University of Nevada, Las Vegas Computer Science 302 Fall 2018

Sparse Graphs and Sparse Directed Graphs and their Implementation

Most large graphs and directed graphs that occur in real life applications are sparse. Here are two extreme examples:

1. The graph whose vertices are all people, and where there is an edge between P and Q if P and Q have shaken hands.
2. The directed graph where vertices are all web pages, and there is an arc from x to y if x contains a link to y.

Neighborlist Implementation

We can implement a sparse graph, or a sparse directed graph, as an array of lists of edges. For each vertex v of G, let Neighbors[v] be the set of neighbors of v. That set should be implemented as a search structure. We find the edge (u, v) by searching Neighbors[u]. If G is directed, we implement either in-neighbors or out-neighbors of each vertex. If G is weighted, we store the weight of each edge as well as the ID of the neighbor.

Write $w(x, y)$ for the weight of the edge $\{x, y\}$, or the arc (x, y). Unless otherwise specified, weights are permitted to be negative.

Shortest Path Problem

There are three versions of the shortest path problem. Let $G = (V, E)$ be a weighted directed graph. We assume there is no negative cycle in G. In each case, "shortest" means least weight.

Let $n = |V|$ and $m = |E|$.

1. The single pair problem. Given vertices x, y, find the shortest path from x to y.
2. The single source problem. Given a vertex s, find the shortest path from s to each vertex x
3. The all-pairs problem. Find the shortest path from x to y for all choices of x and y. The all pairs problem can be solved by solving the single source problem n times, once for each choice of the source.

If G is not directed, we can make it directed by replacing each edge by two arcs, then ask the same questions.

Relaxation. Relaxation is the replacing of a weighted arc by a new arc of smaller weight, obtained by concatenating two existing arcs.

Floyd-Warshall Algorithm

This algorithm solves the all-pairs problem.

1. Initialize a 2-D array F of size n^2 by letting $F[x, x] = 0$ for all x, and $F[x, y] = w(x, y)$ if $(x, y) \in E$, and $F[x, y] = \infty$ otherwise.

2. For all $y \in V$: It is very important that the index of the outer loop correspond to the **middle** index in the relaxation step.

   For all $x \in V$:

   For all $z \in V$:

   If $(F[x, y] + F(y, z) < F[x.z])F[x, z] \leftarrow F[x, y] + F(y, z)$

   For each $x, y \in V$, the value of $F[x, y]$ will be the length (weight) of the shortest path from x to y.

The time complexity of the Floyd-Warshal algorithm is $O(n^3)$.

Bellman-Ford Algorithm

This algorithm solves the single source problem. Let s be the source vertex.

(a) Initialize an array F of size n by letting $F[s] = 0$ and $F[x] = \infty$ if $x \neq s$.

(b) repeat the following loop $n - 1$ times.
   For all $(x, y) \in E$:
   If $(F[x] + w(x, y) < F[y])F[y] \leftarrow F[x] + w(x, y)$

(c) $F[x]$ is the length of the shortest path from s to x.

The time complexity of the Bellman-Ford algorithm is $O(nm)$.

Dijkstra's Algorithm

This algorithm solves the single source problem, provided there is no negative weight. Let s be the source vertex.

(a) At each step of the algorithm, the vertices will be partitioned into three sets, the fully processed vertices, the partially processed vertices, and the unprocessed vertices. Call these FP, PP, and UP.

(b) Initialize an array F of size n by letting $F[s] = 0$ and $F[x] = \infty$ if $x \neq s$.

(c) Initialize FP $\leftarrow \emptyset$, PP $\leftarrow \{s\}$, and UP the set of all vertices except s.

(d) Repeat the following steps until PP $= \emptyset$.

   i. Let x be the member of PP such that $F[x]$ is minimum.

   ii. Move x from PP to FP.

iii. For each y which is an out-neighbor of x:

A. If $y \in$ FP do nothing.

B. $y \in$ PP, if $F[y] <= F[x] + w(x, y)$ do nothing. else, $F[y] \leftarrow F[x] + w[x, y)$.

C. If $y \in$ UP, move y to PP and let $F[y] \leftarrow F[x] + w(x, y)$

(e) $F[x]$ is the length of the shortest path from s to x.

The time complexity of Dijkstra's algorithm is $O(m + n \log n)$, obtained by using Fibonacci heaps. However, in the homework assignment, I have recommended using the structure I described in class; the time complexity is then $O(m \log n)$.

Johnson's Algorithm

This algorithm solves the all-pairs problem, and allows negative weight edges. It is faster than the Floyd-Warshall algorithm if the graph is sufficiently sparse, that is, if $m \log n < n^2$.

Johnson's algorithm uses the Bellman-Ford algorithm and Dijkstra's algorithm as subroutines.

(a) Introduce one new vertex, which we call s, and n new arcs, namely (s, x) for all $x \in V$. Set $w(s, x) = 0$ for all x.

(b) Use the Bellman-Ford algorithm to solve the single source problem, where s is the source. Let $H[x]$ be the minimum cost of any path from s to x. Note that $H[x]$ is either negative or zero for each x. Let $w'(x, y) = w(x, y) + H[y] - H[x]$ for all $(x, y) \in E$. We can show that w' is never negative.

(c) Solve the all pairs problem for the graph G, using the modified weights w' instead of w, by applying Dijkstra's algorithm n times. Let $F'[x, y]$ be the resulting values.

(d) For each x, y, let $F[x, y] = F'[x, y] + H[x] - H[y]$.

(e) For each $x, y \in V$, the value of $F[x, y]$ will be the length (weight) of the shortest path from x to y.

The time complexity of Johnson's algorithm is $O(nm \log n)$ using the version of the implementation of Dijkstra's I recommended. This is less than $O(n^3)$, for the Floyd Warshal Algorithm, if $m \log n$ is less than n^2.

If Fibonacci heaps are used, the time complexity of Dijkstra's algorithm is $O(nm + n^2 \log n)$ which is asymptotically equal to or better than that of the Floyd-Warshall algorithm. However, Floyd-Warshall is much simpler to implement.