Vietnam National University, Ho Chi Minh City
University of Technology
Faculty of Computer Science and Engineering

DISCRETE STRUCTURE
(CO1007)

Assignment

BELLMAN-FORD ALGORITHM

(UPDATE 21/05/2024)

Instructor(s): Nguyễn Văn Minh Mẫn, *Mahidol University*
Nguyễn An Khương, *CSE-HCMUT*
Trần Tuấn Anh, *CSE-HCMUT*
Nguyễn Tiến Thịnh, *CSE-HCMUT*
Trần Hồng Tài, *CSE-HCMUT*
Mai Xuân Toàn, *CSE-HCMUT*

Note: **To reduce the chance and effectiveness of cheating cases, there will be 4 harmony questions in the final exam.**

Ho Chi Minh City, March 2024

Table of Contents

1 Introduction

1.1 Graph

These discrete structures consist of vertices (nodes) and edges that connect these vertices. Graphs come in various forms, depending on specific characteristics:

- **Directed vs. Undirected Graphs:** In directed graphs, edges have a direction (one-way), while undirected graphs have bidirectional edges.
- **Multigraphs:** These allow multiple edges to connect the same pair of vertices.
- **Loops:** Some graphs permit loops (edges connecting a vertex to itself).

1.2 Applications

Graphs serve as powerful models across diverse disciplines. Here are some applications:

1. **Street Navigation:** Using graph models, we can determine if it's feasible to walk all streets in a city without retracing our steps.
2. **Map Coloring:** Graphs help find the minimum number of colors needed to color regions on a map without adjacent regions having the same color.
3. **Circuit Design:** We assess whether a circuit can be implemented on a planar circuit board using graph theory.
4. **Chemical Structures:** Graphs distinguish between chemical compounds with identical molecular formulas but different structures.
5. **Network Connectivity:** Graph models verify whether two computers are connected via a communication link.
6. **Weighted Graphs:** Assigning weights to edges allows us to solve problems like finding the shortest path between cities in a transportation network.
7. **Scheduling and Channel Assignment:** Graphs aid in scheduling exams and allocating channels to television stations.

Graph theory's versatility makes it an indispensable tool for problem-solving and analysis.

1.3 Path

Definition 1. *Let n be a non-negative integer and G an undirected graph. A path of length n from u to v in G is a sequence of n edges $e_1, ..., e_n$ of G for which there exists a sequence $x_0 = u, x_1, ..., x_{n-1}, x_n = v$ of vertices such that e_i has, for $i = 1, ..., n$, the endpoints x_{i-1} and x_i. When the graph is simple, we denote this path by its vertex sequence $x_0, x_1, ..., x_n$ (because listing these vertices uniquely determines the path). The path is a circuit if it begins and ends at the same vertex, that is, if $u = v$, and has a length greater than zero. The path or circuit is said to pass through the vertices $x_1, x_2, ..., x_{n-1}$ or traverse the edges $e_1, e_2, ..., e_n$. A path or circuit is simple if it does not contain the same edge more than once.*

Determining a path of least length between two vertices in a network is one such problem. To be more specific, let the length of a path in a weighted graph be the sum of the weights of the edges of this path. The question is: What is a shortest path, that is, a path of least length, between two given vertices?

1.4 Bellman-Ford Algorithm

```
procedure BellmanFord(G,a)
// Initialization Step
  forall vertices v
     Label[v] := ∞
     Prev[v] := -1
  Label(a) := 0 // a is the source node

// Iteration Step
  for i from 1 to size(vertices)-1
    forall vertices v
      if (Label[u] + Wt(u,v)) < Label[v]
        then
          Label[v] := Label[u] + Wt(u,v)
          Prev[v] := u

// Check circuit of negative weight
     forall vertices v
       if (Label[u] + Wt(u,v)) < Label(v)
         error "Contains circuit of negative weight"
```

Property: any G, any weighted; one-to-all; detect whether there exists a circle of negative length; complexity $O(|V| \times |E|)$.

1.5 The Travelling salesman problem

"The travelling salesman problem, also known as the travelling salesperson problem (TSP), asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in theoretical computer science and operations research" - from Wikipedia

2 Exercise

Students do the following exercises. Constraints on inputs and outputs are given in Section 3.

1. **Bellman-Ford calculation:** Write function BF() to calculate 1 step of Bellman-Ford on an input graph.
   Input:
   - The Graph
   - Number of vertices
   - Start vertex (A-Z)
   - Current Bellman-Ford Value array
   - Current Bellman-Ford Previous array

   Output:
   - Updated Bellman-Ford Value array based on the current Graph.

- Updated Bellman-Ford Previous matrix based on the current Graph.

2. **Bellman-Ford path:** Write function BF_Path() to return a string containing the Bellman-Ford path from input start and goal vertices.
   Input:
   - The Graph
   - Number of vertices
   - Start vertex (A-Z)
   - Goal vertex (A-Z)

   Bellman-Ford matrix **Output:**
   - a string containing the shortest path from Start vertex to Goal vertex

3. **Travelling sale man path calculation:** write function Traveling() to calculate the shortest way to go over all the vertices in the graph and go back to the starting vertex.
   Input:
   - The Graph
   - Number of vertices (updated 13/05/2024)
   - Start vertex (A-Z)

   Output:
   - Print the shortest path over all the vertices and return to the starting vertex.

3 Detail description

- The input graphs will be weight matrices, where all the connected weights are positive integers, and 0 means not-connected. (Datatype int[][])
- The Value array for Bellman-Ford is initiated as an all -1 array, Element [i] is the distance of the path from the Start vertex to vertex i_{th}. (Datatype int[])
- The Previous array for Bellman-Ford is initiated as an all -1 array, Element [i] is the previous vertex on the path to vertex i_{th} from the Start vertex. (Datatype int[])
- The output of the BF_Path() function is a string containing the names of the vertices in the path separate by a whitespace for example: "A D C B" representing the path going from A to D to C and then B. (Datatype string)
- The Start and Goal are type char. The Vertices will always be named using the uppercase alphabet in order where vertex 0_{th} is 'A', 1_{st} is 'B', and so on. (Datatype char)

3.1 Update List

(updated 21/05/2024)

- Updated deadline to **June 9, 2024**.
- Fixed the "partile" calculation in TSM sample input.
- Q5 and further in Q&A

3.2 Q&A

(updated 21/05/2024)

- Q1: What is the output for the negative circle case in the BF_Path function?
  A1: All the input weights are positive
- Q2: What is the output for the case where the start and goal vertices are the same in the BF_Path function?
  A2: A string with 1 letter.
- Q3: Can the return type of the functions be changed?
  A3: No.
- Q4: Does the input for the Traveling function lack the number of vertices?
  A4: Noted and updated.
- Q5: Why are there G and G2 graphs in 1 test case?
  A5: Since BF function only calculates 1 step of Bellman_Ford algorithm for each call it should be able to calculate when the graph is changed.
- Q6: Can the Input type of functions be changed?
  A6: No, but you can convert and calculate with temporary variables in your functions.
- Q7: Is the size of the input graph fixed or not?
  A7: It is not fixed, but the input is a static array and the test cases' graphs are planned to be a maximum of 20 vertices so you can fix the maximum array to [20][20].

4 Instructions and requirements

Students have to follow the instructions and comply with the requirements below. *Lecturers do not solve the cases arising because students do not follow the instructions or do not comply with the requirements.*

4.1 Instructions

If you have any questions about the assignment during work, **please post that question on the class forum on BKeL.**

4.2 Requirements

- Deadline for submission: **June 9, 2024**. Students have to answer each question clearly and coherently.
- Programming languages: C++

4.3 Submission

Students submit and run the code on bkel system automatic evaluation before submitting the files in the below requirement (the place for submitting code will be available later on the video site) Students are required to submit:

- main.cpp for the main function where you set up running all the questions of the assignment. Students decide for themselves what is the input and output in the main function to represent their result;
- bellman.cpp and bellman.h for Bellman-Ford related functions;
- tsm.cpp and tsm.h for travelling salesman related functions along with a tsm.pdf report file to explain your approach to finding the solution to the travelling salesman problem.

The file submissions are on the bkel video site and will be available near the end of the deadline.

4.4 Cheating treatment

The assignment has to be done by individuals. The student will be considered as cheating if:

- There is an unusual similarity between the codes. In this case, ALL similar submissions are considered cheating. Therefore, the student must protect their work.
- They do not understand the works written by themselves. You can consult from any source, but make sure that you know the meaning of everything you have written.

Students will be judged according to the university's regulations if the article is cheating.