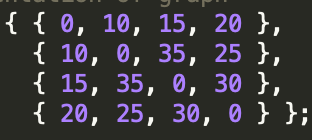
**Lab 3**

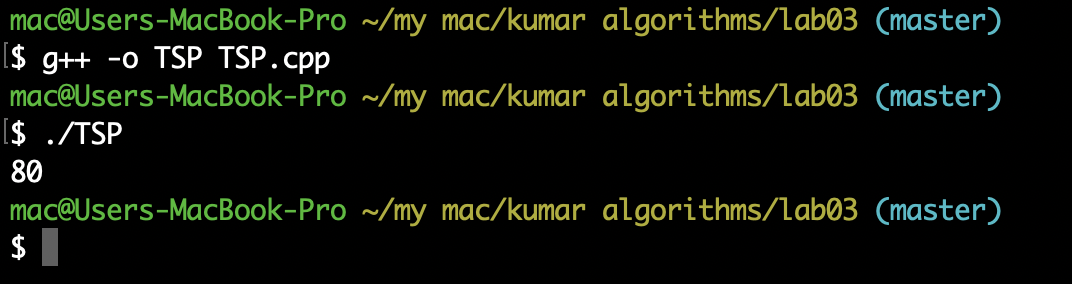
**Isa Dzhumabaev**

**1. Compile/Run the code given in the previous slide**

**Input:**



**Result:**

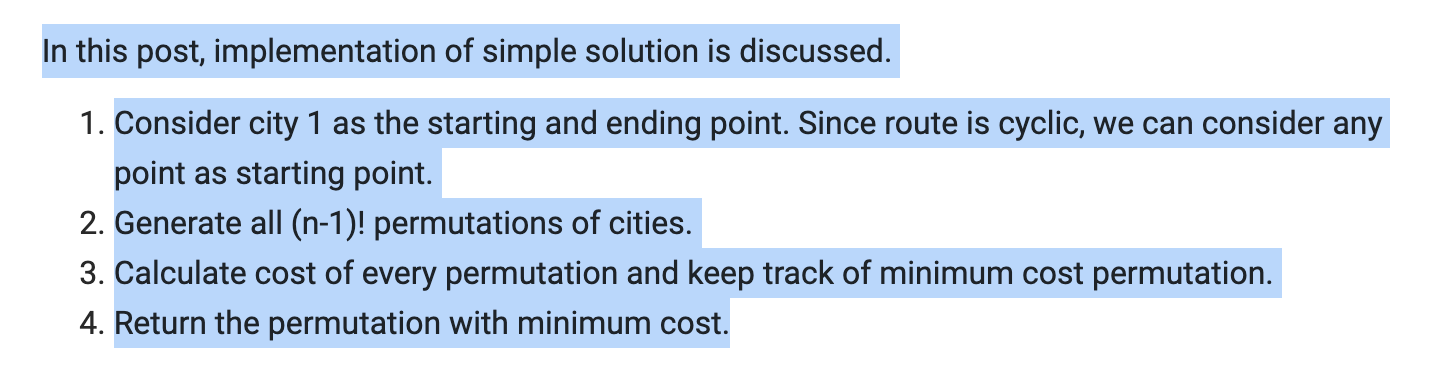


**2. Explain how does it use the greedy algorithm**

I don’t think that this exact implementation uses greedy algorithm. Greedy algorithm is based on making localy optimal decision assuming that this will lead to optimal solution.

I think that this exact implementation actually uses brute-force as it tries to consider all possible permutations, count their cost and take minimal. It is said so it article.

**Algorithm idea used in the article:**



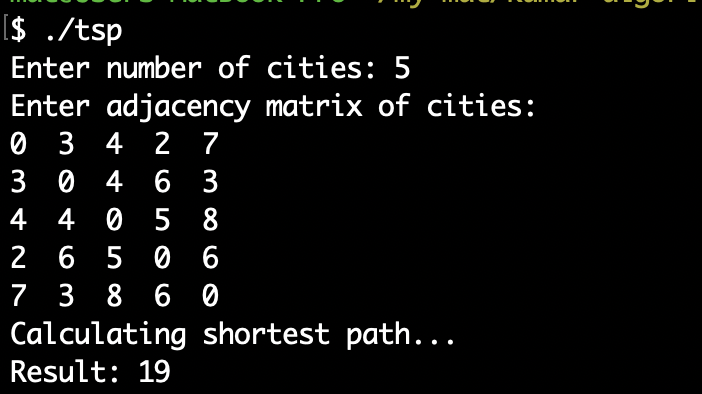
Greedy TSP solution takes minimal cost on every particular vertex and this does not guaranty the best solution.

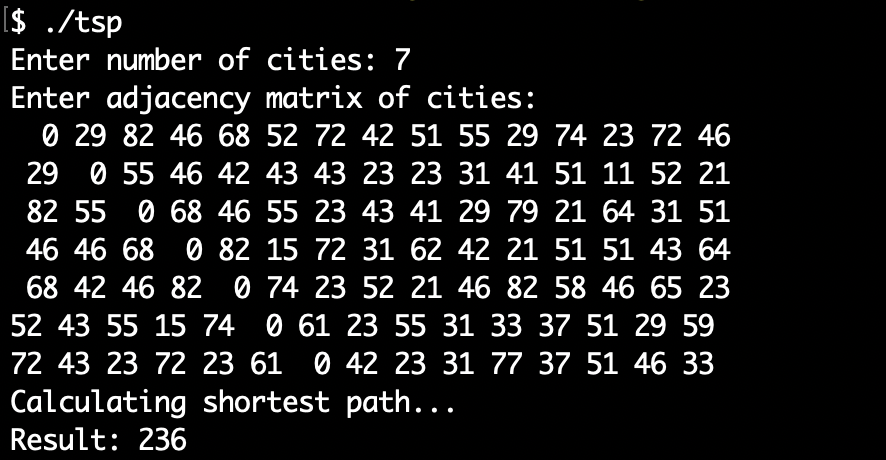
**3. Review the datasets given in the second link**

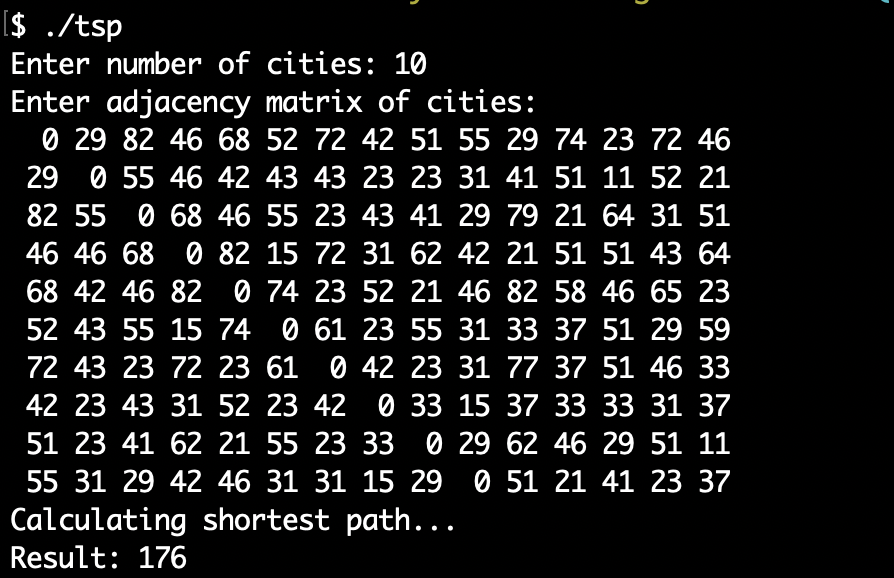
I wanted to test this algorithm with 26, 17, 15 and 5 cities form this link: <https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html>.

But this algorithm was too slow even on 15 cities so I decided not to use 15, 17 and 26. Instead I just used 5 cities and two reduced versions of input with 15 cities.

**Results of tests:**







**4. Modify/Use both programs to use larger inputs**

**Source code of modified version to take larger inputs:**

#include <bits/stdc++.h>

#include <time.h>

using namespace std;

int travllingSalesmanProblem(int\*\* graph, int s, int n)

{

vector<int> vertex;

**for** (int i = 0; i < n; i++)

**if** (i != s)

vertex.push\_back(i);

int min\_path = INT\_MAX;

**do** {

int current\_pathweight = 0;

int k = s;

**for** (int i = 0; i < vertex.size(); i++) {

current\_pathweight += graph[k][vertex[i]];

k = vertex[i];

}

current\_pathweight += graph[k][s];

min\_path = min(min\_path, current\_pathweight);

} **while** (next\_permutation(vertex.begin(), vertex.end()));

**return** min\_path;

}

int main()

{

int n;

printf("Enter number of cities: ");

scanf("%d", &n);

printf("Enter adjacency matrix of cities: **\n**");

int\*\* graph;

graph = new int\*[n];

**for** (int i = 0; i < n; ++i) {

graph[i] = new int[n];

**for** (int j = 0; j < n; ++j) {

scanf("%d", &graph[i][j]);

}

}

printf("**\n**Calculating shortest path...**\n**");

int s = 0;

int result = travllingSalesmanProblem(graph, s, n);

printf("Result: %d**\n**", result);

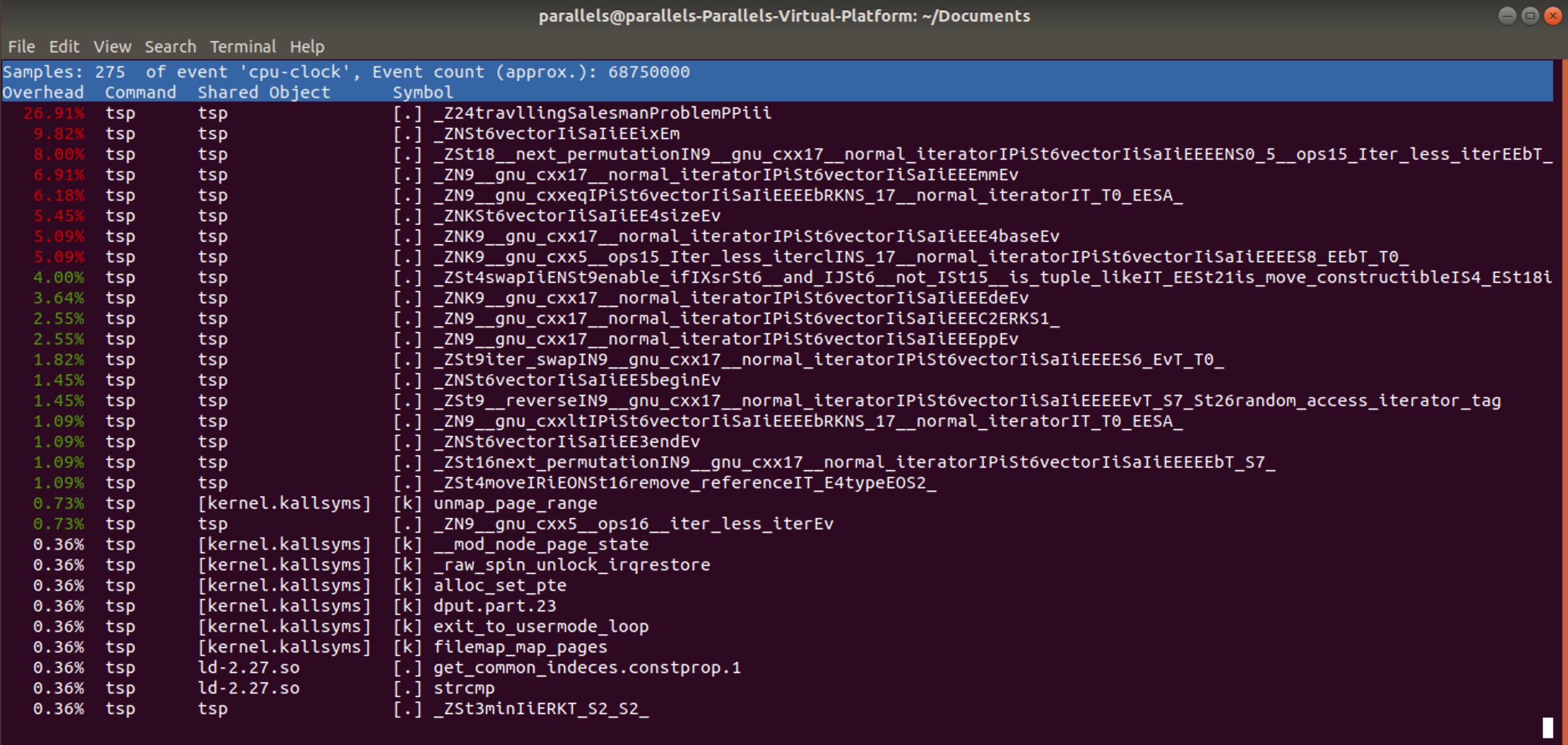
**return** 0;

}

**5. Run operf/opreport on both with larger inputs**

Perf was used on VM using Parallels.

We can observe that **travllingSalesmanProblem()** function takes most time as expected.

****