight;

                if (!visited[v] && dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                    parent[v] = u;
                    pq.push({dist[v], v});
```

```
74
75                      cout << "   C p   n h t  dist[" << v << "] = " <<
    dist[v]
76                           << " (qua   nh   " << u << ")" << endl;
77                  }
78              }
79          }
80      }
81
82      // Dijkstra  c   b n   v i   m ng
83      void dijkstraBasic(int source) {
84          reset();
85
86          dist[source] = 0;
87
88          cout << "\nDijkstra  c   b n   t     nh   " << source << ":" <<
    endl;
89
90          for (int count = 0; count < numVertices - 1; count++) {
91              // T m   nh   ch a  x   l   c   k h o ng  c ch  n h
    n h t
92              int u = findMinDistance();
93              if (u == -1) break;
94
95              visited[u] = true;
96              cout << " B   c  " << count + 1 << ":  C h n    nh   " << u
97                   << "  v i   k h o ng  c ch " << dist[u] << endl;
98
99              //  C p   n h t  k h o ng  c ch   n   c  c   nh    k
100             for (const Edge& edge : adjList[u]) {
101                 int v = edge.to;
102                 int weight = edge.weight;
103
104                 if (!visited[v] && dist[u] != INT_MAX &&
105                     dist[u] + weight < dist[v]) {
106                     dist[v] = dist[u] + weight;
107                     parent[v] = u;
108
109                     cout << "   C p   n h t  dist[" << v << "] = " <<
    dist[v] << endl;
110                 }
111             }
112         }
113     }
114
115 private:
116     int findMinDistance() {
117         int minDist = INT_MAX;
118         int minVertex = -1;
119
120         for (int v = 0; v < numVertices; v++) {
121             if (!visited[v] && dist[v] < minDist) {
122                 minDist = dist[v];
123                 minVertex = v;
124             }
125         }
```

```
126
127         return minVertex;
128     }
129
130 public:
131     void printResults(int source) {
132         cout << "\ n K t   q u   Dijkstra   t      nh    " << source << ":"
    << endl;
133         cout << setw(6) << "   nh   " << setw(12) << " K h o ng   c   ch"
134              << setw(15) << "    ng      i " << endl;
135         cout << string(35, '-') << endl;
136
137         for (int i = 0; i < numVertices; i++) {
138             cout << setw(6) << i;
139
140             if (dist[i] == INT_MAX) {
141                 cout << setw(12) << "    " << setw(15) << "Kh ng  c   " <<
    endl;
142             } else {
143                 cout << setw(12) << dist[i] << setw(15);
144                 printPath(source, i);
145                 cout << endl;
146             }
147         }
148         cout << endl;
149     }
150
151     void printPath(int source, int dest) {
152         if (dest == source) {
153             cout << dest;
154             return;
155         }
156
157         if (parent[dest] == -1) {
158             cout << "Kh ng  c       ng    ";
159             return;
160         }
161
162         vector<int> path;
163         int current = dest;
164
165         while (current != -1) {
166             path.push_back(current);
167             current = parent[current];
168         }
169
170         reverse(path.begin(), path.end());
171
172         for (int i = 0; i < path.size(); i++) {
173             cout << path[i];
174             if (i < path.size() - 1) cout << "    ";
175         }
176     }
177
178     void printGraph() {
179         cout << "       t h     n    c    t r ng    s   :" << endl;
```

```
180         for (int i = 0; i < numVertices; i++) {
181             cout << "   nh    " << i << ": ";
182             for (const Edge& edge : adjList[i]) {
183                 cout << "(" << edge.to << "," << edge.weight << ") ";
184             }
185             cout << endl;
186         }
187         cout << endl;
188     }
189 
190     // T m     ng      i   n g n   n h t   g i a  hai    nh     c    t h
191     pair<int, vector<int>> shortestPath(int source, int dest) {
192         dijkstraWithPQ(source);
193 
194         if (dist[dest] == INT_MAX) {
195             return {-1, {}};
196         }
197 
198         vector<int> path;
199         int current = dest;
200 
201         while (current != -1) {
202             path.push_back(current);
203             current = parent[current];
204         }
205 
206         reverse(path.begin(), path.end());
207         return {dist[dest], path};
208     }
209 
210     // T m k     ng      i   n g n   n h t
211     vector<pair<int, vector<int>>> kShortestPaths(int source, int dest,
    int k) {
212         vector<pair<int, vector<int>>> result;
213 
214         // Dijkstra  c    b n
215         auto shortestPath = this->shortestPath(source, dest);
216         if (shortestPath.first == -1) return result;
217 
218         result.push_back(shortestPath);
219 
220         // T h u t  t o  n  Yen       t  m  k-1    ng     i   t i p  theo (
      n    g i n  h  a)
221         cout << "T  m  " << k << "    ng     i   n g n   n h t   t   " <<
    source
222              << "   n    " << dest << ":" << endl;
223 
224         for (int i = 0; i < result.size() && i < k; i++) {
225             cout << "   ng    " << i + 1 << " (      d  i  " << result[i
    ].first << "): ";
226             for (int j = 0; j < result[i].second.size(); j++) {
227                 cout << result[i].second[j];
228                 if (j < result[i].second.size() - 1) cout << "   ";
229             }
230             cout << endl;
231         }
```

```
232
233         return result;
234     }
235
236     // K i m  tra t nh  c h t  c a  t h
237     void analyzeGraph() {
238         cout << "Ph n t ch  t h  :" << endl;
239
240         // K i m  tra t nh li n th ng
241         bool isConnected = true;
242         dijkstraWithPQ(0);
243
244         for (int i = 0; i < numVertices; i++) {
245             if (dist[i] == INT_MAX) {
246                 isConnected = false;
247                 break;
248             }
249         }
250
251         cout << "  t h  li n th ng: " << (isConnected ? "C  " : "
    Kh ng") << endl;
252
253         // T m  c p  nh  c  k h o ng  c ch  l n  n h t
254         int maxDist = 0;
255         pair<int, int> farthestPair = {-1, -1};
256
257         for (int i = 0; i < numVertices; i++) {
258             dijkstraWithPQ(i);
259             for (int j = 0; j < numVertices; j++) {
260                 if (dist[j] != INT_MAX && dist[j] > maxDist) {
261                     maxDist = dist[j];
262                     farthestPair = {i, j};
263                 }
264             }
265         }
266
267         if (farthestPair.first != -1) {
268             cout << " C p  nh  xa  n h t : " << farthestPair.first
269                  << " v  " << farthestPair.second
270                  << " ( k h o ng  c ch: " << maxDist << ")" << endl;
271         }
272
273         cout << endl;
274     }
275 };
276
277 // Demo cho  t h  n
278 void demonstrateSimpleDijkstra() {
279     cout << "=== DEMO DIJKSTRA CHO  T H  N  ===" << endl <<
    endl;
280
281     SimpleDijkstra graph(5);
282
283     // T o  t h  m u
284     graph.addEdge(0, 1, 4);
285     graph.addEdge(0, 2, 2);
```

```
    graph.addEdge(1, 2, 1);
    graph.addEdge(1, 3, 5);
    graph.addEdge(2, 3, 8);
    graph.addEdge(2, 4, 10);
    graph.addEdge(3, 4, 2);

    cout << endl;
    graph.printGraph();

    // Dijkstra  v i  priority queue
    graph.dijkstraWithPQ(0);
    graph.printResults(0);

    // Dijkstra  c    b n
    graph.dijkstraBasic(0);
    graph.printResults(0);

    // T  m     ng      i   c    t h
    auto path = graph.shortestPath(0, 4);
    cout << "    ng      i   n g n   n h t   t   0   n   4:" << endl;
    cout << "       d  i : " << path.first << endl;
    cout << "    ng      i  : ";
    for (int i = 0; i < path.second.size(); i++) {
        cout << path.second[i];
        if (i < path.second.size() - 1) cout << "   ";
    }
    cout << endl << endl;

    // T  m  k     ng      i   n g n   n h t
    graph.kShortestPaths(0, 4, 3);

    // Ph  n  t  ch        t h
    graph.analyzeGraph();
}
```

Listing 1: Dijkstra cho đồ thị đơn - C++

```
import heapq
from typing import List, Tuple, Dict, Optional
import sys

class SimpleDijkstra:
    """Dijkstra cho        t h     n  """

    def __init__(self, num_vertices: int):
        self.num_vertices = num_vertices
        self.adj_list = [[] for _ in range(num_vertices)]
        self.reset()

    def reset(self):
        self.dist = [float('inf')] * self.num_vertices
        self.parent = [-1] * self.num_vertices
        self.visited = [False] * self.num_vertices

    def add_edge(self, u: int, v: int, weight: int):
        """Th  m   c nh   v  o        t h   v    h    ng """
```

```
        self.adj_list[u].append((v, weight))
        self.adj_list[v].append((u, weight))
        print(f"Th m  c nh  {u} - {v}  v i  t r ng  s  {weight}")

    def dijkstra_with_heap(self, source: int):
        """Dijkstra  s  d ng  heap"""
        self.reset()

        # Heap: (distance, vertex)
        heap = [(0, source)]
        self.dist[source] = 0

        print(f"\nDijkstra  v i  heap  t  nh  {source}:")

        while heap:
            current_dist, u = heapq.heappop(heap)

            if self.visited[u]:
                continue

            self.visited[u] = True
            print(f" X  l  nh  {u}  v i  k h o ng  c ch {
    current_dist}")

            #  D u y t  c c  nh  k
            for v, weight in self.adj_list[u]:
                if not self.visited[v]:
                    new_dist = self.dist[u] + weight

                    if new_dist < self.dist[v]:
                        self.dist[v] = new_dist
                        self.parent[v] = u
                        heapq.heappush(heap, (new_dist, v))

                        print(f"  C p  n h t  dist[{v}] = {new_dist} (
    qua  nh  {u})")

    def dijkstra_basic(self, source: int):
        """Dijkstra c  b n  v i  m ng """
        self.reset()

        self.dist[source] = 0

        print(f"\nDijkstra c  b n  t  nh  {source}:")

        for count in range(self.num_vertices):
            # T m  nh  ch a  x  l  c  k h o ng  c ch  n h
    n h t
            u = self._find_min_distance()
            if u == -1:
                break

            self.visited[u] = True
            print(f" B  c {count + 1}:  C h n  nh  {u}  v i
    k h o ng  c ch {self.dist[u]}")

```

```
            # C p  n h t  k h o ng  c ch  n  c c  nh  k
            for v, weight in self.adj_list[u]:
                if (not self.visited[v] and
                    self.dist[u] != float('inf') and
                    self.dist[u] + weight < self.dist[v]):

                    self.dist[v] = self.dist[u] + weight
                    self.parent[v] = u

                    print(f"  C p  n h t  dist[{v}] = {self.dist[v]}")

    def _find_min_distance(self) -> int:
        """T m  nh  ch a  x  l  c  k h o ng  c ch  n h  n h t """
        min_dist = float('inf')
        min_vertex = -1

        for v in range(self.num_vertices):
            if not self.visited[v] and self.dist[v] < min_dist:
                min_dist = self.dist[v]
                min_vertex = v

        return min_vertex

    def print_results(self, source: int):
        """In  k t  q u  Dijkstra"""
        print(f"\ n K t  q u  Dijkstra  t  nh  {source}:")
        print(f"{'  nh  ':>6} {' K h o ng  c  ch ':>12} {'  ng  i ':>15}")
        print("-" * 35)

        for i in range(self.num_vertices):
            print(f"{i:>6}", end="")

            if self.dist[i] == float('inf'):
                print(f"{'  ':>12} {'Kh ng  c  ':>15}")
            else:
                print(f"{self.dist[i]:>12}", end="")
                path_str =self._get_path_string(source, i)
                print(f"{path_str:>15}")
        print()

    def _get_path_string(self, source: int, dest: int) -> str:
        """ L y  c h u i  b i u  d i n  ng  i """
        if dest == source:
            return str(dest)

        if self.parent[dest] == -1:
            return "Kh ng  c  ng  "

        path = []
        current = dest

        while current != -1:
            path.append(current)
            current = self.parent[current]
```

```
126
127         path.reverse()
128         return "   ".join(map(str, path))
129
130     def print_graph(self):
131         """In        t h  """
132         print("       t h     n    c    t r ng   s  :")
133         for i in range(self.num_vertices):
134             print(f"   nh    {i}: ", end="")
135             for v, weight in self.adj_list[i]:
136                 print(f"({v},{weight}) ", end="")
137             print()
138         print()
139
140     def shortest_path(self, source: int, dest: int) -> Tuple[int, List[
    int]]:
141         """T  m     ng      i   n g n   n h t   g i a  hai    nh   """
142         self.dijkstra_with_heap(source)
143
144         if self.dist[dest] == float('inf'):
145             return (-1, [])
146
147         path = []
148         current = dest
149
150         while current != -1:
151             path.append(current)
152             current = self.parent[current]
153
154         path.reverse()
155         return (self.dist[dest], path)
156
157     def analyze_graph(self):
158         """Ph  n  t  ch        t h  """
159         print("Ph  n  t  ch        t h  :")
160
161         #  K i m  tra  t  nh  li  n  th ng
162         self.dijkstra_with_heap(0)
163         is_connected = all(dist != float('inf') for dist in self.dist)
164
165         print(f"       t h   li  n  th ng: {'C  ' if is_connected else '
    Kh  ng'}")
166
167         # T  m   c p     nh    c   k h o ng  c  ch   l  n   n h t
168         max_dist = 0
169         farthest_pair = (-1, -1)
170
171         for i in range(self.num_vertices):
172             self.dijkstra_with_heap(i)
173             for j in range(self.num_vertices):
174                 if self.dist[j] != float('inf') and self.dist[j] >
    max_dist:
175                     max_dist = self.dist[j]
176                     farthest_pair = (i, j)
177
178         if farthest_pair[0] != -1:
```

```
            print(f" C p     nh    xa  n h t : {farthest_pair[0]} v  {farthest_pair[1]} "
                  f"( k h o ng  c  ch: {max_dist})")
        print()

def demonstrate_simple_dijkstra():
    """Demo Dijkstra cho       t h    n  """
    print("=== DEMO DIJKSTRA CHO       T H    N   ===\n")

    graph = SimpleDijkstra(5)

    #  T o        t h    m u
    graph.add_edge(0, 1, 4)
    graph.add_edge(0, 2, 2)
    graph.add_edge(1, 2, 1)
    graph.add_edge(1, 3, 5)
    graph.add_edge(2, 3, 8)
    graph.add_edge(2, 4, 10)
    graph.add_edge(3, 4, 2)

    print()
    graph.print_graph()

    # Dijkstra  v i  heap
    graph.dijkstra_with_heap(0)
    graph.print_results(0)

    # Dijkstra  c    b n
    graph.dijkstra_basic(0)
    graph.print_results(0)

    # T m     ng     i   c    t h
    dist, path = graph.shortest_path(0, 4)
    print("    ng     i   n g n   n h t   t   0   n   4:")
    print(f"      d  i : {dist}")
    print(f"    ng     i : {'   '.join(map(str, path))}\n")

    # Ph n  t  ch       t h
    graph.analyze_graph()

if __name__ == "__main__":
    demonstrate_simple_dijkstra()
```

Listing 2: Dijkstra cho đồ thị đơn - Python

4 Bài toán 15: Dijkstra trên đa đồ thị

4.1 Mô tả bài toán

Đề bài: Cho đa đồ thị (multigraph) có trọng số $G = (V, E, w)$ với n đỉnh và m cạnh. Giữa hai đỉnh có thể có nhiều cạnh với trọng số khác nhau. Cài đặt thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh nguồn s đến tất cả các đỉnh khác.

Đặc điểm của đa đồ thị:

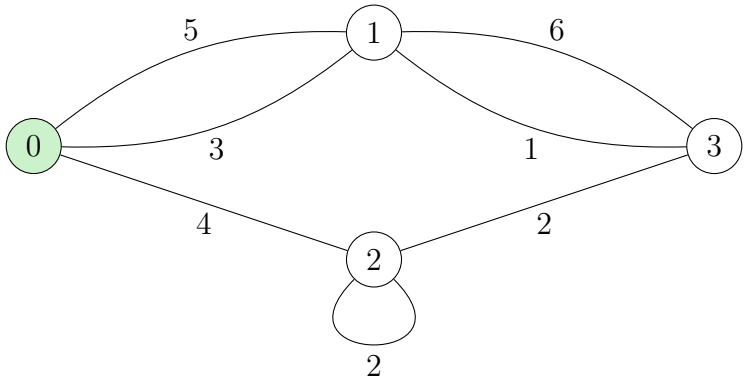
- Cho phép nhiều cạnh giữa hai đỉnh (parallel edges)
- Mỗi cạnh có thể có trọng số khác nhau
- Có thể có vòng lặp (self-loops)
- Cần xử lý đặc biệt để chọn cạnh có trọng số nhỏ nhất

4.2 Điều chỉnh thuật toán cho đa đồ thị

Thay đổi chính:

1. Lưu trữ tất cả các cạnh song song
2. Khi relaxation, xem xét tất cả các cạnh từ u đến v
3. Chọn cạnh có trọng số nhỏ nhất trong quá trình cập nhật
4. Xử lý trường hợp có vòng lặp

4.3 Minh họa đa đồ thị



Đa đồ thị có cạnh song song và vòng lặp

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <algorithm>
#include <map>
#include <set>

using namespace std;

struct MultiEdge {
    int to;
    int weight;
    int id; // ID        ph n  b i t  c  c  c nh  song song

    MultiEdge(int t, int w, int i) : to(t), weight(w), id(i) {}
```

```
};

class MultiGraphDijkstra {
private:
    int numVertices;
    vector<vector<MultiEdge>> adjList;
    vector<int> dist;
    vector<int> parent;
    vector<int> parentEdgeId;
    vector<bool> visited;
    int edgeIdCounter;

public:
    MultiGraphDijkstra(int n) : numVertices(n), edgeIdCounter(0) {
        adjList.resize(n);
        reset();
    }

    void reset() {
        dist.assign(numVertices, INT_MAX);
        parent.assign(numVertices, -1);
        parentEdgeId.assign(numVertices, -1);
        visited.assign(numVertices, false);
    }

    void addEdge(int u, int v, int weight) {
        // Th m   c nh   t   u   n   v
        adjList[u].push_back(MultiEdge(v, weight, edgeIdCounter));

        //  N u  kh ng  p h i  v ng  l p , th m  c nh  n g  c  l i
        if (u != v) {
            adjList[v].push_back(MultiEdge(u, weight, edgeIdCounter));
        }

        cout << "Th m  c nh  " << u << " - " << v << "  v i  t r ng
    s  "
             << weight << " (ID: " << edgeIdCounter << ")" << endl;

        edgeIdCounter++;
    }

    void dijkstraMultiGraph(int source) {
        reset();

        priority_queue<pair<int, int>, vector<pair<int, int>>,
                       greater<pair<int, int>>> pq;

        dist[source] = 0;
        pq.push({0, source});

        cout << "\nDijkstra  tr n  a        t h   t     nh   " <<
   source << ":" << endl;

        while (!pq.empty()) {
            int u = pq.top().second;
            int d = pq.top().first;
```

```
71              pq.pop();
72
73              if (visited[u]) continue;
74
75              visited[u] = true;
76              cout << " X   l     nh    " << u << "  v i   k h o ng   c  ch "
    << d << endl;
77
78              // N h  m   c  c   c nh   theo    nh      ch    v    c h n   c nh
    n h    n h t
79              map<int, MultiEdge> bestEdges;
80
81              for (const MultiEdge& edge : adjList[u]) {
82                  int v = edge.to;
83
84                  if (!visited[v]) {
85                      //  C h n   c nh   c    t r ng   s    n h    n h t
      n       nh    v
86                      if (bestEdges.find(v) == bestEdges.end() ||
87                          edge.weight < bestEdges[v].weight) {
88                          bestEdges[v] = edge;
89                      }
90                  }
91              }
92
93              //  C p   n h t   k h o ng   c  ch   v i   c  c   c nh   t t
    n h t
94              for (const auto& pair : bestEdges) {
95                  int v = pair.first;
96                  const MultiEdge& edge = pair.second;
97
98                  if (dist[u] + edge.weight < dist[v]) {
99                      dist[v] = dist[u] + edge.weight;
100                     parent[v] = u;
101                     parentEdgeId[v] = edge.id;
102                     pq.push({dist[v], v});
103
104                     cout << "   C p   n h t  dist[" << v << "] = " <<
    dist[v]
105                          << " (qua   c nh   ID " << edge.id << ")" << endl
    ;
106                 }
107             }
108         }
109     }
110
111     // Phi n   b n    x    l    t t    c     c nh   song song
112     void dijkstraAllEdges(int source) {
113         reset();
114
115         priority_queue<pair<int, int>, vector<pair<int, int>>,
116                        greater<pair<int, int>>> pq;
117
118         dist[source] = 0;
119         pq.push({0, source});
120
```

```
121         cout << "\nDijkstra  x    l    t t    c     c nh   song song   t
      nh    "
122              << source << ":" << endl;
123
124         while (!pq.empty()) {
125             int u = pq.top().second;
126             int d = pq.top().first;
127             pq.pop();
128
129             if (visited[u]) continue;
130
131             visited[u] = true;
132             cout << " X    l      nh     " << u << "   v i    k h o ng   c  ch "
    << d << endl;
133
134             //  X    l    t t    c    c  c   c nh   (bao   g m    c nh   song
    song)
135             for (const MultiEdge& edge : adjList[u]) {
136                 int v = edge.to;
137                 int weight = edge.weight;
138
139                 if (!visited[v] && dist[u] + weight < dist[v]) {
140                     dist[v] = dist[u] + weight;
141                     parent[v] = u;
142                     parentEdgeId[v] = edge.id;
143                     pq.push({dist[v], v});
144
145                     cout << "    C p    n h t   dist[" << v << "] = " <<
    dist[v]
146                          << " (qua   c nh   ID " << edge.id
147                          << "   v i    t r ng    s     " << weight << ")" <<
    endl;
148                 }
149             }
150         }
151     }
152
153     void printMultiGraph() {
154         cout << " a          t h    c     t r ng    s   :" << endl;
155
156         for (int i = 0; i < numVertices; i++) {
157             cout << "    nh     " << i << ": ";
158
159             // Nh  m   c nh   theo     nh       ch
160             map<int, vector<MultiEdge>> edgeGroups;
161             for (const MultiEdge& edge : adjList[i]) {
162                 edgeGroups[edge.to].push_back(edge);
163             }
164
165             for (const auto& group : edgeGroups) {
166                 int dest = group.first;
167                 const vector<MultiEdge>& edges = group.second;
168
169                 cout << dest << "[";
170                 for (int j = 0; j < edges.size(); j++) {
171                     cout << edges[j].weight;
```

```
172                     if (j < edges.size() - 1) cout << ",";
173                 }
174                 cout << "] ";
175             }
176             cout << endl;
177         }
178         cout << endl;
179     }
180
181     void printResults(int source) {
182         cout << "\ n K t   q u   Dijkstra  t      nh    " << source << ":"
    << endl;
183         cout << setw(6) << "   nh   " << setw(12) << " K h o ng  c  ch"
184              << setw(10) << " C nh  ID" << setw(15) << "    ng      i "
    << endl;
185         cout << string(45, '-') << endl;
186
187         for (int i = 0; i < numVertices; i++) {
188             cout << setw(6) << i;
189
190             if (dist[i] == INT_MAX) {
191                 cout << setw(12) << "    " << setw(10) << "-"
192                      << setw(15) << "Kh ng  c  " << endl;
193             } else {
194                 cout << setw(12) << dist[i]
195                      << setw(10) << parentEdgeId[i] << setw(15);
196                 printPath(source, i);
197                 cout << endl;
198             }
199         }
200         cout << endl;
201     }
202
203     void printPath(int source, int dest) {
204         if (dest == source) {
205             cout << dest;
206             return;
207         }
208
209         if (parent[dest] == -1) {
210             cout << "Kh ng  c       ng    ";
211             return;
212         }
213
214         vector<int> path;
215         vector<int> edgeIds;
216         int current = dest;
217
218         while (current != -1 && current != source) {
219             path.push_back(current);
220             if (parentEdgeId[current] != -1) {
221                 edgeIds.push_back(parentEdgeId[current]);
222             }
223             current = parent[current];
224         }
225
```

```
226         if (current == source) {
227             path.push_back(source);
228         }
229 
230         reverse(path.begin(), path.end());
231         reverse(edgeIds.begin(), edgeIds.end());
232 
233         for (int i = 0; i < path.size(); i++) {
234             cout << path[i];
235             if (i < path.size() - 1) {
236                 cout << "   ";
237             }
238         }
239     }
240 
241     // T m  t t   c   c c    ng     i   v i  chi ph   t i   u  ( x
      l   c nh  song song)
242     void findAllOptimalPaths(int source, int dest) {
243         dijkstraAllEdges(source);
244 
245         if (dist[dest] == INT_MAX) {
246             cout << "Kh ng  c      ng     i   t   " << source << "
       n    " << dest << endl;
247             return;
248         }
249 
250         cout << " T t   c      ng     i   t i   u   t   " << source <<
    "    n    " << dest
251              << " (chi  ph   " << dist[dest] << "):" << endl;
252 
253         vector<vector<int>> allPaths;
254         vector<int> currentPath;
255         findAllPathsHelper(source, dest, dist[dest], 0, currentPath,
    allPaths);
256 
257         for (int i = 0; i < allPaths.size(); i++) {
258             cout << "   ng    " << i + 1 << ": ";
259             for (int j = 0; j < allPaths[i].size(); j++) {
260                 cout << allPaths[i][j];
261                 if (j < allPaths[i].size() - 1) cout << "   ";
262             }
263             cout << endl;
264         }
265     }
266 
267 private:
268     void findAllPathsHelper(int current, int dest, int targetCost,
269                             int currentCost, vector<int>& path,
270                             vector<vector<int>>& allPaths) {
271         path.push_back(current);
272 
273         if (current == dest && currentCost == targetCost) {
274             allPaths.push_back(path);
275             path.pop_back();
276             return;
277         }
```

```cpp

        if (currentCost >= targetCost) {
            path.pop_back();
            return;
        }

        for (const MultiEdge& edge : adjList[current]) {
            int next = edge.to;
            int newCost = currentCost + edge.weight;

            if (newCost <= targetCost) {
                findAllPathsHelper(next, dest, targetCost, newCost, path, allPaths);
            }
        }

        path.pop_back();
    }
};
```

Listing 3: Dijkstra cho đa đồ thị - C++

```python
import heapq
from typing import List, Tuple, Dict, Optional
from collections import defaultdict

class MultiEdge:
    """ L p  b i u  d i n  c nh  trong  a  t h """
    def __init__(self, to: int, weight: int, edge_id: int):
        self.to = to
        self.weight = weight
        self.id = edge_id

class MultiGraphDijkstra:
    """Dijkstra cho  a  t h """

    def __init__(self, num_vertices: int):
        self.num_vertices = num_vertices
        self.adj_list = [[] for _ in range(num_vertices)]
        self.edge_id_counter = 0
        self.reset()

    def reset(self):
        self.dist = [float('inf')] * self.num_vertices
        self.parent = [-1] * self.num_vertices
        self.parent_edge_id = [-1] * self.num_vertices
        self.visited = [False] * self.num_vertices

    def add_edge(self, u: int, v: int, weight: int):
        """Th m  c nh  v o  a  t h """
        # Th m  c nh  t  u  n  v
        self.adj_list[u].append(MultiEdge(v, weight, self.edge_id_counter))

        #  N u  kh ng  p h i  v ng  l p , th m  c nh  n g  c  l i
        if u != v:
```

```
34            self.adj_list[v].append(MultiEdge(u, weight, self.
    edge_id_counter))
35
36        print(f"Th m  c nh  {u} - {v}  v i  t r ng  s  {weight} (ID:
    {self.edge_id_counter})")
37        self.edge_id_counter += 1
38
39    def dijkstra_multigraph(self, source: int):
40        """Dijkstra cho  a  th  - c h n  c nh  t t  n h t """
41        self.reset()
42
43        heap = [(0, source)]
44        self.dist[source] = 0
45
46        print(f"\nDijkstra tr n  a  th  t  nh  {source}:"
    )
47
48        while heap:
49            current_dist, u = heapq.heappop(heap)
50
51            if self.visited[u]:
52                continue
53
54            self.visited[u] = True
55            print(f" X  l  nh  {u}  v i  k h o ng  c ch {
    current_dist}")
56
57            # N h m  c c  c nh  theo  nh  ch  v  c h n  c nh
     t t  n h t
58            best_edges = {}
59
60            for edge in self.adj_list[u]:
61                v = edge.to
62
63                if not self.visited[v]:
64                    # C h n  c nh  c  t r ng  s  n h  n h t  n
        nh  v
65                    if v not in best_edges or edge.weight < best_edges[v
    ].weight:
66                        best_edges[v] = edge
67
68            #  C p  n h t  k h o ng  c ch  v i  c c  c nh  t t  n h t
69            for v, edge in best_edges.items():
70                new_dist = self.dist[u] + edge.weight
71
72                if new_dist < self.dist[v]:
73                    self.dist[v] = new_dist
74                    self.parent[v] = u
75                    self.parent_edge_id[v] = edge.id
76                    heapq.heappush(heap, (new_dist, v))
77
78                    print(f"  C p  n h t  dist[{v}] = {new_dist} "
79                          f"(qua  c nh  ID {edge.id})")
80
81    def dijkstra_all_edges(self, source: int):
82        """Dijkstra  x  l  t t  c  c nh  song song"""
```

```
        self.reset()

        heap = [(0, source)]
        self.dist[source] = 0

        print(f"\nDijkstra  x    l    t t   c     c nh  song song  t
    nh    {source}:")

        while heap:
            current_dist, u = heapq.heappop(heap)

            if self.visited[u]:
                continue

            self.visited[u] = True
            print(f" X    l      nh    {u}  v i   k h o ng  c  ch {
    current_dist}")

            #  X    l    t t   c    c  c   c nh
            for edge in self.adj_list[u]:
                v = edge.to
                weight = edge.weight

                if not self.visited[v]:
                    new_dist = self.dist[u] + weight

                    if new_dist < self.dist[v]:
                        self.dist[v] = new_dist
                        self.parent[v] = u
                        self.parent_edge_id[v] = edge.id
                        heapq.heappush(heap, (new_dist, v))

                        print(f"   C p   n h t  dist[{v}] = {new_dist} "
                              f"(qua   c nh  ID {edge.id}   v i   t r ng
     s   {weight})")

    def print_multigraph(self):
        """In  a         t h  """
        print(" a         t h   c    t r ng   s  :")

        for i in range(self.num_vertices):
            print(f"   nh    {i}: ", end="")

            # N h  m   c nh  theo    nh      ch
            edge_groups = defaultdict(list)
            for edge in self.adj_list[i]:
                edge_groups[edge.to].append(edge)

            for dest, edges in edge_groups.items():
                weights = [str(edge.weight) for edge in edges]
                print(f"{dest}[{','.join(weights)}] ", end="")

            print()
        print()

    def print_results(self, source: int):
```

```
136         """In  k t   q u  """
137         print(f"\ n K t   q u   Dijkstra  t      nh    {source}:")
138         print(f"{’   nh   ’:>6} {’ K h o ng  c  ch ’:>12} {’ C nh  ID’:>10}
    {’    ng      i  ’:>15}")
139         print("-" * 45)
140 
141         for i in range(self.num_vertices):
142             print(f"{i:>6}", end="")
143 
144             if self.dist[i] == float(’inf’):
145                 print(f"{’    ’:>12} {’-’:>10} {’Kh ng  c  ’:>15}")
146             else:
147                 print(f"{self.dist[i]:>12} {self.parent_edge_id[i]:>10}"
    , end="")
148                 path_str = self._get_path_string(source, i)
149                 print(f"{path_str:>15}")
150         print()
151 
152     def _get_path_string(self, source: int, dest: int) -> str:
153         """ L y   c h u i   b i u   d i n      ng      i """
154         if dest == source:
155             return str(dest)
156 
157         if self.parent[dest] == -1:
158             return "Kh ng  c       ng    "
159 
160         path = []
161         current = dest
162 
163         while current != -1 and current != source:
164             path.append(current)
165             current = self.parent[current]
166 
167         if current == source:
168             path.append(source)
169 
170         path.reverse()
171         return "    ".join(map(str, path))
172 
173     def find_all_optimal_paths(self, source: int, dest: int):
174         """T  m   t t   c       ng      i   t i   u """
175         self.dijkstra_all_edges(source)
176 
177         if self.dist[dest] == float(’inf’):
178             print(f"Kh ng  c       ng      i   t   {source}   n   {dest}
    ")
179             return
180 
181         print(f" T t   c       ng      i   t i   u   t   {source}   n
    {dest} "
182               f"(chi  ph   {self.dist[dest]}):")
183 
184         all_paths = []
185         self._find_all_paths_helper(source, dest, self.dist[dest], 0,
    [], all_paths)
186 
```

```
187         for i, path in enumerate(all_paths):
188             print(f"    ng     {i + 1}: {'    '.join(map(str, path))}")
189 
190     def _find_all_paths_helper(self, current: int, dest: int,
    target_cost: int,
191                                current_cost: int, path: List[int],
192                                all_paths: List[List[int]]):
193         """Helper function        t  m   t t   c       ng      i """
194         path.append(current)
195 
196         if current == dest and current_cost == target_cost:
197             all_paths.append(path[:])
198             path.pop()
199             return
200 
201         if current_cost >= target_cost:
202             path.pop()
203             return
204 
205         for edge in self.adj_list[current]:
206             next_vertex = edge.to
207             new_cost = current_cost + edge.weight
208 
209             if new_cost <= target_cost:
210                 self._find_all_paths_helper(next_vertex, dest,
    target_cost,
211                                             new_cost, path, all_paths)
212 
213         path.pop()
214 
215 def demonstrate_multigraph_dijkstra():
216     """Demo Dijkstra cho  a         t h  """
217     print("=== DEMO DIJKSTRA CHO  A         T H   ===\n")
218 
219     graph = MultiGraphDijkstra(4)
220 
221     #  T o   a         t h    v i    c nh  song song
222     graph.add_edge(0, 1, 5)  #  C nh  song song 1
223     graph.add_edge(0, 1, 3)  #  C nh  song song 2
224     graph.add_edge(0, 2, 4)
225     graph.add_edge(2, 3, 2)
226     graph.add_edge(1, 3, 6)  #  C nh  song song 1
227     graph.add_edge(1, 3, 1)  #  C nh  song song 2
228     graph.add_edge(2, 2, 2)  # V ng  l p
229 
230     print()
231     graph.print_multigraph()
232 
233     # Dijkstra  c h n   c nh   t t   n h t
234     graph.dijkstra_multigraph(0)
235     graph.print_results(0)
236 
237     # Dijkstra  x   l    t t   c    c nh
238     graph.dijkstra_all_edges(0)
239     graph.print_results(0)
240 
```

```
    # T m   t t   c       ng      i   t i   u
    graph.find_all_optimal_paths(0, 3)

if __name__ == "__main__":
    demonstrate_multigraph_dijkstra()
```

Listing 4: Dijkstra cho đa đồ thị - Python

4.4 Demo và kết quả

```
void demonstrateMultiGraphDijkstra() {
    cout << "=== DEMO DIJKSTRA CHO  A         T H   ===" << endl << endl;

    MultiGraphDijkstra graph(4);

    //  T o   a         t h    v i   c nh  song song
    graph.addEdge(0, 1, 5);  //  C nh  song song 1
    graph.addEdge(0, 1, 3);  //  C nh  song song 2 ( t t  h  n )
    graph.addEdge(0, 2, 4);
    graph.addEdge(2, 3, 2);
    graph.addEdge(1, 3, 6);  //  C nh  song song 1
    graph.addEdge(1, 3, 1);  //  C nh  song song 2 ( t t  h  n )
    graph.addEdge(2, 2, 2);  // V ng  l p

    cout << endl;
    graph.printMultiGraph();

    // So  s nh hai  ph   ng  ph p
    graph.dijkstraMultiGraph(0);
    graph.printResults(0);

    graph.dijkstraAllEdges(0);
    graph.printResults(0);

    // T m   t t   c       ng      i   t i   u
    graph.findAllOptimalPaths(0, 3);
}

int main() {
    demonstrateMultiGraphDijkstra();
    return 0;
}
```

Listing 5: Demo Dijkstra cho đa đồ thị

Kết quả mong đợi:

- Đường đi ngắn nhất từ 0 đến 3: $0 \to 1 \to 3$ với chi phí 4
- Sử dụng cạnh $(0, 1)$ có trọng số 3 và cạnh $(1, 3)$ có trọng số 1
- Bỏ qua các cạnh có trọng số lớn hơn giữa cùng cặp đỉnh

5 Bài toán 16: Dijkstra trên đồ thị tổng quát

5.1 Mô tả bài toán

Đề bài: Cho đồ thị tổng quát $G = (V, E, w)$ có thể chứa:

- Đồ thị có hướng hoặc vô hướng
- Cạnh song song (parallel edges)
- Vòng lặp (self-loops)
- Trọng số không âm
- Đỉnh có thể không liên thông

Cài đặt thuật toán Dijkstra tổng quát để xử lý mọi trường hợp và cung cấp thông tin chi tiết về đồ thị.

5.2 Phân tích các trường hợp đặc biệt

1. Đồ thị có hướng vs vô hướng:

- Đồ thị có hướng: chỉ duyệt theo chiều cạnh
- Đồ thị vô hướng: duyệt cả hai chiều
- Cần flag để phân biệt loại đồ thị

2. Xử lý vòng lặp:

- Vòng lặp có trọng số dương: có thể bỏ qua
- Vòng lặp trọng số 0: cần xử lý cẩn thận
- Kiểm tra và báo cáo vòng lặp

3. Tính liên thông:

- Đồ thị không liên thông: một số đỉnh không đạt được
- Báo cáo các thành phần liên thông
- Chạy Dijkstra từ nhiều nguồn nếu cần

5.3 Thuật toán tổng quát

Algorithm 2 General Dijkstra's Algorithm

Require: Graph $G = (V, E, w)$, source vertex s, directed flag
Ensure: Shortest distances, paths, and graph analysis

Initialize $dist[v] = \infty$ for all $v \in V$
Initialize $parent[v] = -1$ for all $v \in V$
Initialize $edgeInfo[v] = null$ for all $v \in V$
$dist[s] = 0$
Initialize priority queue Q and visited set
Initialize graph statistics
while Q is not empty **do**
  $u \leftarrow$ vertex in Q with minimum $dist[u]$
  Remove u from Q and mark as visited
  for each edge $e = (u, v)$ from u **do**
    if edge is valid for relaxation **then**
      if $dist[u] + w(e) < dist[v]$ **then**
        $dist[v] = dist[u] + w(e)$
        $parent[v] = u$
        $edgeInfo[v] = e$
        Update v in priority queue Q
        Record edge usage statistics
      end if
    end if
  end for
end while
Analyze graph properties
Generate comprehensive report
return Complete results and analysis

5.4 Cài đặt đồ thị tổng quát

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <unordered_set>
#include <unordered_map>
#include <iomanip>
#include <algorithm>

using namespace std;

struct GeneralEdge {
    int from, to;
    int weight;
    int id;
    string type; // "normal", "self-loop", "parallel"
```

```
17
18      GeneralEdge(int f, int t, int w, int i, string ty = "normal")
19          : from(f), to(t), weight(w), id(i), type(ty) {}
20  };
21
22  struct GraphStatistics {
23      int numVertices;
24      int numEdges;
25      int numSelfLoops;
26      int numParallelEdges;
27      bool isDirected;
28      bool isConnected;
29      vector<vector<int>> components;
30      unordered_map<string, int> edgeTypeCount;
31
32      GraphStatistics() : numVertices(0), numEdges(0), numSelfLoops(0),
33                          numParallelEdges(0), isDirected(false),
    isConnected(false) {}
34  };
35
36  class GeneralDijkstra {
37  private:
38      int numVertices;
39      bool isDirected;
40      vector<vector<GeneralEdge>> adjList;
41      vector<int> dist;
42      vector<int> parent;
43      vector<int> parentEdgeId;
44      vector<bool> visited;
45      vector<bool> inCurrentComponent;
46
47      int edgeIdCounter;
48      GraphStatistics stats;
49      unordered_map<string, int> edgeKeyCount; //    m     c nh   song song
50
51  public:
52      GeneralDijkstra(int n, bool directed = false)
53          : numVertices(n), isDirected(directed), edgeIdCounter(0) {
54          adjList.resize(n);
55          stats.numVertices = n;
56          stats.isDirected = directed;
57          reset();
58      }
59
60      void reset() {
61          dist.assign(numVertices, INT_MAX);
62          parent.assign(numVertices, -1);
63          parentEdgeId.assign(numVertices, -1);
64          visited.assign(numVertices, false);
65          inCurrentComponent.assign(numVertices, false);
66      }
67
68      void addEdge(int u, int v, int weight) {
69          string edgeKey = to_string(min(u, v)) + "-" + to_string(max(u, v
    ));
70          edgeKeyCount[edgeKey]++;
```

```
71
72          string edgeType = "normal";
73          if (u == v) {
74              edgeType = "self-loop";
75              stats.numSelfLoops++;
76          } else if (edgeKeyCount[edgeKey] > 1) {
77              edgeType = "parallel";
78              stats.numParallelEdges++;
79          }
80
81          // Th m   c nh   t   u   n   v
82          adjList[u].push_back(GeneralEdge(u, v, weight, edgeIdCounter,
    edgeType));
83
84          //  N u  l        t h   v   h   ng  v  kh ng  p h i  v ng
     l p
85          if (!isDirected && u != v) {
86              adjList[v].push_back(GeneralEdge(v, u, weight, edgeIdCounter
    , edgeType));
87          }
88
89          stats.numEdges++;
90          stats.edgeTypeCount[edgeType]++;
91
92          cout << "Th m  c nh  " << (isDirected ? "c   h   ng  " : "")
93               << u << " -> " << v << " ( t r ng  s  : " << weight
94               << ",  l o i : " << edgeType << ", ID: " << edgeIdCounter <<
     ")" << endl;
95
96          edgeIdCounter++;
97      }
98
99      void dijkstraGeneral(int source) {
100         reset();
101
102         priority_queue<pair<int, int>, vector<pair<int, int>>,
103                        greater<pair<int, int>>> pq;
104
105         dist[source] = 0;
106         pq.push({0, source});
107
108         cout << "\nDijkstra  t ng  qu t  t     nh   " << source
109              << " tr n       t h  " << (isDirected ? "c   h   ng :" :
    "v   h   ng :") << endl;
110
111         int processedVertices = 0;
112
113         while (!pq.empty()) {
114             int u = pq.top().second;
115             int d = pq.top().first;
116             pq.pop();
117
118             if (visited[u]) continue;
119
120             visited[u] = true;
121             inCurrentComponent[u] = true;
```

```
122             processedVertices++;
123 
124             cout << " B    c   " << processedVertices << ":  X    l
    nh    " << u
125                  << " ( k h o ng  c  ch: " << d << ")" << endl;
126 
127             //  X    l    t t   c    c  c   c nh   t   u
128             for (const GeneralEdge& edge : adjList[u]) {
129                 int v = edge.to;
130                 int weight = edge.weight;
131 
132                 cout << "   K i m  tra  c nh  " << u << " -> " << v
133                      << " ( t r ng   s  : " << weight << ",  l o i : " <<
    edge.type << ")";
134 
135                 //  X    l     c    b i t  cho  v  ng  l p
136                 if (edge.type == "self-loop") {
137                     cout << " [V  ng  l p  -  b   qua]" << endl;
138                     continue;
139                 }
140 
141                 if (!visited[v] && dist[u] + weight < dist[v]) {
142                     dist[v] = dist[u] + weight;
143                     parent[v] = u;
144                     parentEdgeId[v] = edge.id;
145                     pq.push({dist[v], v});
146 
147                     cout << " [ C p   n h t  dist[" << v << "] = " <<
    dist[v] << "]" << endl;
148                 } else {
149                     cout << " [Kh  ng  c p   n h t ]" << endl;
150                 }
151             }
152         }
153 
154         cout << "      x    l   " << processedVertices << "/" <<
    numVertices << "    nh   " << endl;
155     }
156 
157     void analyzeConnectivity() {
158         stats.components.clear();
159         vector<bool> globalVisited(numVertices, false);
160 
161         for (int i = 0; i < numVertices; i++) {
162             if (!globalVisited[i]) {
163                 vector<int> component;
164                 dfsComponent(i, globalVisited, component);
165                 stats.components.push_back(component);
166             }
167         }
168 
169         stats.isConnected = (stats.components.size() == 1);
170 
171         cout << "\nPh n  t  ch  t  nh  li  n  th  ng:" << endl;
172         cout << " S   th nh  p h n  li  n  th  ng: " << stats.components.
    size() << endl;
```

```
173
174         for (int i = 0; i < stats.components.size(); i++) {
175             cout << "Th nh  p h n  " << i + 1 << ": {";
176             for (int j = 0; j < stats.components[i].size(); j++) {
177                 cout << stats.components[i][j];
178                 if (j < stats.components[i].size() - 1) cout << ", ";
179             }
180             cout << "}" << endl;
181         }
182     }
183
184 private:
185     void dfsComponent(int u, vector<bool>& globalVisited, vector<int>&
    component) {
186         globalVisited[u] = true;
187         component.push_back(u);
188
189         for (const GeneralEdge& edge : adjList[u]) {
190             int v = edge.to;
191             if (!globalVisited[v]) {
192                 dfsComponent(v, globalVisited, component);
193             }
194         }
195     }
196
197 public:
198     void printGeneralGraph() {
199         cout << "\ n       t h    t ng   qu t (" << (isDirected ? "c
    h    ng " : "v    h    ng ") << "):" << endl;
200
201         for (int i = 0; i < numVertices; i++) {
202             cout << "   nh    " << i << ": ";
203
204             // Nh m   c nh   theo    ch    v    l o i
205             unordered_map<int, vector<GeneralEdge>> edgeGroups;
206             for (const GeneralEdge& edge : adjList[i]) {
207                 edgeGroups[edge.to].push_back(edge);
208             }
209
210             for (const auto& group : edgeGroups) {
211                 int dest = group.first;
212                 const vector<GeneralEdge>& edges = group.second;
213
214                 cout << dest;
215                 if (edges.size() > 1 || edges[0].type != "normal") {
216                     cout << "[";
217                     for (int j = 0; j < edges.size(); j++) {
218                         cout << edges[j].weight;
219                         if (edges[j].type == "self-loop") cout << "L";
220                         if (edges[j].type == "parallel" && edges.size()
    > 1) cout << "P";
221                         if (j < edges.size() - 1) cout << ",";
222                     }
223                     cout << "]";
224                 } else {
225                     cout << "(" << edges[0].weight << ")";
```

```
226                 }
227                 cout << " ";
228             }
229             cout << endl;
230         }
231         cout << endl;
232     }
233 
234     void printDetailedResults(int source) {
235         cout << "\n===  K T   Q U   CHI  T I T  DIJKSTRA ===" << endl;
236         cout << " N g u n : " << source << " |        t h  : "
237              << (isDirected ? " c    h    ng " : " v    h    ng ") << endl;
238         cout << string(70, '-') << endl;
239 
240         cout << left << setw(6) << "   nh   "
241              << setw(12) << " K h o ng  c  c h"
242              << setw(8) << "Cha"
243              << setw(10) << " C nh  ID"
244              << setw(15) << "    ng      i "
245              << setw(15) << " T r ng  th  i " << endl;
246         cout << string(70, '-') << endl;
247 
248         for (int i = 0; i < numVertices; i++) {
249             cout << left << setw(6) << i;
250 
251             if (dist[i] == INT_MAX) {
252                 cout << setw(12) << "    "
253                      << setw(8) << "-"
254                      << setw(10) << "-"
255                      << setw(15) << "Kh ng    t       c    "
256                      << setw(15) << "Kh ng  li  n  th ng";
257             } else {
258                 cout << setw(12) << dist[i]
259                      << setw(8) << (parent[i] == -1 ? "-" : to_string(
    parent[i]))
260                      << setw(10) << (parentEdgeId[i] == -1 ? "-" :
    to_string(parentEdgeId[i]));
261 
262                 string pathStr = getPathString(source, i);
263                 cout << setw(15) << pathStr;
264 
265                 string status = (i == source) ? " N g u n " : "   t
       c    ";
266                 cout << setw(15) << status;
267             }
268             cout << endl;
269         }
270         cout << string(70, '-') << endl;
271     }
272 
273     string getPathString(int source, int dest) {
274         if (dest == source) return to_string(dest);
275         if (parent[dest] == -1) return "N/A";
276 
277         vector<int> path;
278         int current = dest;
```

```
279
280         while (current != -1) {
281             path.push_back(current);
282             current = parent[current];
283         }
284
285         reverse(path.begin(), path.end());
286
287         string result;
288         for (int i = 0; i < path.size(); i++) {
289             result += to_string(path[i]);
290             if (i < path.size() - 1) result += "   ";
291         }
292
293         return result;
294     }
295
296     void printGraphStatistics() {
297         cout << "\n===  T H NG  K          T H   ===" << endl;
298         cout << " S      nh   : " << stats.numVertices << endl;
299         cout << " S    c nh : " << stats.numEdges << endl;
300         cout << " L o i        t h  : " << (stats.isDirected ? "C
    h   ng " : "V   h   ng ") << endl;
301         cout << "Li  n  th ng: " << (stats.isConnected ? "C  " : "Kh ng
    ") << endl;
302         cout << " S   th nh  p h n  li  n  th ng: " << stats.components.
    size() << endl;
303
304         cout << "\ n T h ng  k   l o i   c nh :" << endl;
305         for (const auto& pair : stats.edgeTypeCount) {
306             cout << "  " << pair.first << ": " << pair.second << endl;
307         }
308
309         if (stats.numSelfLoops > 0) {
310             cout << "C   " << stats.numSelfLoops << " v ng  l p " <<
    endl;
311         }
312
313         if (stats.numParallelEdges > 0) {
314             cout << "C    c nh  song song" << endl;
315         }
316
317         cout << endl;
318     }
319
320     void runCompleteAnalysis(int source) {
321         cout << "=== PH  N  T CH  HO  N  C H NH        T H   ===" << endl;
322
323         // 1. In        t h
324         printGeneralGraph();
325
326         // 2.  T h ng  k         t h
327         printGraphStatistics();
328
329         // 3. Ph  n  t  ch  t  nh  li  n  th ng
330         analyzeConnectivity();
```

```
331
332         // 4.  C h y  Dijkstra
333         dijkstraGeneral(source);
334
335         // 5. In  k t  q u  chi  t i t
336         printDetailedResults(source);
337
338         // 6. Ph n  t ch  c  c  t r  ng  h p  c  b i t
339         analyzeSpecialCases(source);
340     }
341
342     void analyzeSpecialCases(int source) {
343         cout << "\n===  PH  N  T  CH  T R  NG  H P  C  B I T  ===" <<
    endl;
344
345         //  nh  kh ng  t  c
346         vector<int> unreachable;
347         for (int i = 0; i < numVertices; i++) {
348             if (dist[i] == INT_MAX) {
349                 unreachable.push_back(i);
350             }
351         }
352
353         if (!unreachable.empty()) {
354             cout << "  nh  kh ng  t  c  t  " << source <<
    ": ";
355             for (int i = 0; i < unreachable.size(); i++) {
356                 cout << unreachable[i];
357                 if (i < unreachable.size() - 1) cout << ", ";
358             }
359             cout << endl;
360         }
361
362         //  ng  i  d i  n h t
363         int maxDist = 0;
364         int farthestVertex = -1;
365         for (int i = 0; i < numVertices; i++) {
366             if (dist[i] != INT_MAX && dist[i] > maxDist) {
367                 maxDist = dist[i];
368                 farthestVertex = i;
369             }
370         }
371
372         if (farthestVertex != -1) {
373             cout << "  nh  xa  n h t  t  " << source << ": " <<
    farthestVertex
374                  << " ( k h o ng  c  ch: " << maxDist << ")" << endl;
375         }
376
377         //  S  nh  trong  c ng  th nh  p h n  li  n  th ng
378         int reachableCount = 0;
379         for (int i = 0; i < numVertices; i++) {
380             if (dist[i] != INT_MAX) reachableCount++;
381         }
382
383         cout << "  nh  t  c  : " << reachableCount << "/" <<
```

```
    numVertices
384             << " (" << (100.0 * reachableCount / numVertices) << "%)"
    << endl;
385
386         cout << endl;
387     }
388
389     // C h y  Dijkstra  t    t t   c      nh    (All-Pairs Shortest Path)
390     void allPairsShortestPath() {
391         cout << "===  T T   C    C P      NG      I   N G N   N H T  ==="
    << endl;
392
393         vector<vector<int>> allDist(numVertices, vector<int>(numVertices
    , INT_MAX));
394
395         for (int source = 0; source < numVertices; source++) {
396             dijkstraGeneral(source);
397
398             for (int dest = 0; dest < numVertices; dest++) {
399                 allDist[source][dest] = dist[dest];
400             }
401         }
402
403         // In ma  t r n  k h o ng  c ch
404         cout << "\nMa  t r n  k h o ng  c ch  n g n  n h t :" << endl;
405         cout << setw(6) << "";
406         for (int j = 0; j < numVertices; j++) {
407             cout << setw(6) << j;
408         }
409         cout << endl;
410
411         for (int i = 0; i < numVertices; i++) {
412             cout << setw(6) << i;
413             for (int j = 0; j < numVertices; j++) {
414                 if (allDist[i][j] == INT_MAX) {
415                     cout << setw(6) << "   ";
416                 } else {
417                     cout << setw(6) << allDist[i][j];
418                 }
419             }
420             cout << endl;
421         }
422         cout << endl;
423     }
424 };
```

Listing 6: Dijkstra tổng quát - C++

```
1  import heapq
2  from typing import List, Tuple, Dict, Optional, Set
3  from collections import defaultdict
4  import sys
5
6  class GeneralEdge:
7      """ C nh  trong       t h   t ng  q u  t """
8      def __init__(self, from_vertex: int, to_vertex: int, weight: int,
```

```
                    edge_id: int, edge_type: str = "normal"):
        self.from_vertex = from_vertex
        self.to_vertex = to_vertex
        self.weight = weight
        self.id = edge_id
        self.type = edge_type  # "normal", "self-loop", "parallel"

class GraphStatistics:
    """ T h ng  k         t h  """
    def __init__(self):
        self.num_vertices = 0
        self.num_edges = 0
        self.num_self_loops = 0
        self.num_parallel_edges = 0
        self.is_directed = False
        self.is_connected = False
        self.components = []
        self.edge_type_count = defaultdict(int)

class GeneralDijkstra:
    """Dijkstra cho       t h   t ng  qu t"""

    def __init__(self, num_vertices: int, is_directed: bool = False):
        self.num_vertices = num_vertices
        self.is_directed = is_directed
        self.adj_list = [[] for _ in range(num_vertices)]
        self.edge_id_counter = 0
        self.stats = GraphStatistics()
        self.edge_key_count = defaultdict(int)

        self.stats.num_vertices = num_vertices
        self.stats.is_directed = is_directed
        self.reset()

    def reset(self):
        """Reset  t r ng  th i  cho  l n  c h y  m i """
        self.dist = [float('inf')] * self.num_vertices
        self.parent = [-1] * self.num_vertices
        self.parent_edge_id = [-1] * self.num_vertices
        self.visited = [False] * self.num_vertices
        self.in_current_component = [False] * self.num_vertices

    def add_edge(self, u: int, v: int, weight: int):
        """Th m  c nh  v o       t h  """
        edge_key = f"{min(u, v)}-{max(u, v)}"
        self.edge_key_count[edge_key] += 1

        edge_type = "normal"
        if u == v:
            edge_type = "self-loop"
            self.stats.num_self_loops += 1
        elif self.edge_key_count[edge_key] > 1:
            edge_type = "parallel"
            self.stats.num_parallel_edges += 1

        # Th m  c nh  t  u  n  v
```

```
65          self.adj_list[u].append(GeneralEdge(u, v, weight, self.
    edge_id_counter, edge_type))
66
67          #  N u  l          t h   v   h   ng  v   kh ng  p h i  v ng
     l p
68          if not self.is_directed and u != v:
69              self.adj_list[v].append(GeneralEdge(v, u, weight, self.
    edge_id_counter, edge_type))
70
71          self.stats.num_edges += 1
72          self.stats.edge_type_count[edge_type] += 1
73
74          direction = " c   h   ng  " if self.is_directed else ""
75          print(f"Th  m   c nh  {direction}{u} -> {v} ( t r ng   s  : {
    weight}, "
76                f" l o i : {edge_type}, ID: {self.edge_id_counter})")
77
78          self.edge_id_counter += 1
79
80      def dijkstra_general(self, source: int):
81          """Dijkstra  t ng  qu  t """
82          self.reset()
83
84          heap = [(0, source)]
85          self.dist[source] = 0
86
87          graph_type = " c   h   ng " if self.is_directed else " v
     h   ng "
88          print(f"\nDijkstra  t ng  qu t  t     nh   {source} tr n
            t h  {graph_type}:")
89
90          processed_vertices = 0
91
92          while heap:
93              current_dist, u = heapq.heappop(heap)
94
95              if self.visited[u]:
96                  continue
97
98              self.visited[u] = True
99              self.in_current_component[u] = True
100             processed_vertices += 1
101
102             print(f" B   c  {processed_vertices}:  X   l     nh   {u} (
     k h o ng  c  c h: {current_dist})")
103
104             #  X   l   t t   c   c  c  c nh   t   u
105             for edge in self.adj_list[u]:
106                 v = edge.to_vertex
107                 weight = edge.weight
108
109                 print(f"   K i m  tra  c nh  {u} -> {v} ( t r ng   s  : {
    weight},  l o i : {edge.type})", end="")
110
111                 #  X   l     c    b i t  cho  v  n g   l p
112                 if edge.type == "self-loop":
```

```
113                     print(" [V ng   l p   -   b    qua]")
114                     continue
115
116                 if not self.visited[v]:
117                     new_dist = self.dist[u] + weight
118
119                     if new_dist < self.dist[v]:
120                         self.dist[v] = new_dist
121                         self.parent[v] = u
122                         self.parent_edge_id[v] = edge.id
123                         heapq.heappush(heap, (new_dist, v))
124
125                         print(f" [ C p   n h t  dist[{v}] = {new_dist}]")
126                     else: print(" [Kh ng   c p   n h t ]")
127                 else:
128                     print(" [   nh         th  m]")
129
130         print(f"      x    l   {processed_vertices}/{self.num_vertices}
      nh   ")
131
132     def analyze_connectivity(self):
133         """Ph  n   t  ch   t  nh   li  n   th  ng"""
134         self.stats.components = []
135         global_visited = [False] * self.num_vertices
136
137         for i in range(self.num_vertices):
138             if not global_visited[i]:
139                 component = []
140                 self._dfs_component(i, global_visited, component)
141                 self.stats.components.append(component)
142
143         self.stats.is_connected = (len(self.stats.components) == 1)
144
145         print("\nPh  n   t  ch   t  nh   li  n   th  ng:")
146         print(f" S    th  nh   p h n   li  n   th  ng: {len(self.stats.
    components)}")
147
148         for i, component in enumerate(self.stats.components):
149             print(f"Th  nh   p h n   {i + 1}: {{{', '.join(map(str,
    component))}}}")
150
151     def _dfs_component(self, u: int, global_visited: List[bool],
    component: List[int]):
152         """DFS        t  m   th  nh   p h n   li  n   th  ng"""
153         global_visited[u] = True
154         component.append(u)
155
156         for edge in self.adj_list[u]:
157             v = edge.to_vertex
158             if not global_visited[v]:
159                 self._dfs_component(v, global_visited, component)
160
161     def print_general_graph(self):
162         """In         t  h    t  ng   qu  t"""
163         graph_type = " c    h    ng " if self.is_directed else " v
     h    ng "
```

```
        print(f"\ n      t h    t ng  qu  t  ({graph_type}):")

        for i in range(self.num_vertices):
            print(f"  nh    {i}: ", end="")

            # Nh m  c nh  theo   ch   v   l o i
            edge_groups = defaultdict(list)
            for edge in self.adj_list[i]:
                edge_groups[edge.to_vertex].append(edge)

            for dest, edges in edge_groups.items():
                print(f"{dest}", end="")
                if len(edges) > 1 or edges[0].type != "normal":
                    weights = []
                    for edge in edges:
                        weight_str = str(edge.weight)
                        if edge.type == "self-loop":
                            weight_str += "L"
                        elif edge.type == "parallel" and len(edges) > 1:
                            weight_str += "P"
                        weights.append(weight_str)
                    print(f"[{','.join(weights)}]", end="")
                else:
                    print(f"({edges[0].weight})", end="")
                print(" ", end="")
            print()
        print()

    def print_detailed_results(self, source: int):
        """In  k t  q u  chi  t i t """
        print("\n===  K T  Q U  CHI  T I T  DIJKSTRA ===")
        graph_type = "c   h   ng " if self.is_directed else "v
    h   ng "
        print(f" N g u n : {source} |       t h  : {graph_type}")
        print("-" * 70)

        headers = ["  nh  ", " K h o ng  c ch", "Cha", " C nh  ID", "
       ng     i ", " T r ng  th  i"]
        print(f"{headers[0]:<6} {headers[1]:<12} {headers[2]:<8} {
    headers[3]:<10} {headers[4]:<15} {headers[5]:<15}")
        print("-" * 70)

        for i in range(self.num_vertices):
            print(f"{i:<6}", end="")

            if self.dist[i] == float('inf'):
                print(f"{'   ':<12} {'-':<8} {'-':<10} {'Kh ng   t
       c   ':<15} {'Kh ng  li  n  th ng':<15}")
            else:
                parent_str = "-" if self.parent[i] == -1 else str(self.
    parent[i])
                edge_id_str = "-" if self.parent_edge_id[i] == -1 else
    str(self.parent_edge_id[i])
                path_str = self._get_path_string(source, i)
                status = " N g u n " if i == source else "  t      c   "

```

```
214                 print(f"{self.dist[i]:<12} {parent_str:<8} {edge_id_str
    :<10} {path_str:<15} {status:<15}")
215
216         print("-" * 70)
217
218     def _get_path_string(self, source: int, dest: int) -> str:
219         """ L y   c h u i   b i u   d i n      ng      i """
220         if dest == source:
221             return str(dest)
222         if self.parent[dest] == -1:
223             return "N/A"
224
225         path = []
226         current = dest
227
228         while current != -1:
229             path.append(current)
230             current = self.parent[current]
231
232         path.reverse()
233         return "   ".join(map(str, path))
234
235     def print_graph_statistics(self):
236         """In  t h ng  k          t h  """
237         print("\n===  T H NG  K          T H   ===")
238         print(f" S      nh   : {self.stats.num_vertices}")
239         print(f" S    c nh : {self.stats.num_edges}")
240         graph_type = " C    h    ng " if self.stats.is_directed else " V
    h    ng "
241         print(f" L o i         t h  : {graph_type}")
242         connected_str = " C  " if self.stats.is_connected else "Kh ng"
243         print(f"Li  n  th ng: {connected_str}")
244         print(f" S   th nh  p h n  li  n  th ng: {len(self.stats.
    components)}")
245
246         print("\ n T h ng  k    l o i   c nh :")
247         for edge_type, count in self.stats.edge_type_count.items():
248             print(f"  {edge_type}: {count}")
249
250         if self.stats.num_self_loops > 0:
251             print(f" C   {self.stats.num_self_loops} v  ng   l p ")
252
253         if self.stats.num_parallel_edges > 0:
254             print(" C    c nh  song song")
255
256         print()
257
258     def run_complete_analysis(self, source: int):
259         """ C h y  ph n  t ch  ho  n  c h nh """
260         print("===  PH  N  T  CH  HO  N  C H NH         T H   ===")
261
262         # 1. In         t h
263         self.print_general_graph()
264
265         # 2.  T h ng  k          t h
266         self.print_graph_statistics()
```

```
267
268         # 3. Ph n t ch t nh li n th ng
269         self.analyze_connectivity()
270
271         # 4.  C h y  Dijkstra
272         self.dijkstra_general(source)
273
274         # 5. In  k t  q u  chi  t i t
275         self.print_detailed_results(source)
276
277         # 6. Ph n t ch c c  t r  ng  h p  c  b i t
278         self.analyze_special_cases(source)
279
280     def analyze_special_cases(self, source: int):
281         """Ph n t ch  t r  ng  h p  c  b i t """
282         print("\n=== PH N T CH  T R  NG  H P  C  B I T  ===")
283
284         #  nh  kh ng  t  c
285         unreachable = [i for i in range(self.num_vertices) if self.dist[
    i] == float('inf')]
286
287         if unreachable:
288             print(f"  nh  kh ng  t  c  t  {source}: {', '.
    join(map(str, unreachable))}")
289
290         #  ng  i  d i  n h t
291         max_dist = 0
292         farthest_vertex = -1
293         for i in range(self.num_vertices):
294             if self.dist[i] != float('inf') and self.dist[i] > max_dist:
295                 max_dist = self.dist[i]
296                 farthest_vertex = i
297
298         if farthest_vertex != -1:
299             print(f"  nh  xa  n h t  t  {source}: {farthest_vertex} (
    k h o ng  c  ch: {max_dist})")
300
301         #  S  nh  trong c ng th nh  p h n  li n  th ng
302         reachable_count = sum(1 for d in self.dist if d != float('inf'))
303         percentage = (100.0 * reachable_count / self.num_vertices)
304
305         print(f"  nh  t  c  : {reachable_count}/{self.
    num_vertices} ({percentage:.1f}%)")
306         print()
307
308     def all_pairs_shortest_path(self):
309         """ T t  c  c p  ng  i  n g n  n h t """
310         print("===  T T  C  C P  NG  I  N G N  N H T  ===")
311
312         all_dist = []
313
314         for source in range(self.num_vertices):
315             self.dijkstra_general(source)
316             all_dist.append(self.dist[:])
317
318         # In ma  t r n  k h o ng  c  ch
```

```
        print("\nMa  t r n  k h o ng  c ch  n g n  n h t :")
        print(f"{'':>6}", end="")
        for j in range(self.num_vertices):
            print(f"{j:>6}", end="")
        print()

        for i in range(self.num_vertices):
            print(f"{i:>6}", end="")
            for j in range(self.num_vertices):
                if all_dist[i][j] == float('inf'):
                    print(f"{'  ':>6}", end="")
                else:
                    print(f"{all_dist[i][j]:>6}", end="")
            print()
        print()

def demonstrate_general_dijkstra():
    """Demo Dijkstra  t ng  qu t"""
    print("=== DEMO DIJKSTRA CHO  T H  T NG  QU T ===\n")

    # Test case 1:  t h  v  h  ng  v i  c nh  song song
    print(">>> TEST CASE 1:  t h  v  h  ng  v i  c nh  song song <<<")
    graph1 = GeneralDijkstra(5, is_directed=False)

    graph1.add_edge(0, 1, 4)
    graph1.add_edge(0, 1, 2)  #  C nh  song song
    graph1.add_edge(0, 2, 3)
    graph1.add_edge(1, 3, 1)
    graph1.add_edge(2, 3, 5)
    graph1.add_edge(3, 4, 2)
    graph1.add_edge(1, 1, 3)  # V ng  l p

    graph1.run_complete_analysis(0)

    print("\n" + "="*60 + "\n")

    # Test case 2:  t h  c  h  ng  kh ng  li n  th ng
    print(">>> TEST CASE 2:  t h  c  h  ng  kh ng  li n  th ng <<<")
    graph2 = GeneralDijkstra(6, is_directed=True)

    graph2.add_edge(0, 1, 2)
    graph2.add_edge(1, 2, 3)
    graph2.add_edge(3, 4, 1)
    graph2.add_edge(4, 5, 4)
    graph2.add_edge(2, 2, 1)  # V ng  l p

    graph2.run_complete_analysis(0)
    graph2.all_pairs_shortest_path()

if __name__ == "__main__":
    demonstrate_general_dijkstra()
```

Listing 7: Dijkstra tổng quát - Python

6 So sánh và đánh giá

6.1 Bảng so sánh các biến thể Dijkstra

Tiêu chí	Đồ thị đơn	Đa đồ thị	Đồ thị tổng quát
Độ phức tạp thời gian	$O((V+E)\log V)$	$O((V+E)\log V)$	$O((V+E)\log V)$
Độ phức tạp không gian	$O(V+E)$	$O(V+E+P)$	$O(V+E+P+L)$
Xử lý cạnh song song	Không có	Có	Có
Xử lý vòng lặp	Không có	Có	Có
Hỗ trợ có hướng	Có	Có	Có
Phân tích liên thông	Cơ bản	Cơ bản	Chi tiết
Thống kê đồ thị	Không	Cơ bản	Đầy đủ
Xử lý trường hợp đặc biệt	Không	Một phần	Đầy đủ

Chú thích: P = số cạnh song song, L = số vòng lặp

6.2 Ưu nhược điểm từng phương pháp

Dijkstra cho đồ thị đơn:

- **Ưu điểm:** Đơn giản, hiệu quả, dễ hiểu và cài đặt
- **Nhược điểm:** Chỉ áp dụng cho đồ thị đơn giản
- **Ứng dụng:** Bài toán cơ bản, giáo dục, prototype

Dijkstra cho đa đồ thị:

- **Ưu điểm:** Xử lý cạnh song song, linh hoạt hơn
- **Nhược điểm:** Phức tạp hơn, cần xử lý thêm trường hợp đặc biệt
- **Ứng dụng:** Mạng giao thông với nhiều tuyến đường, mạng máy tính

Dijkstra tổng quát:

- **Ưu điểm:** Xử lý mọi loại đồ thị, phân tích toàn diện
- **Nhược điểm:** Phức tạp nhất, overhead cao
- **Ứng dụng:** Nghiên cứu, phân tích đồ thị phức tạp, hệ thống thực tế

7 Ứng dụng thực tế

7.1 GPS và Navigation Systems

- Tìm đường đi ngắn nhất giữa hai điểm

- Xử lý đường một chiều (đồ thị có hướng)
- Nhiều tuyến đường giữa hai giao lộ (đa đồ thị)
- Cập nhật thời gian thực về tình trạng giao thông

7.2 Network Routing Protocols

- OSPF (Open Shortest Path First) protocol
- IS-IS (Intermediate System to Intermediate System)
- Tối ưu hóa băng thông và độ trễ
- Xử lý link failures và recovery

7.3 Game Development

- AI pathfinding cho NPCs
- Tối ưu hóa di chuyển trong game world
- Xử lý terrain costs và obstacles
- Real-time strategy games

7.4 Social Network Analysis

- Tìm mối quan hệ gần nhất giữa người dùng
- Phân tích độ ảnh hưởng (influence propagation)
- Community detection
- Recommendation systems

8 Tối ưu hóa và cải tiến

8.1 Dijkstra với A* heuristic

```
class AStarDijkstra {
private:
    vector<pair<double, double>> coordinates; //  T a           nh

    double euclideanDistance(int u, int v) {
        double dx = coordinates[u].first - coordinates[v].first;
        double dy = coordinates[u].second - coordinates[v].second;
        return sqrt(dx*dx + dy*dy);
    }

public:
```

```
    void aStarSearch(int source, int target) {
        priority_queue<pair<double, int>, vector<pair<double, int>>,
                       greater<pair<double, int>>> pq;

        vector<double> gScore(numVertices, DBL_MAX);
        vector<double> fScore(numVertices, DBL_MAX);

        gScore[source] = 0;
        fScore[source] = euclideanDistance(source, target);

        pq.push({fScore[source], source});

        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();

            if (u == target) {
                cout << "     t  m   t h y     ng     i     n     ch   !"
   << endl;
                return;
            }

            if (visited[u]) continue;
            visited[u] = true;

            for (const Edge& edge : adjList[u]) {
                int v = edge.to;
                double tentativeGScore = gScore[u] + edge.weight;

                if (tentativeGScore < gScore[v]) {
                    parent[v] = u;
                    gScore[v] = tentativeGScore;
                    fScore[v] = gScore[v] + euclideanDistance(v, target)
   ;

                    if (!visited[v]) {
                        pq.push({fScore[v], v});
                    }
                }
            }
        }
    }
};
```

Listing 8: A* Algorithm - Cải tiến của Dijkstra

8.2 Bidirectional Dijkstra

```
class BidirectionalDijkstra {
private:
    vector<int> distForward, distBackward;
    vector<bool> visitedForward, visitedBackward;

public:
    int bidirectionalSearch(int source, int target) {
```

```
        distForward.assign(numVertices, INT_MAX);
        distBackward.assign(numVertices, INT_MAX);
        visitedForward.assign(numVertices, false);
        visitedBackward.assign(numVertices, false);

        priority_queue<pair<int, int>, vector<pair<int, int>>,
                       greater<pair<int, int>>> pqForward, pqBackward;

        distForward[source] = 0;
        distBackward[target] = 0;
        pqForward.push({0, source});
        pqBackward.push({0, target});

        int bestDistance = INT_MAX;

        while (!pqForward.empty() || !pqBackward.empty()) {
            //  M    r ng   t   ph a source
            if (!pqForward.empty()) {
                int u = pqForward.top().second;
                pqForward.pop();

                if (!visitedForward[u]) {
                    visitedForward[u] = true;

                    if (visitedBackward[u]) {
                        bestDistance = min(bestDistance,
                                           distForward[u] + distBackward[u
]);
                    }

                    // Relaxation cho forward search
                    for (const Edge& edge : adjList[u]) {
                        int v = edge.to;
                        if (distForward[u] + edge.weight < distForward[v
]) {
                            distForward[v] = distForward[u] + edge.
weight;
                            pqForward.push({distForward[v], v});
                        }
                    }
                }
            }

            //  M    r ng   t   ph a target ( t   ng   t  )
            if (!pqBackward.empty()) {
                int u = pqBackward.top().second;
                pqBackward.pop();

                if (!visitedBackward[u]) {
                    visitedBackward[u] = true;

                    if (visitedForward[u]) {
                        bestDistance = min(bestDistance,
                                           distForward[u] + distBackward[u
]);
                    }
```

```
60
61                      // Relaxation cho backward search
62                      for (const Edge& edge : reverseAdjList[u]) {
63                          int v = edge.to;
64                          if (distBackward[u] + edge.weight < distBackward
    [v]) {
65                              distBackward[v] = distBackward[u] + edge.
    weight;
66                              pqBackward.push({distBackward[v], v});
67                          }
68                      }
69                  }
70              }
71          }
72
73          return bestDistance;
74      }
75  };
```

Listing 9: Bidirectional Dijkstra

9 Kết luận

9.1 Tóm tắt kiến thức

Thuật toán Dijkstra là một trong những thuật toán quan trọng nhất trong lý thuyết đồ thị với các đặc điểm chính:

- **Nguyên lý:** Sử dụng chiến lược tham lam để tìm đường đi ngắn nhất
- **Điều kiện:** Chỉ áp dụng với trọng số không âm
- **Độ phức tạp:** $O((V + E) \log V)$ với heap, $O(V^2)$ với mảng
- **Tính đúng đắn:** Đảm bảo tìm được đường đi ngắn nhất tối ưu

9.2 Ứng dụng quan trọng

- Hệ thống định vị GPS và navigation
- Giao thức định tuyến mạng (OSPF, IS-IS)
- Trí tuệ nhân tạo trong game (pathfinding)
- Phân tích mạng xã hội và hệ thống khuyến nghị
- Tối ưu hóa logistic và supply chain

9.3 Hướng phát triển

- Kết hợp với heuristic (A*, IDA*)
- Dijkstra song song (parallel processing)
- Ứng dụng trong machine learning và deep learning
- Tối ưu hóa cho đồ thị động (dynamic graphs)
- Xử lý đồ thị lớn (big graph processing)