Kosen Club CP 勉強会

GRAPH-BFS, DIJKSTRA, BELLMAN-FORD, FLOYD-WARSHALL

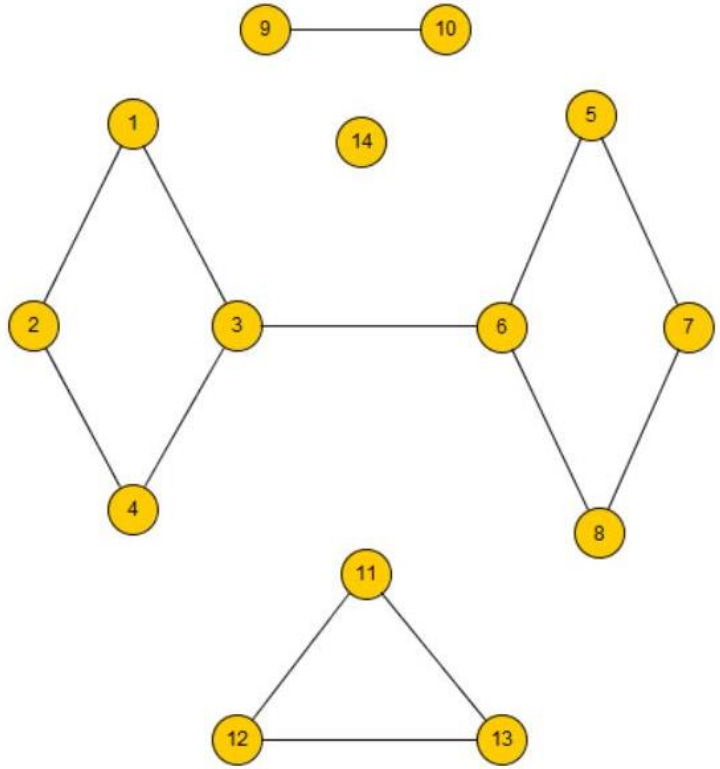
1.Shortest Path Algorithms

Algorithms	BFS	Dijksra	Bellman-Ford	Floyd-Warshall
Scope	1 to all	1 to all	1 to all	All to all
Weight	Only uniform	Non-negative	All	All
Implementation	Medium	Hard	Medium	Easy
Time complexity	O(N)	O(E*logV)	O(V*E)	O(V^3)
Space complexity	O(E)	O(E)	O(E)	O(V^2)

2.BFS

[Breadth-first search – Wikipedia]

```
 1  procedure BFS(G, root) is
 2      let Q be a queue
 3      label root as explored
 4      Q.enqueue(root)
 5      while Q is not empty do
 6          v := Q.dequeue()
 7          if v is the goal then
 8              return v
 9          for all edges from v to w in G.adjacentEdges(v) do
10              if w is not labeled as explored then
11                  label w as explored
12                  Q.enqueue(w)
```



3.Dijkstra

Dijkstra's algorithm - Wikipedia

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                           // Initialization
3
4      create vertex priority queue Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                 // Unknown distance from source to v
9              prev[v] ← UNDEFINED                // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                      // The main loop
15         u ← Q.extract_min()                    // Remove and return best vertex
16         for each neighbor v of u:              // only v that are still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist, prev
```

4.Bellman-Ford

[Bellman–Ford algorithm - Wikipedia]

```
distance := list of size n
predecessor := list of size n

// Step 1: initialize graph
for each vertex v in vertices do

    distance[v] := inf             // Initialize the distanc
    predecessor[v] := null         // And having a null pred

distance[source] := 0              // The distance from the

// Step 2: relax edges repeatedly

repeat |V|-1 times:
     for each edge (u, v) with weight w in edges do
         if distance[u] + w < distance[v] then
             distance[v] := distance[u] + w
             predecessor[v] := u

// Step 3: check for negative-weight cycles
for each edge (u, v) with weight w in edges do
    if distance[u] + w < distance[v] then
        error "Graph contains a negative-weight cycle"

return distance, predecessor
```

5.Floyd-Warshall

[Floyd–Warshall algorithm - Wikipedia]

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```