**Term Assignment 1**   
**Objective**  
Analyze and compare the performance of two all-pairs shortest path algorithms:  
- Johnson’s Algorithm  
- Floyd–Warshall Algorithm  
  
**Evaluate them on two graphs with 50 nodes each:**  
1. Sparse Graph: 50 nodes, 100–150 randomly generated edges.  
2. Dense Graph: 50 nodes, complete graph (1225 edges).  
  
**Tasks**  
1. Graph Generation  
- Generate a sparse weighted directed graph (50 nodes, 100–150 edges).  
- Generate a complete weighted directed graph (50 nodes).  
- Assign random positive weights (1–100).  
  
2. Algorithm Implementation  
- Implement Johnson’s Algorithm.  
- Implement Floyd–Warshall Algorithm.  
  
3. Performance Measurement  
For each run, measure:  
- Execution time (s)  
- Estimated energy consumption (J)  
 Energy (J) = [CPU's approximate average power](https://www.intel.com/content/www/us/en/products/sku/140642/intel-core-i78700-processor-12m-cache-up-to-4-60-ghz-includes-intel-optane-memory-16gb/specifications.html?wapkw=i78700) x Execution Time (s)  
- Peak memory usage (KB)  
- Estimated CO₂ emissions (Bangladesh)  
 Energy (kWh) = Energy (J) / 3,600,000  
 CO₂ (kg) = Energy (kWh) x 0.62  
  
**Deliverables**- Source code implementing graph generation, algorithms, and performance measurement.  
- Filled results table and discussion.

| **Algorithm** | **Graph Type** | **Nodes** | **Edges** | **Execution Time (s)** | **Energy (J)** | **Peak Memory (KB)** | **CO₂ Emissions (kg, BD)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Johnson | Sparse | 50 | 125 | 0.534212 | 34.72 | 1500 | 0.000006 |
| Floyd–Warshall | 50 | 125 | 0.789654 | 51.32 | 1800 | 0.000009 |
| Johnson | Dense | 50 | 1225 | 3.214567 | 208.95 | 2500 | 0.000036 |
| Floyd–Warshall | 50 | 1225 | 5.876543 | 381.97 | 3200 | 0.000066 |

Note: Numbers are illustrative. Students will fill in actual measured values.  
  
Discussion / Conclusion  
- Sparse Graph: Johnson’s Algorithm is faster, uses less memory, and consumes less energy. CO₂ emissions are lower. Recommended for sparse graphs to reduce carbon footprint.  
- Dense Graph: Johnson’s Algorithm is still slightly better in time and energy efficiency, but memory use is closer to Floyd–Warshall. Recommended for dense graphs as well.  
- Overall: Johnson’s Algorithm is generally more carbon-efficient, especially for sparse graphs. Floyd–Warshall is simpler but less energy-efficient and emits more CO₂ as graph density increases.

**Code for Performance Measurement**

#include <windows.h>

#include <psapi.h>

#include <stdio.h>

#pragma comment(lib, "Psapi.lib") // Needed for MSVC linking

// Replace with your CPU's approximate average power (Watts)

#define CPU\_POWER\_WATTS 65.0

// Bangladesh grid emission factor (kg CO₂ / kWh)

#define BD\_EMISSION\_FACTOR 0.62

int main() {

LARGE\_INTEGER freq, start, end;

QueryPerformanceFrequency(&freq);

QueryPerformanceCounter(&start);

// -------- Your algorithm here --------

volatile long long sum = 0;

for (long long i = 0; i < 100000000; i++) {

sum += i;

}

int a[500000]; // ~2 MB on stack

for (long long i = 0; i < 100000000; i++) {

sum += i;

}

// -------------------------------------

QueryPerformanceCounter(&end);

double elapsed = (double)(end.QuadPart - start.QuadPart) / freq.QuadPart;

double energy = CPU\_POWER\_WATTS \* elapsed; // Joules

double energy\_kWh = energy / 3.6e6; // kWh

double co2 = energy\_kWh \* BD\_EMISSION\_FACTOR; // kg CO₂

// ----- Memory usage -----

PROCESS\_MEMORY\_COUNTERS\_EX pmc;

if (GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS\_MEMORY\_COUNTERS\*)&pmc, sizeof(pmc))) {

SIZE\_T peakMemUsed = pmc.PeakWorkingSetSize; // Peak RAM usage

printf("Execution time: %.6f seconds\n", elapsed);

printf("Estimated energy consumption: %.2f Joules\n", energy);

printf("Peak Memory Usage: %zu KB\n", peakMemUsed / 1024);

printf("Estimated CO2 emissions (Bangladesh): %.8f kg\n", co2);

} else {

printf("Failed to get memory info.\n");

}

return 0;

}