AllPairsSP-Floyd

August 7, 2020

1 All-Pairs Shortest Paths

https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/Floyd.html

1.1 Table of Contents

1.2 All-pairs shortest path problem

- find the shortest distance between all pairs of vertices in the graph
- for every $u, v \in V$, calculate $d(u, v)$
- one solution:
  - from each $v \in V$, run Dijkstra's algorithm starting from v
  - if G is sparse, (i.e. $|E| = \Theta(|V|)$), Dijkstra's algorithm has the cost of $\Theta(|V|^2 + |V||E|log|V|) = \Theta(|V|^2 log|V|)$ using priority queue; so all-pairs will cost $\Theta|V|^3 log|V|$
  - if G is dense, Dijkstra's algorithm (MinVertex version), yields cost of $\Theta(|V|^3)$

1.3 Floyd's Algorithm

- regardless the number of edges, Floyd's algorithm yields cost of $\Theta|V|^3$
- applies dynamic programming technique (avoids repeatedly solving the same subproblems)
- algorithm uses the concept of $k - path$
  - $k - path$ from vertex u to v is defined to be any path whose intermediate vertices (aside u and v) all have indices less than k
  - $0 - path$ is defined to be a direct edge from u to v
- following figure illustrates the concept of $k - path$
- path **1 -> 3** is a **0-path** by definition
- path **3 -> 0 -> 2** is NOT a **0-path**, but a **1-path** (as well as a **2-path**, a **3-path**, and a **4-path**)
- path **1 -> 3 -> 2** is a **4-path**
- all paths in the graph are **4-paths**

1.3.1 Floyd's algorithm steps:

- define $D_k(u, v)$ - the length of the shortest *k-path* from vertex u to v
  - assume that we already know the shortest *k-path* from u to v
- the shortest *(k+1)-path* either goes through vertex k or it does not

  - if it does go through k, the best path is the best *k-path* from u to k followed by the best *k-path* from k to v
  - otherwise, keep the best *k-path* seen before
- Floyd's algorithm simply checks all of the possibilities in a triple loop

[1]:
```
#include <iostream>
#include <vector>
#include <climits> // sizes of integral types INT_MAX
#include <algorithm>
#include <iomanip>

using namespace std;
```

[2]:
```
// 2D vector D stores all-pairs shortest paths
template<class T>
void Floyd(T & G, vector<vector<int> > & D) {
    // initialize D[u][v] with weights(u, v)
    for(int u=0; u<G.V; u++) {
        D.push_back(vector<int>(G.V, INT_MAX));
        for(int v=0; v<G.V; v++)
            D[u][v] = G.weight(u, v);
    }
    for(int k=0; k<G.V; k++) // compute all k paths for every u->v pairs
        for(int u=0; u<G.V; u++)
            for(int v=0; v<G.V; v++)
                // if there's a path via k and the distance is shorter; update
 ↪the distance
                if ((D[u][k] != INT_MAX) && (D[k][v] != INT_MAX) && (D[u][v] >
 ↪(D[u][k] + D[k][v])))
                    D[u][v] = D[u][k]+D[k][v];
}
```

1.3.2 Representing graph using adjacency matrix

[3]:
```
struct Graph {
    vector<vector<int> > graph;
    size_t V; //no. of vertices

    Graph(size_t v) {
        V = v;
        for (int i=0; i<V; i++) {
            //initialize to INT_MAX; means not connected
            vector<int> row(V, INT_MAX);
            graph.push_back(row);
            for (int j=0; j<V; j++)
                if(i==j)
                    graph[i][j] = 0; //distance between u to u is 0
```

```
        }
    }

    // add a new edge from node u to node v, with weight w
    void addEdge(int u, int v, int w) {
        graph[u][v] = w;
    }

    int weight(int u, int v) {
        return graph[u][v];
    }
};
```

1.3.3 Exercise: Find all-pairs shortest paths for the following graph

[4]:
```
// represent undirected graph shown in above diagram
// A->0, B->1, C->2, D->3, E->4
Graph graph(5);
vector<vector<int> > dist;
```

[5]:
```
graph.addEdge(0, 1, 10);
graph.addEdge(0, 3, 20);
graph.addEdge(0, 2, 3);
graph.addEdge(1, 3, 5);
graph.addEdge(2, 1, 2);
graph.addEdge(2, 4, 15);
graph.addEdge(3, 4, 11);
```

[6]:
```
Floyd(graph, dist);
```

[7]:
```
void printDistances(Graph & G, vector<vector<int> >&D) {
    int w = 8;
    cout << "Distance Matrix: " << endl;
    for (int u=0; u<G.V; u++) {
        cout << setw(w) << char(u+65);
    }
    cout << endl << setfill('-') << setw(w*G.V) << "" << endl;
    cout << setfill(' ');
    for(int u=0; u<G.V; u++) {
        cout << setw(0) << char(u+65) << "|";
        for (int v=0; v<G.V; v++)
            if (D[u][v] == INT_MAX)
                cout << setw(w) << "INF";
            else
                cout << setw(w) << D[u][v];
        cout << endl;
    }
```

```
}
```

[8]:
```
printDistances(graph, dist);
```

```
Distance Matrix:
       A       B       C       D       E
----------------------------------------
A|       0       5       3      10      18
B|     INF       0     INF       5      16
C|     INF       2       0       7      15
D|     INF     INF     INF       0      11
E|     INF     INF     INF     INF       0
```

1.4 Exercises

1. All Pairs Shortest Path - https://open.kattis.com/problems/allpairspath
   - check and update negative weights/cycles

[ ]:
```
```