This uses floyd’s algorithm to compute the minimal distance between New York City and Toronto. It works by iteratively improving the pathing between each city by trying every intermediate node. Then to extract the path, you just have to traverse the parent matrix. The non-bookkeeping code is below. Invoking this with New York City, Toronto will print the following:

New York City
Scranton
Binghamton
Syracuse
Rochester
Buffalo
Toronto

This path has a cost of 460mi. The included program should be invoked with make && ./submission input.txt

```
void floyd(cs375::WeightedDirectedGraph<std::string> graph,
std::string from,
           std::string to) {
  std::pair<bool, int> def{false, 0};
  std::vector<std::string> result;
  result.push_back(from);
  std::vector<std::vector<std::pair<bool, int> > >& D =
graph.ad_matrix_;
  std::vector<std::vector<std::string> > P{graph.idx_,
{graph.idx_, "NIL"}};
  for (auto i : graph) {
    for (auto j : graph) {
      if (i != j && graph.distance(i.first, j.first).first)
        P.at(i.second).at(j.second) = i.first;
    }
```

```
  }
  for (auto kr : graph) {
    for (auto ir : graph) {
      for (auto jr : graph) {
        int i = ir.second;
        int j = jr.second;
        auto dist = graph.distance(ir.first, kr.first);
        if (!dist.first) continue;
        auto dist_2 = graph.distance(kr.first, jr.first);
        if (!dist_2.first) continue;
        int dist_total = dist_2.second + dist.second;
        int min = D.at(i).at(j).second;
        if (dist_total < min || D.at(i).at(j).first == false) {
          min = dist_total;
          P[i][j] = P[kr.second][j];
        }
        D.at(i).at(j) = {true, min};
      }
    }
  }
  auto b = graph.index_of(from);
  auto e = graph.index_of(to);
  std::cout << from << std::endl;
  while (b != e) {
    std::cout << P.at(e).at(b) << std::endl;
    b = graph.index_of(P.at(e).at(b));
  }
}
```