CS302: Data Structures using C++

Fall 2020

Final Exam Preparation - Additional Exercises: **Traveling Salesman Problem #1**

Kostas Alexis

Problem Description: Consider 5 cities of interest, namely a) Reno, b) San Francisco, c) Salt Lake City, d) Seattle, and e) Las Vegas. Use information on the road network and derive the miles from one city to the other. Assume a fixed metric of Miles Per Gallon = 40 and derive the cost of each transition in terms of miles. Then on that basis, conduct the following:

- Create a graph with each of its vertices corresponding to the one of these cities and its edges being weighted by the associated miles for each trip. Note that if (and only if) to go from city A to B you must go through C then you must add one edge from A to C and one edge from C to B and there is no need to add an edge directly from A to B (equivalently – one edge of the additive weight).
- Solve the Traveling Salesman Problem such that the traveling salesman starts from Reno, visits all cities in the above list and returns to the origin. Solve this problem in the brutal force-way, i.e. by identifying all possible paths and picking the best (lower-cost).

Solution: We start by identifying the identifying the distances and paths between the five cities of our problem (Reno, San Francisco, Salt Lake City, Seattle, Las Vegas). We will use Google Maps to derive these distances which in turn allows us to build the cost matrix for the TSP problem at hand.

An instance of constructing the graph is shown below:

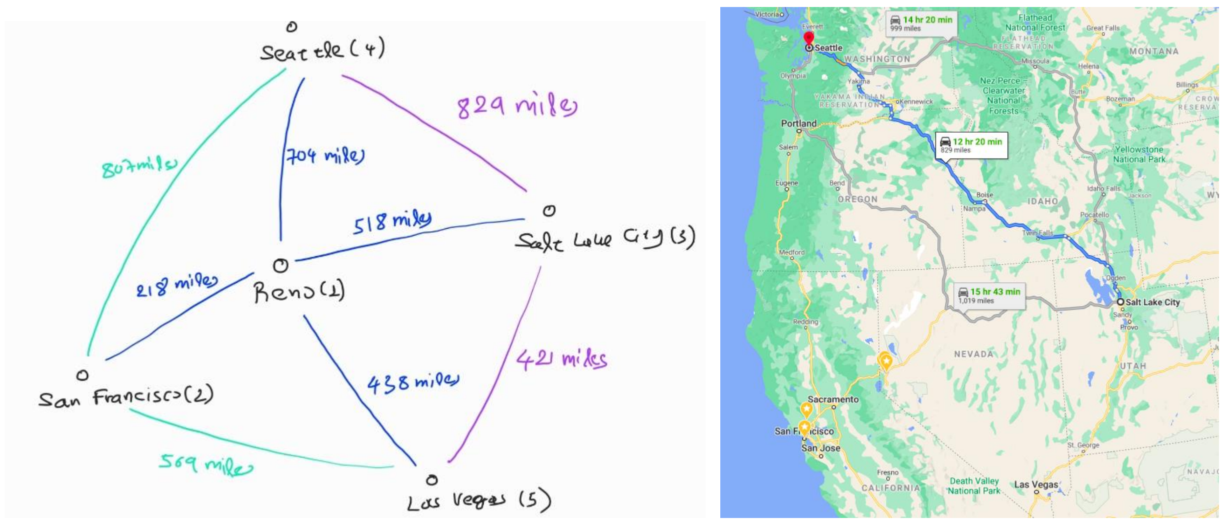


Figure 1. Constructing the graph of our problem.

The complete Cost Matrix takes the form (distances in miles):

	Reno	**San Francisco**	**Salt Lake City**	**Seattle**	**Las Vegas**
Reno	0	218.4	518.2	704.3	438.8
San Francisco	218.4	0	735.9	807.7	568.8
Salt Lake City	518.2	735.9	0	829.1	421.2
Seattle	704.3	807.7	829.1	0	1114.5
Las Vegas	438.8	568.8	421.2	1114.5	0

Before proceeding with coding the solution to the problem in a brute force manner we can do a small exercise of deriving some of the solutions manually by using the functionality on the maps API to place multiple destinations at an arbitrary order of visit. The images below present instances of indicative random solutions.

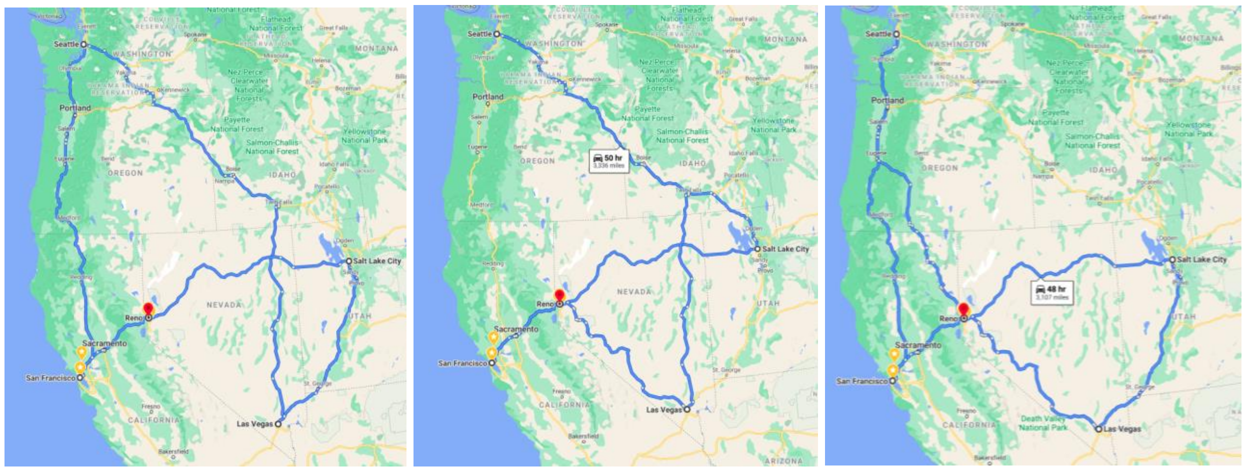


Figure 2. Indicative random, admissible but not optimal solutions to the TSP problem at hand.

Naturally, the optimal solution to the problem of starting from Reno, visiting San Francisco, Salt Lake City, Seattle and Las Vegas would look something like this:

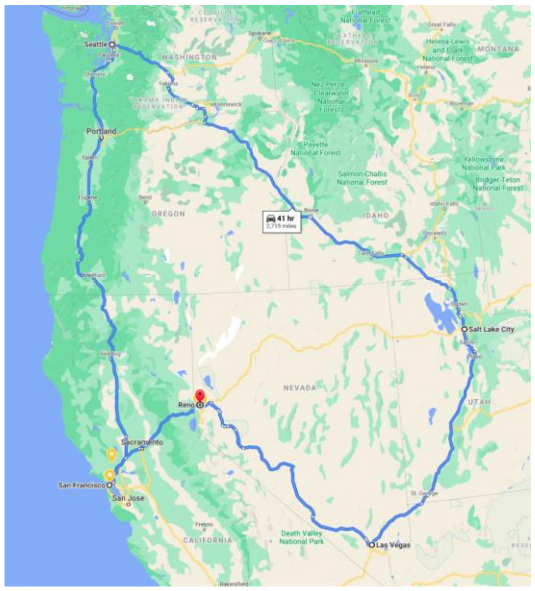


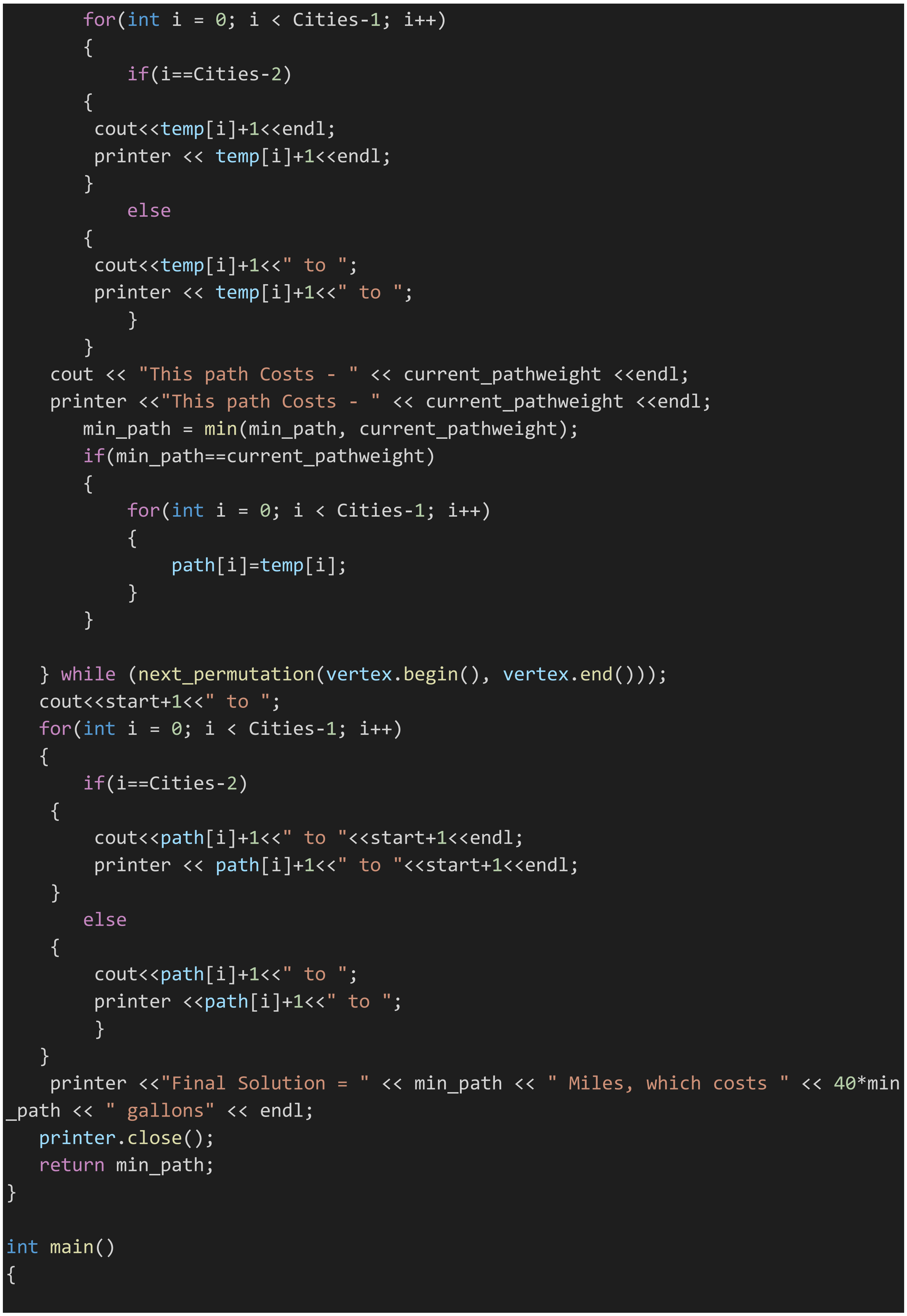
Figure 3. Optimal solution to the TSP problem at hand.

The solution to this problem in a brute-force manner is straightforward (albeit inefficient) and consists of the following steps:

1. Construct the cost matrix representing the graph of the problem.
2. Derive all permutations of paths in the graph that start from Reno, visit all other cities exactly once and return to Reno.
3. Select the one that has minimum cost in terms of gallons used (and thus since we assumed fixed number of gallons per mile, it is the same with the path that has the minimum distance cost).

The example solution below solves this problem in the manner discussed.

```
// Exercise on TSP - Final Exams Preparation
#include <iostream>
#include <vector>
#include <cstdint>
#include <climits>
#include <algorithm>
#include <fstream>
using namespace std;
#define Cities 5

float solution(float graph[][Cities], int start)
{

   ofstream printer;
   printer.open("output.txt");

   vector<int> vertex;
   int path[Cities],temp[Cities];
   for (int i = 0; i < Cities; i++)
       if (i != start)
           vertex.push_back(i);

   float min_path = INT_MAX;
   do {

       float current_pathweight = 0;

       int k = start;
       cout<<k+1<<" to ";
       for (int i = 0; i < vertex.size(); i++) {

           current_pathweight += graph[k][vertex[i]];
           k = vertex[i];
           temp[i]=k;
       }
       current_pathweight += graph[k][start];
```



```
       for(int i = 0; i < Cities-1; i++)
       {
           if(i==Cities-2)
       {
        cout<<temp[i]+1<<endl;
        printer << temp[i]+1<<endl;
       }
           else
       {
        cout<<temp[i]+1<<" to ";
        printer << temp[i]+1<<" to ";
           }
       }
    cout << "This path Costs - " << current_pathweight <<endl;
    printer <<"This path Costs - " << current_pathweight <<endl;
       min_path = min(min_path, current_pathweight);
       if(min_path==current_pathweight)
       {
           for(int i = 0; i < Cities-1; i++)
           {
               path[i]=temp[i];
           }
       }

   } while (next_permutation(vertex.begin(), vertex.end()));
   cout<<start+1<<" to ";
   for(int i = 0; i < Cities-1; i++)
   {
       if(i==Cities-2)
    {
        cout<<path[i]+1<<" to "<<start+1<<endl;
        printer << path[i]+1<<" to "<<start+1<<endl;
    }
       else
    {
        cout<<path[i]+1<<" to ";
        printer <<path[i]+1<<" to ";
        }
   }
    printer <<"Final Solution = " << min_path << " Miles, which costs " << 40*min
_path << " gallons" << endl;
   printer.close();
   return min_path;
}

int main()
{
```

```
//0=Reno 1=San Francisco 2=Salt Lake City 3=Seattle 4=Las Vegas
   float graph[][Cities] = { { 0, 218.4, 518.2, 704.3, 438.8 },{ 218.4, 0, 735.9,
 807.7, 568.8 },{ 518.2, 735.9, 0, 829.1, 421.2 },{ 704.3, 807.7, 829.1, 0, 1114.
5 },{438.8, 568.8, 421.2, 1114.5, 0 }};
   int start = 0;
   float sol = solution(graph, start);
   cout << "Final Solution = " << sol << " Miles, which costs " << 40*sol << " ga
llons" << endl;

   return 0;
}
```

If we simply prepare the following makefile

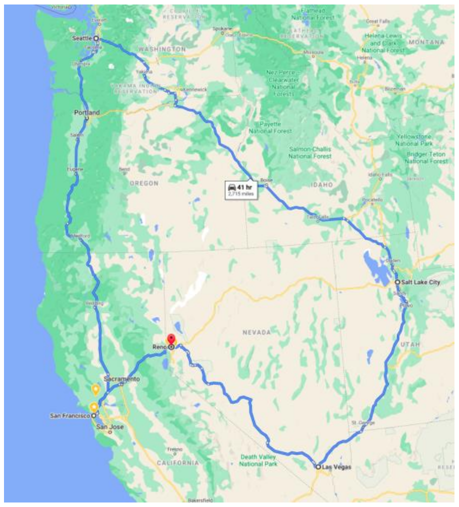
```
run:
    g++ main.cpp -o tsp_example


clean:
    rm output.txt tsp_example
```

and make our project then we derive the "tsp_example" executable which in turn returns the following "output.txt" file with the optimal solution being (1: Reno, 2: San Francisco, 3: Salt Lake City, 4: Seattle, 5: Las Vegas) starting from Reno (1):

```
5 to 3 to 4 to 2 to 1
Final Solution = 2715.2 Miles, which costs 108608 gallons
```

This solution is the one we also visualized in our map intuitively:

Optimal path: Reno > Las Vegas > Salt Lake City > Seattle > San Francisco > Reno



Follow-up exercise: Code the solution to this problem using the Nearest-Neighbor-Heuristic discussed in class and present its solution.