Chapter 44: Travelling Salesman

Section 44.1: Brute Force Algorithm

A path through every vertex exactly once is the same as ordering the vertex in some way. Thus, to calculate the minimum cost of travelling through every vertex exactly once, we can brute force every single one of the `N!` permutations of the numbers from `1` to `N`.

Psuedocode

```
minimum = INF
for all permutations P

    current = 0

    for i from 0 to N-2
        current = current + cost[P[i]][P[i+1]]  <- Add the cost of going from 1 vertex to the next

    current = current + cost[P[N-1]][P[0]]      <- Add the cost of going from last vertex to the
first

    if current < minimum                        <- Update minimum if necessary
        minimum = current

output minimum
```

Time Complexity

There are `N!` permutations to go through and the cost of each path is calculated in `O(N)`, thus this algorithm takes `O(N * N!)` time to output the exact answer.

Section 44.2: Dynamic Programming Algorithm

Notice that if we consider the path (in order):

```
(1,2,3,4,6,0,5,7)
```

and the path

```
(1,2,3,5,0,6,7,4)
```

The cost of going from vertex `1` to vertex `2` to vertex `3` remains the same, so why must it be recalculated? This result can be saved for later use.

Let `dp[bitmask][vertex]` represent the minimum cost of travelling through all the vertices whose corresponding bit in `bitmask` is set to `1` ending at `vertex`. For example:

```
dp[12][2]

   12   =   1 1 0 0
            ^ ^
vertices:   3 2 1 0
```

Since `12` represents `1100` in binary, `dp[12][2]` represents going through vertices `2` and `3` in the graph with the path ending at vertex 2.

Thus we can have the following algorithm (C++ implementation):

```
int cost[N][N];                         //Adjust the value of N if needed
int memo[1 << N][N];                    //Set everything here to -1
int TSP(int bitmask, int pos){
    int cost = INF;
    if (bitmask == ((1 << N) - 1)){      //All vertices have been explored
        return cost[pos][0];             //Cost to go back
    }
    if (memo[bitmask][pos] != -1){       //If this has already been computed
        return memo[bitmask][pos];       //Just return the value, no need to recompute
    }
    for (int i = 0; i < N; ++i){         //For every vertex
        if ((bitmask & (1 << i)) == 0){  //If the vertex has not been visited
            cost = min(cost,TSP(bitmask | (1 << i) , i) + cost[pos][i]);   //Visit the vertex
        }
    }
    memo[bitmask][pos] = cost;           //Save the result
    return cost;
}
//Call TSP(1,0)
```

This line may be a little confusing, so lets go through it slowly:

```
cost = min(cost,TSP(bitmask | (1 << i) , i) + cost[pos][i]);
```

Here, `bitmask | (1 << i)` sets the ith bit of `bitmask` to 1, which represents that the ith vertex has been visited. The `i` after the comma represents the new `pos` in that function call, which represents the new "last" vertex. `cost[pos][i]` is to add the cost of travelling from vertex `pos` to vertex `i`.

Thus, this line is to update the value of `cost` to the minimum possible value of travelling to every other vertex that has not been visited yet.

Time Complexity

The function `TSP(bitmask,pos)` has `2^N` values for `bitmask` and `N` values for `pos`. Each function takes `O(N)` time to run (the **`for`** loop). Thus this implementation takes `O(N^2 * 2^N)` time to output the exact answer.