

- **GitHub** <https://github.com/JanPastorek/1-AIN-413-22-Graphs>
 - **New Home project is already on Moodle!**
-

Problem -1. [Previous week - TSP]

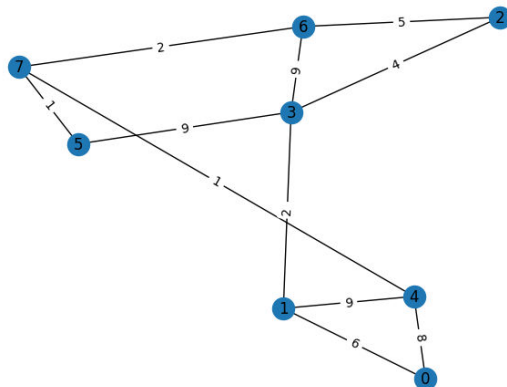
Stanoyevitch 9.2.35 (a) Fix a positive integer n . Construct a TSP for which the nearest neighbor heuristic produces a tour of weight greater than 10^n times the weight of an optimal tour. (b) Repeat Part (a) for the nearest insertion method. (c) Repeat Part (a) for the furthest insertion method. (d) Repeat Part (a) for the cheapest insertion method.

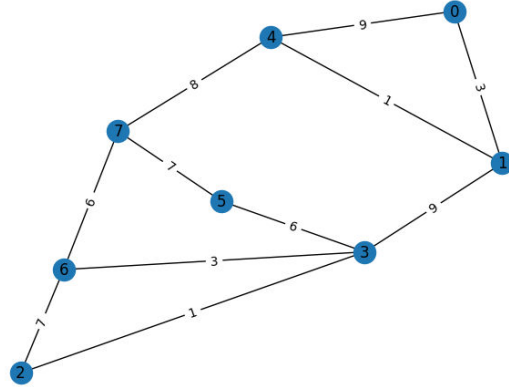
- program these methods

Stanoyevitch 9.2.10 (Random TSP Generator) Write a program that will randomly generate Euclidean distance TSPs and that has the following syntax: $[W, \text{Points}] = \text{TSPRand}(n, d)$. Here, the first input n is an integer greater than 1, and the second optional input d is a positive number (default value 10). The program will generate n points randomly distributed in the square $0 \leq x, y \leq d$. These points will be the vertices of the complete graph for the TSP. The first output variable is the $n \times n$ edge-weight matrix W whose entries give the Euclidean distance between corresponding pairs of points. The second output variable, Points is an $n \times 2$ matrix whose rows give the x - and y -coordinates of the vertices.

Problem 0. [Any questions?]

Is there anything unclear from the lectures or about the home project?





Problem 1. [Simulating algorithms]

- Simulate shortest-path algorithms on the graphs above.

Dijkstra algorithm

```
def dijkstra(v1):  
    # in array D are current distances of all vertices from v1  
    D[all vertices] = inf      # initialize to infinity & and continuously decrease  
    D[v1] = 0  
    queue = priority queue for all vertices using keys of D  
    while queue:  
        v1 = queue.remove_min()  
        for v2 in neighbours(v1):  
            if v2 je v queue:      # v2 was not yet processed  
                if D[v1] + weight(v1, v2) < D[v2]: # here is why weights can NOT be negative  
                    D[v2] = D[v1] + weight(v1, v2)  
                    change the key D[v2] in queue for v2  
    return D      # array of distances of all vertices from v1
```

Johnson algorithm

```
JOHNSON( $G, w$ )
1  compute  $G'$ , where  $G'.V = G.V \cup \{s\}$ ,
    $G'.E = G.E \cup \{(s, v) : v \in G.V\}$ , and
    $w(s, v) = 0$  for all  $v \in G.V$ 
2  if BELLMAN-FORD( $G', w, s$ ) == FALSE
3    print "the input graph contains a negative-weight cycle"
4  else for each vertex  $v \in G'.V$ 
5    set  $h(v)$  to the value of  $\delta(s, v)$ 
   computed by the Bellman-Ford algorithm
6  for each edge  $(u, v) \in G'.E$ 
7     $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ 
8  let  $D = (d_{uv})$  be a new  $n \times n$  matrix
9  for each vertex  $u \in G.V$ 
10    run DIJKSTRA( $G, \hat{w}, u$ ) to compute  $\hat{\delta}(u, v)$  for all  $v \in G.V$ 
11    for each vertex  $v \in G.V$ 
12       $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$ 
13  return  $D$ 
```

Idea of the algorithm:

Step 1: adding a base vertex

Step 2: Reweighting the edges using Bellman algorithm to positive -
if a certain path is the shortest path between those vertices before reweighting,
it must also be the shortest path between those vertices after reweighting

**Step 3: finding all pairs shortest path on the reweighted Graph, and
reweighting back**

- (b) (Cormen) 25.3-2 What is the purpose of adding the new vertex s to V , yielding V' ?

Problem 2. [Choice of algorithms]

- (a) Apart from finding shortest paths, what can you use these algorithms for?
- (b) Decide which algorithm would you use for these graphs? Make simulations on random graphs of that many vertices and edges and different algorithms.

1. **Small Sparse Graph:**

- Description: A graph with 20 vertices and 30 edges.
- Characteristics: Sparse connectivity, no negative weights, no cycles.

2. **Small Dense Graph:**

- Description: A graph with 20 vertices and 190 edges (a nearly complete graph).

-
- Characteristics: Dense connectivity, no negative weights, possible cycles.
 - 3. **Medium Graph with Negative Weights:**
 - Description: A graph with 100 vertices and 300 edges, including some negative weights.
 - Characteristics: Medium size, sparse, negative weights, possible cycles.
 - 4. **Large Sparse Graph:**
 - Description: A graph with 1000 vertices and 2000 edges.
 - Characteristics: Large, sparse, no negative weights, possible cycles.
 - 5. **Graph with Many Negative Cycles:**
 - Description: A graph with 50 vertices and 120 edges, several negative cycles.
 - Characteristics: Medium size, negative cycles.
 - 6. **Directed Acyclic Graph (DAG):**
 - Description: A DAG with 200 vertices and 400 edges.
 - Characteristics: No cycles, directed edges, no negative weights.
 - 7. **Weighted Tree:**
 - Description: A tree graph with 100 vertices, 99 edges.
 - Characteristics: Tree structure (acyclic), potentially negative weights.
 - 8. **Large Dense Graph:**
 - Description: A graph with 1000 vertices and 499,500 edges (a nearly complete graph).
 - Characteristics: Large, dense, no negative weights, many cycles.
 - 9. **Random Graph with Negative Weights:**
 - Description: A graph with 500 vertices and 1000 edges, random distribution of negative weights.
 - Characteristics: Medium size, sparse, negative weights, possible cycles.
 - 10. **DiGraph:**
 - Description: A DiGraph with 500 vertices and 1000 edges

Runtime

Bellman-Ford $O(|V| \cdot |E|)$ **Dijkstra's (with list)** $O(|V|^2)$ **Dijkstra's (with heap)** $O(|V| \log_2(|V|))$ **Floyd-Warshall** $O(|V|^3)$ **Johnson's** $> O(|E| \cdot |V| + |V|^2 \cdot \log_2(|V|)) = O(\text{Bellman-Ford} + V \cdot \text{Dijkstra})$

Hints:

- For graphs with negative weight edges, the single source shortest path problem needs **Bellman-Ford** to succeed. Otherwise **Dijkstra**.
- For dense graphs and the all-pairs problem, **Floyd-Warshall** should be used. Negative weights are allowed.
- For sparse graphs and the all-pairs problem, it might be obvious to use **Johnson's** algorithm.
 - However, if there are no negative edge weights, then it is actually better to use **Dijkstra's** algorithm with binary heaps in the implementation. Running Dijkstra's from each vertex will yield a better result.

- **Bellman and Johnson works on directed only.** So either you have to change your graph or choose a diff. algorithm.