Advanced network algorithms, random graph models

You are given four networks in Pajek format.

- Tiny toy network for testing (toy.net)
- Zachary karate club network (karate_club.net)
- iMDB actors collaboration network (collaboration_imdb.net)
- A small part of Google web graph (www_google.net)

I. Average node distance and network diameter

1. **(discuss)** Study the following algorithm for computing the distances between the nodes $\{d\}$ by level order link traversal. Does the algorithm implement breadth-first or depth-first search? Why? What is the time complexity of the algorithm?

```
input   graph G
output  network distances {D}
1:  {D} ← empty list
2:  for nodes i ∈ N do
3:      {D}.add(distances(G, i))
4:  return {D}
```

```
input   graph G, node i
output  directed distances D
5:  ...
6:  for successors j ∈ Γ_i^out do
7:      ...
```

```
input   graph G, node i
output  undirected distances D
1:  D ← empty array
2:  Q ← empty queue
3:  D[Q.add(i)] ← 0
4:  while not Q empty do
5:      i ← Q.remove()
6:      for neighbors j ∈ Γ_i do
7:          if D[j] undefined then
8:              D[j] ← D[i] + 1
9:              Q.add(j)
10: return D
```

2. **(code)** Implement the algorithm and compute the average distance between the nodes $\langle d \rangle$ and the maximum distance or diameter $d_{\max}$ of *smaller* networks. Are the results expected?

3. **(discuss)** How is the algorithm different from the famous Dijkstra algorithm? In which case you would have to use the Dijkstra algorithm?

4. **(discuss)** How could you speed up the algorithm to *only* approximate $\langle d \rangle$ and $d_{\max}$?

II. Average node clustering coefficient

1. **(discuss)** Study the following algorithm for computing the node clustering coefficients $\{C\}$ by link triad

counting. Why does the algorithm count triads over the links and not over the nodes? What is the time complexity of the algorithm?

```
input   graph G
output  average clustering ⟨C⟩
1:  ⟨C⟩ ← 0
2:  for nodes i ∈ N do
3:      ⟨C⟩ ← clustering(G, i)/n
4:  return ⟨C⟩
```

```
input   graph G, node i
output  node clustering C
1:  if k_i ≤ 1 then
2:      return 0
3:  return triads(G, i) · 2/(k_i^2 − k_i)
```

```
input   graph G, node i
output  node triads t
1:  t ← 0
2:  for neighbors j ∈ Γ_i do
3:      if |Γ_i| ≤ |Γ_j| then
4:          t ← t + triads(G, i, j)/2
5:      else
6:          t ← t + triads(G, j, i)/2
7:  return t
```

```
input   graph G, link i, j
output  link triads t
1:  t ← 0
2:  for neighbors k ∈ Γ_i do
3:      if k ∈ Γ_j then
4:          t ← t + 1
5:  return t
```

2. **(homework)** Implement the algorithm and compute the average node clustering coefficient $\langle C \rangle$ of all four networks. Are the results expected?

3. **(discuss)** What kind of network representation is required by the algorithm?

III. Erdös-Rényi random graphs and link indexing

1. **(discuss)** Study the following two algorithms for generating Erdös-Rényi random graphs $G(n, m)$ with and without link indexing $\binom{i}{2} + j, i > j$. What is the main difference between the algorithms? What is the time complexity of the algorithms?

```
input   nodes n, links m
output  simple random G
1:  H ← empty set
2:  G ← n isolated nodes
3:  while not G has m links do
4:      h ← {0, . . . , (n^2 − n)/2 − 1}.random()
5:      if H.add(h) then
6:          i ← 1 + ⌊−0.5 + √(0.25 + 2h)⌋
7:          add link between i and h − (i^2 − i)/2
8:  return G
```

```
input   nodes n, links m
output  random multi G
1:  G ← n isolated nodes
2:  while not G has m links do
3:      i, j ← {0, . . . , n − 1}.random()
4:      if i ≠ j then
5:          add link between i and j
6:  return G
```

2. **(code)** Implement one of the algorithms and generate Erdös-Rényi random graphs corresponding to all four networks, and compute their S, $\langle d \rangle$ and $\langle C \rangle$. Are the results expected?

IV. Configuration model graphs and link rewiring

1. **(discuss)** Study the following two algorithms for generating configuration model graphs $G(\{k\})$ with either link rewiring or stub matching. What is the main difference between the algorithms? What is the time complexity of the algorithms?

```
input  simple links L
output configuration simple G
H ← empty set
for links {i, j} ∈ L do
    H.add(h_ij)
while not links L rewired do
    {i, j}, {s, t} ← L.random()   ▷ removes links
    if H.contains(h_it or h_sj) or i = t or s = j then
        L.add({i, j})
        L.add({s, t})
    else
        L.add({i, t}) H.add(h_it) H.remove(h_ij)
        L.add({s, j}) H.add(h_sj) H.remove(h_st)
return G on links L
```

```
input  nodes n, degrees {k}
output configuration pseudo G
Q ← empty queue
G ← n isolated nodes
for nodes i ∈ N do
    for k_i times do
        Q.add(i)
while not Q empty do
    i, j ← Q.random()   ▷ removes nodes
    add link between i and j
return G
```

2. **(code)** Implement one of the algorithms and generate configuration model graphs corresponding to all four networks, and compute their S, $\langle d \rangle$ and $\langle C \rangle$. Are the results expected?