CST 370
Sabrina Ferras
12.07.2023

All-Pairs Shortest Paths

Program in C++ that displays "All-Pairs Shortest Paths" by implementing Floyd's Algorithm

Following the algorithm below I wrote a program to take a adjacency matrix as an input, with -1 representing infinity and outputting “all-pairs shortest path.”

ALGORITHM *Floyd*$(W[1..n, 1..n])$

```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W  //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```

~ Introduction to the Design and Analysis of Algorithms, 3rd Edition

---

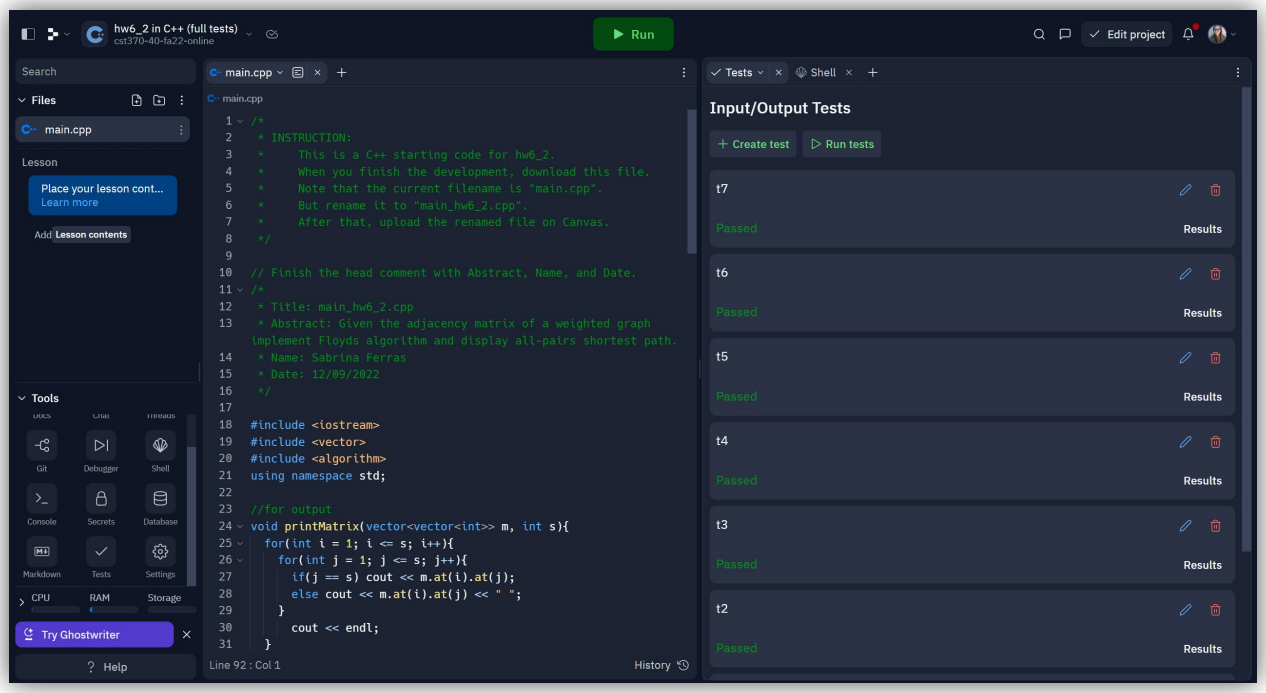
Code

```cpp
/*
 * INSTRUCTION:
 *     This is a C++ starting code for hw6_2.
 *     When you finish the development, download this file.
 *     Note that the current filename is "main.cpp".
 *     But rename it to "main_hw6_2.cpp".
 *     After that, upload the renamed file on Canvas.
 */

// Finish the head comment with Abstract, Name, and Date.
/*
 * Title: main_hw6_2.cpp
 * Abstract: Given the adjacency matrix of a weighted graph implement Floyds algorithm and display all-
pairs shortest path.
 * Name: Sabrina Ferras
 * Date: 12/09/2022
 */

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

//for output
void printMatrix(vector<vector<int>> m, int s){
  for(int i = 1; i <= s; i++){
    for(int j = 1; j <= s; j++){
      if(j == s) cout << m.at(i).at(j);
      else cout << m.at(i).at(j) << " ";
    }
      cout << endl;
  }
}
```

```
void floydsAlgorithm(vector<vector<int>>& m, int s){
int val1, val2, val3;
  //NOTE: -1 represents inf
  //for loop based off of floyds algorithm from the slides with variation because -1 = inf
  for(int k = 1; k <= s; k++){
    for(int i = 1; i <= s; i++){
      for(int j = 1; j <= s; j++){

        val1 = m.at(i).at(j);
        val2 = m.at(i).at(k);
        val3 = m.at(k).at(j);
        if(val1 == -1){  //if current place in inf
          if(val2 == -1 || val3 == -1) m.at(i).at(j) = -1;  //if other route is inf
          else m.at(i).at(j) = m.at(i).at(k) + m.at(k).at(j);  //if other route is not inf set equal to
that
        }else{  //if current place in not inf
          if(val2 == -1 || val3 == -1) continue;  //if other route is inf
          else m.at(i).at(j) = min(m.at(i).at(j), m.at(i).at(k) + m.at(k).at(j));  //if non are inf
normal value set
        }
        //commented out: works for if working with other numbers that are negative
        // if(i == j) continue;
        // if(j == k || i == k) continue;
        // m.at(i).at(j) = min(m.at(i).at(j), m.at(i).at(k) + m.at(k).at(j));

     }
    }
  }
  printMatrix(m, s);
}

void myAlgorithm(vector<vector<int>>& m, int s){
  int i = 1;
  int j = 1;

}

int main()
{
  //get input for number of verticies
  int numOfVerts;
  cin >> numOfVerts;

  vector<vector<int>> matrix(numOfVerts + 1, vector<int>(numOfVerts + 1));

  //get input of matrix
  int temp;
  for(int i = 1; i <= numOfVerts; i++){
    for(int j = 1; j <= numOfVerts; j++){
      cin >> temp;
      matrix.at(i).at(j) = temp;
    }
  }

  //send to floyds algorithm
  floydsAlgorithm(matrix, numOfVerts);

    return 0;
}


  return go(f, seed, [])
}
```

CST 370
Sabrina Ferras
12.07.2023

Tests



t0

Output

```
1  0 10 3 4
2  2 0 5 6
3  7 7 0 1
4  6 16 9 0
```

Input

```
1  4
2  0 -1 3 -1
3  2 0 -1 -1
4  -1 7 0 1
5  6 -1 -1 0
```

t6

Output

```
1  0 10 30 60 100 150 210 280 360 450
2  -1 0 20 50 90 140 200 270 350 440
3  -1 -1 0 30 70 120 180 250 330 420
4  -1 -1 -1 0 40 90 150 220 300 390
5  -1 -1 -1 -1 0 50 110 180 260 350
6  -1 -1 -1 -1 -1 0 60 130 210 300
7  -1 -1 -1 -1 -1 -1 0 70 150 240
8  -1 -1 -1 -1 -1 -1 -1 0 80 170
9  -1 -1 -1 -1 -1 -1 -1 -1 0 90
10 -1 -1 -1 -1 -1 -1 -1 -1 -1 0
```

Input

```
1  10
2  0 10 -1 -1 -1 -1 -1 -1 -1 -1
3  -1 0 20 -1 -1 -1 -1 -1 -1 -1
4  -1 -1 0 30 -1 -1 -1 -1 -1 -1
5  -1 -1 -1 0 40 -1 -1 -1 -1 -1
6  -1 -1 -1 -1 0 50 -1 -1 -1 -1
7  -1 -1 -1 -1 -1 0 60 -1 -1 -1
8  -1 -1 -1 -1 -1 -1 0 70 -1 -1
9  -1 -1 -1 -1 -1 -1 -1 0 80 -1
10 -1 -1 -1 -1 -1 -1 -1 -1 0 90
11 -1 -1 -1 -1 -1 -1 -1 -1 -1 0
```

t3

Output

```
1  0 -1 -1 -1
2  -1 0 -1 -1
3  -1 -1 0 -1
4  -1 -1 -1 0
```

Input

```
1  4
2  0 -1 -1 -1
3  -1 0 -1 -1
4  -1 -1 0 -1
5  -1 -1 -1 0
```