Shortest Path

The Floyd-Warshall Algorithm

The Problem

- Given a weighted directed graph G with vertices (v_1, v_2, …, v_n)
- We want to know the shortest path from any point a to any point b in G
- This is called an All-Pairs Shortest Path Problem

The Solutions

- We can always use Dijkstra’s algorithm or the Bellman-Ford’s algorithm
- But they’re pretty slow when the graph is dense
  - Worst case for Dijkstra’s: $O(|V|^3 \log |V|)$
  - Worst case for Bellman-Ford: $O(|V|^4)$

Floyd-Warshall Algorithm

```
Algorithm FloydWarshall(G)
    for i ← 1 to n do
        for j ← 1 to n do
            if i == j then
                $D^0[i,j]$ ← 0;
            else if $(v_i,v_j)$ is an edge in G then
                $D^0[i,j]$ ← $w(v_i,v_j)$; /* <-- weight function w */
            else
                $D^0[i,j]$ ← $+\infty$;
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                $D^k[i,j] \leftarrow \min\{D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j]\}$;
    return matrix $D^n$
```

- Example

Sample C++ Code

```
#include <vector>
typedef unsigned int uint;
typedef std::vector< std::vector<uint> > matrix;
#define INF ~0U

struct Edge {
  Edge *next;
  uint end;
  uint weight;
};

void floyd_warshall(std::vector<Edge*> graph)
{
  matrix dist;
  matrix pred;

  uint size = graph.size();
  dist.resize(size);
  pred.resize(size);

  /* Initialization */
  for(uint i = 0; i < size; ++i) {
    dist[i].resize(size);
    pred[i].resize(size);
    for(uint j = 0; j < size; ++j) {
      pred[i][j] = INF;
      if(i == j)
        dist[i][j] = 0;
      else
        dist[i][j] = INF;
    }
  }
/* Import graph */
 for(uint i = 0; i < size; ++i) {
   Edge *iter = graph[i];
   while(iter) {
     dist[i][iter->end] = iter->weight;
     pred[i][iter->end] = i;
     iter = iter->next;
   }
 }

 /* Algorithm main loop */
 for(uint k = 0; k < size; ++k) {
   for(uint i = 0; i < size; ++i) {
     for(uint j = 0; j < size; ++j) {
       if(dist[i][k] == INF || dist[k][j] == INF)
         continue;
       if(dist[i][j] > dist[i][k] + dist[k][j]) {
         dist[i][j] = dist[i][k] + dist[k][j];
         pred[i][j] = pred[k][j];
       }
     }
   }
 }

 // TODO: Do something with the results here
 // dist[i][j] is now the shortest distance between every two vertices, i and j,
 //            or INF if there is no such path
 // pred[i][j] == i if the shortest path goes directly from i to j
 //            Otherwise the shortest path from i to j is made up of the
 //            shortest path from i to pred[i][j] + the edge from pred[i][j] to j
}
```

Why does it work?

- Invariant: *after the k*th *iteration of the outer loop has finished,* $D^k[i,j]$ *contains the shortest path for all i and j.*
- Before the algorithm starts, $D^0[i,j]$ contains the shortest path from v_i to v_j without any intermediate vertices → invariant holds true

Why does it work?

- Assume the invariant is true after the k-1th iteration
- During the kth iteration, $D^k[i,j]$ will contain either $D^{k-1}[i,j]$ or $(D^{k-1}[i,k] + D^{k-1}[k,j])$, where $D^{k-1}[i,k]$ = shortest path from v_i to v_k, and $D^{k-1}[k,j]$ = shortest path from v_k to v_j
- Then $D^k[i,j]$ will choose the shorter path between the two best paths, so $D^k[i,j]$ must contain the shortest path from v_i to v_j $\rightarrow$ invariant holds true

Why does it work?

- By induction, the invariant must hold for every iteration of the outer loop
- After the outer loop has finished, $D^n[i,j]$ will contain the shortest path v_i to v_j using any vertices from v_1 to v_n $\rightarrow$ thus proves the correctness of the Floyd-Warshall algorithm

Runtime

```
Algorithm FloydWarshall(G)
  for  i ← 1  to  n  do
     for  j ← 1  to  n  do
       if  i == j  then
         $D^0[i,j]$ ← 0;
       else if  $(v_i,v_j)$ is an edge in G  then
         $D^0[i,j]$ ← $w(v_i,v_j)$; /* <-- weight function w */
       else
         $D^0[i,j]$ ← $+\infty$;
  for  k ← 1  to  n  do
     for  i ← 1  to  n  do
       for  j ← 1  to  n  do
         $D^k[i,j]$ ← min{$D^{k-1}[i,j]$, $D^{k-1}[i,k] + D^{k-1}[k,j]$};
  return matrix $D^n$
```

$O(|V|^3)$

When does it work/not work?

- Does **not** work when the graph has a negative cycle
- Works when it doesn’t contain any negative cycles, even when it has negative edges

Applications

- Shortest paths in directed graphs.
- Transitive closure of directed graphs.
- Inversion of real matrices (Gauss-Jordan algorithm).
- Optimal routing. In this application one is interested in finding the path with the maximum flow between two vertices. The edge weights represent fixed constraints on flow. Path weights represent bottlenecks.
- Testing whether an undirected graph is bipartite.

References

- Algorithm Design Foundation, Analysis, and Internet Examples. Michle T. Goodrich, Roberto Tamassia. ch 7.2.1
- Wikipedia: http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm
- Igor's notes from last year. Thank you ☺