Faculty of Mathematics and Physics
Charles University in Prague
8th April 2016

C# Made Easy!

Programming II

Lab 06 –Theme Hospital Lite
Part 2 – The Simulation

Lab 06

Outline

1. No Test
2. Revisiting Workshop 05
3. Assignment 06+07
   - The Simulation

Test 06

No Test ;)

Find the test here (no-ads):

https://goo.gl/FkyFWX

`O vs 0, i vs. l vs. 1`

Permanent link:

https://docs.google.com/forms/d/1F2WFyiBkjhEJxwwxaohPXESHFIW7O7sOmX59HGmaWEk/viewform

Time for the test:

3 mins

Topic

Theme Hospital Lite

Have you tried playing it already? Fun guaranteed!

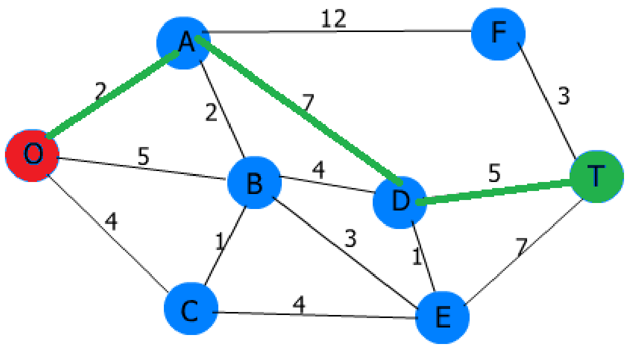
Topic
Navigation

- For the input of graph, rooms, etc. check slides from previous Lab 05 !

- Now let us revisit some problems from Lab 05
  1. Dijkstra's Algorithm
  2. Heap
  3. Dictionary
  4. WalkLink vs. LiftLink … where to hold GetCapacity()
  5. Debugging + ToString()
  6. Questions?

Dijkstra’s Algorithm

Dijkstra's Algorithm
Explained



- *Input*
  - Un/oriented (weighted) graph
- *Task*
  - Find the shortest path between two given nodes O, T within the graph
- *Output*
  - Valid path within the graph between nodes O and T that is the shortest given graph's weights or "NULL" if no such a path exists

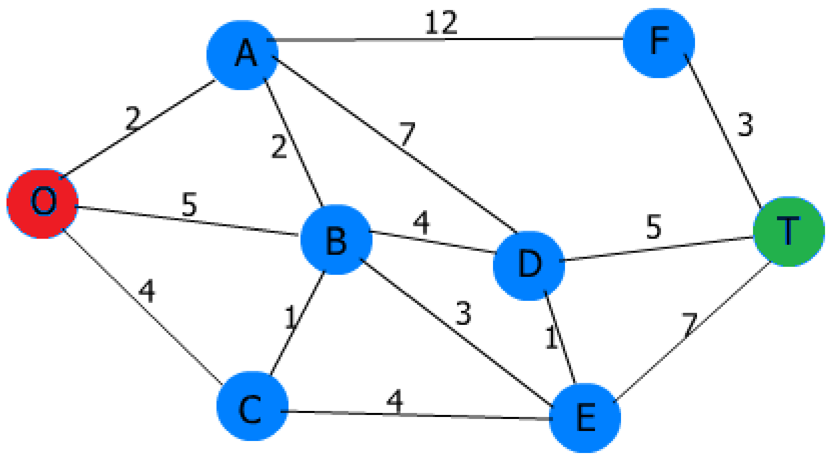
Dijkstra’s Algorithm

Pseudocode

```
Dijkstra(start, end):
      openList   = [start]                // priority queue
      closedList = []                     // set
      pathCost   = {start = 0}            // map: node => int
      pathParent = {start = null}         // map: node => node

      while (openList is not empty) {     // until we have nodes to search through
         node = openList.Dequeue()        // get the node with “current shortest path”
         if (node == end)                 // if it is our TARGET
          return ReconstructPath(end)     // we’ve done!
         Expand(node)                     // otherwise EXPAND it!
         closedList.Add(node)             // and mark it as “closed”
      }
      return null                         // if there is no nodes left, no path exists

Expand(probingNode):
      foreach (link in probingNode.links) {        // iterate through all outgoing links
         child = link.otherEnd(probingNode)         // get the link’s end
         newCost = pathCost[probingNode.links] + link.cost  // calculate path cost to child
         if (child is not in pathCost)             // have we touched the child in past?
         {                                          // NO => include child into openList
           pathCost[child] = newCost
           pathParent[child] = probingNode
           openList.Queue(child, newCost)
         } else
         {                                          // YES => compare costs
           oldCost = pathCost[child]                // retrieve current cost
           if (newCost < oldCost) {                 // have we found a better path?
             pathCost[child] = newCost              // YES => reinclude child in openList
             pathParent[child] = probingNode
             closedList.Remove(child)
             openList.QueueOrUpdateCost(child, newCost)
           }
         }
      }
```

Dijkstra's Algorithm
Visualization

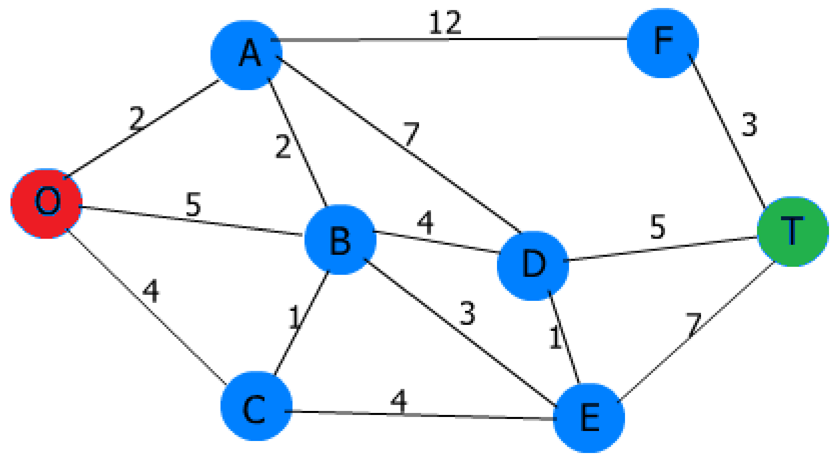
Task: Find shortest path between O and T



Dijkstra's Algorithm

Visualization

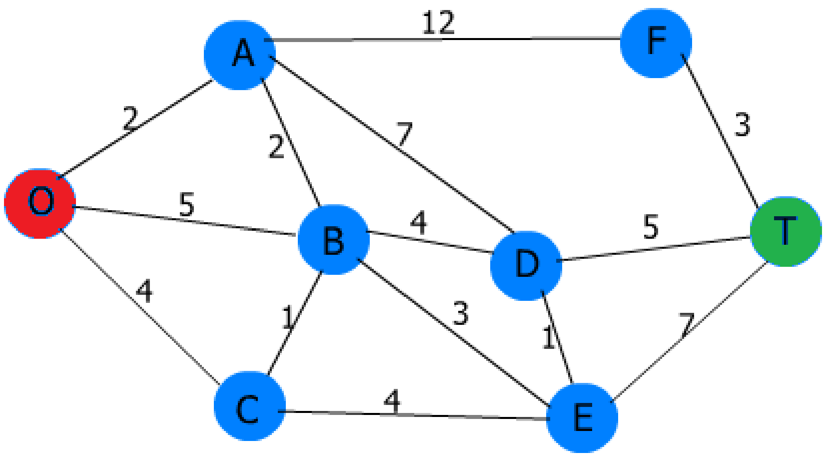
Step: Init data structures




```
probingNode = null
openList    = [O]
closedList  = []

pathCost    = { O => 0              pathParent = { O => null




            }                                   }
```

Dijkstra's Algorithm
Visualization

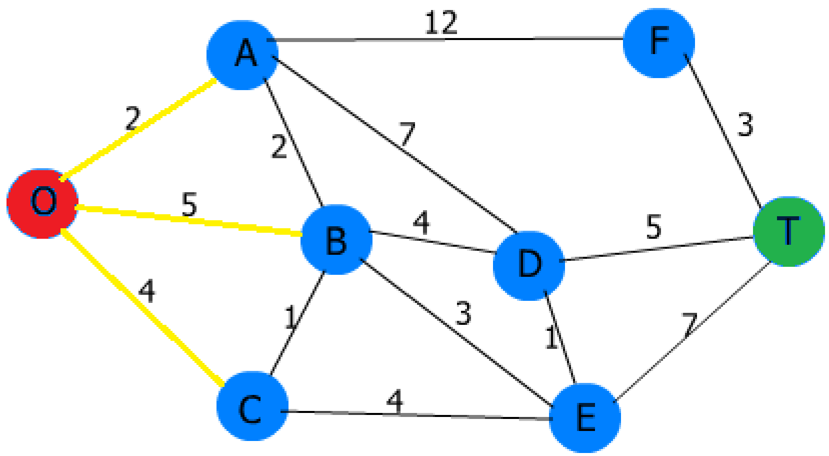
Step 1.1: Examining node **O**




```
probingNode = O
openList    = []
closedList  = []

pathCost    = { O => 0          pathParent = { O => null



              }                              }
```

Dijkstra's Algorithm
Visualization

Step 1.2: Probing links connecting node **O**

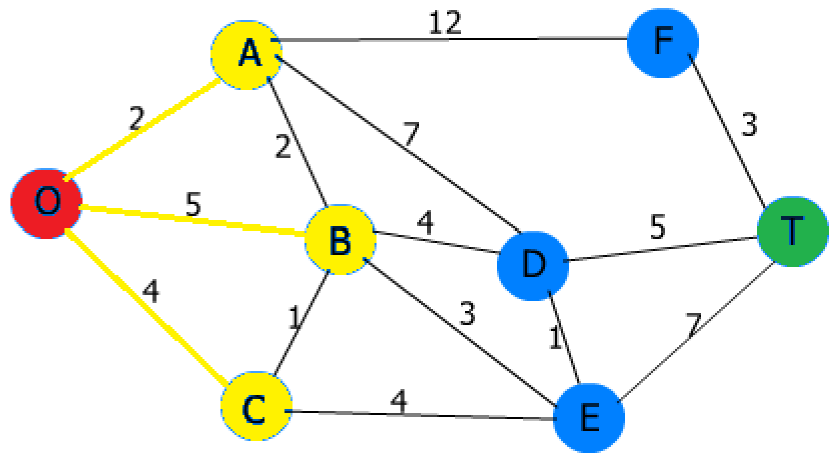



```
probingNode = O
openList    = []
closedList  = []

pathCost    = { O => 0              pathParent = { O => null





             }                                    }
```

Dijkstra's Algorithm

Visualization

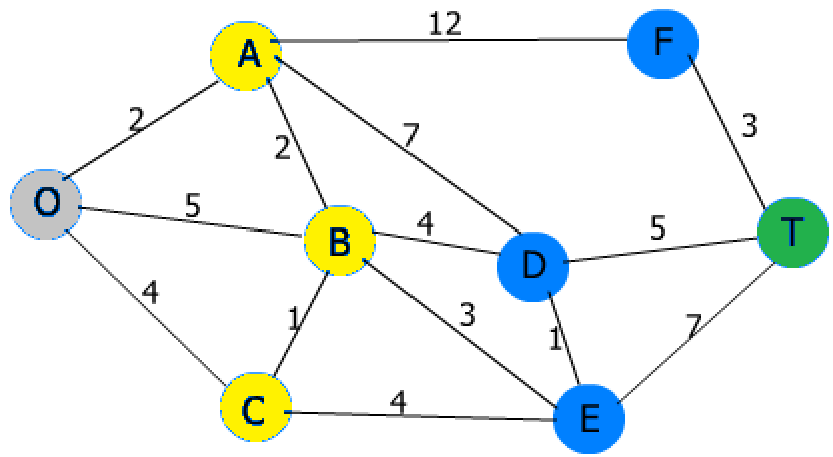
Step 1.3: Probed links connecting node **O**




```
probingNode = O
openList    = [A,C,B]
closedList  = []

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 5                             B => O
                C => 4                             C => O



              }                                  }
```

Dijkstra's Algorithm
Visualization

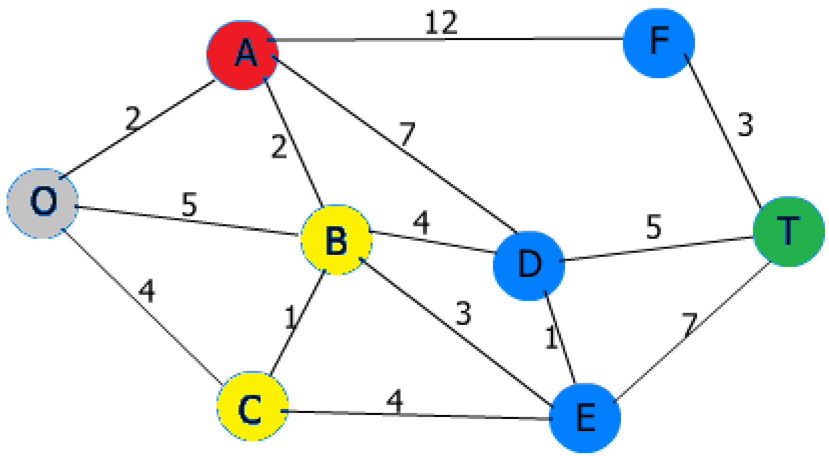
Step 1.4: Node **O** examined and moved to **closedList**




```
probingNode = null
openList    = [A,C,B]
closedList  = [O]

pathCost    = { O => 0            pathParent = { O => null
                A => 2                           A => O
                B => 5                           B => O
                C => 4                           C => O



              }                                }
```

Dijkstra's Algorithm
Visualization

Step 2.1: Examining node **A**



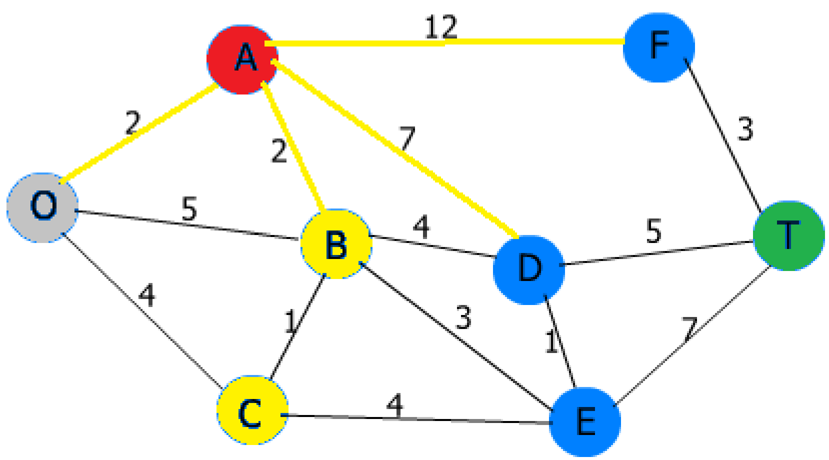
probingNode = **A**
openList = **[C,B]**
closedList = [O]

```
pathCost   = { O => 0            pathParent = { O => null
               A => 2                           A => O
               B => 5                           B => O
               C => 4                           C => O
             }                                }
```

Dijkstra's Algorithm

Visualization

Step 2.2: Probing links connecting node **A**

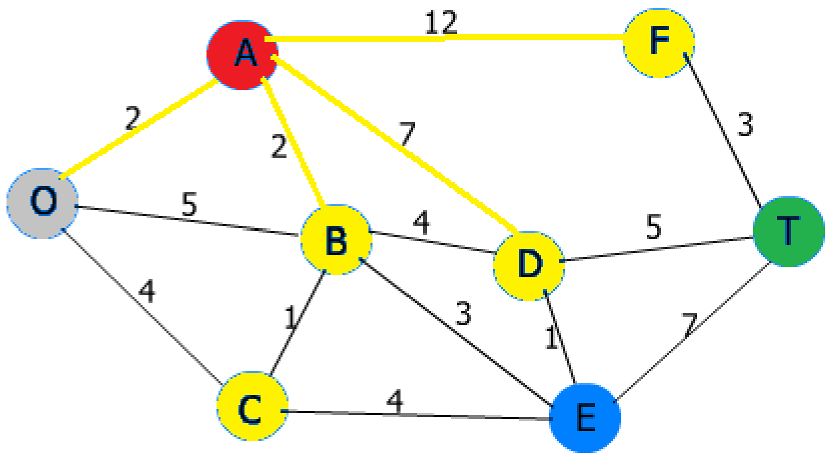



```
probingNode = A
openList    = [C,B]
closedList  = [O]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 5                         B => O
                C => 4                         C => O



              }                              }
```

Dijkstra's Algorithm

Visualization

Step 2.3: Probed links connecting node **A**



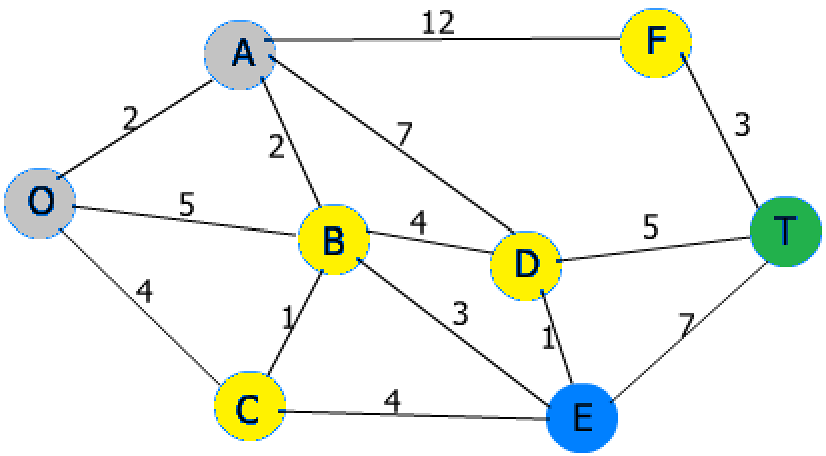
probingNode = A
openList = [C,B,**D**,**F**]
closedList = [O]

pathCost = { O => 0
A => 2
B => 4
C => 4
D => 9
F => 14
}

pathParent = { O => null
A => O
B => A
C => O
D => A
F => A
}

Dijkstra's Algorithm
Visualization

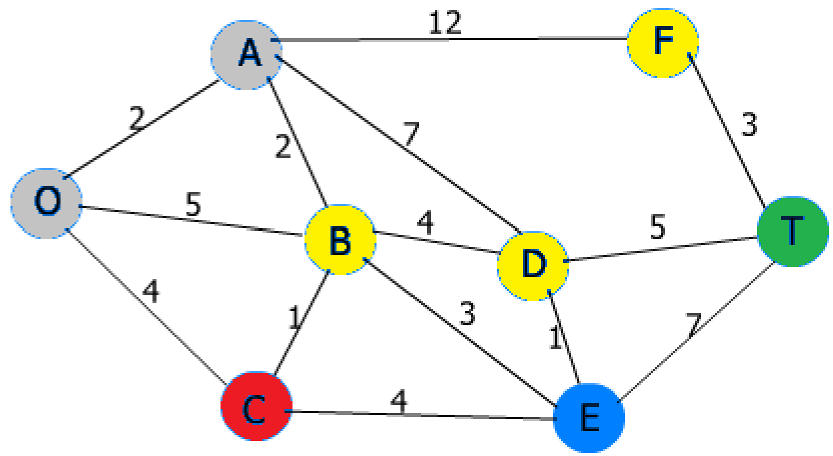
Step 2.4: Node **A** examined and moved to **closedList**




```
probingNode = null
openList    = [C,B,D,F]
closedList  = [O,A]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 9                             D => A
                F => 14                            F => A


              }                                  }
```

Dijkstra's Algorithm
Visualization

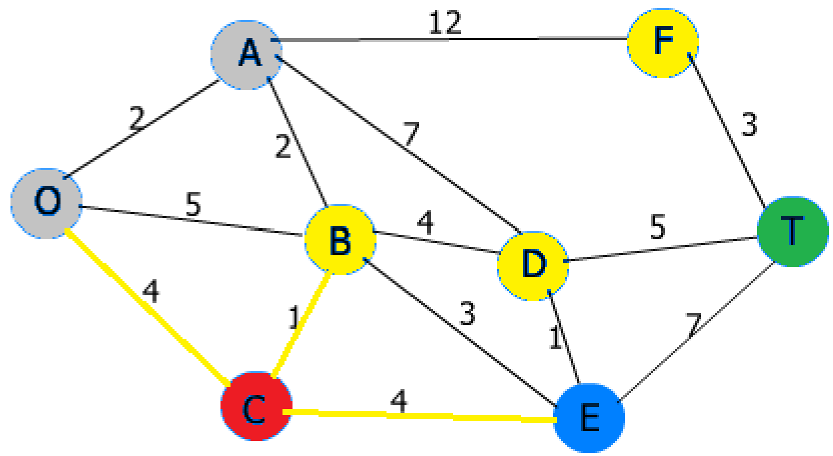
Step 3.1: Examining node **C**




```
probingNode = C
openList    = [B,D,F]
closedList  = [O,A]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 9                             D => A
                F => 14                            F => A


              }                                  }
```

Dijkstra's Algorithm
Visualization

Step 3.2: Probing links connecting node **C**

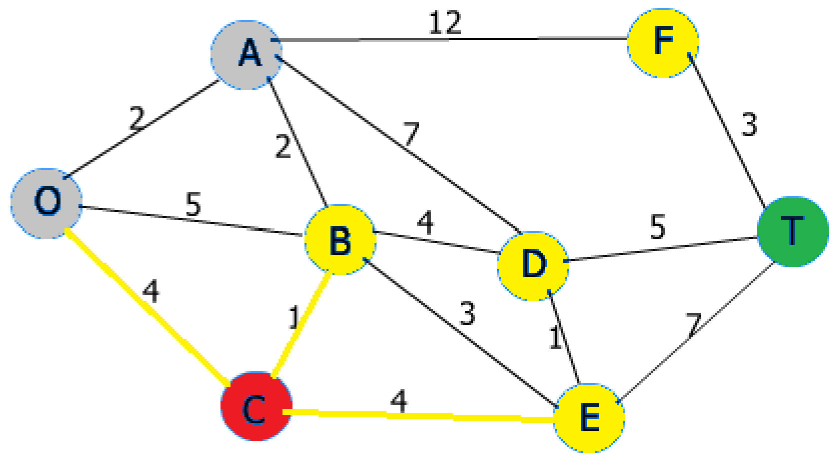



```
probingNode = C
openList    = [B,D,F]
closedList  = [O,A]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 4                         B => A
                C => 4                         C => O
                D => 9                         D => A
                F => 14                        F => A


              }                              }
```

Dijkstra's Algorithm

Visualization

Step 3.3: Probed links connecting node **C**

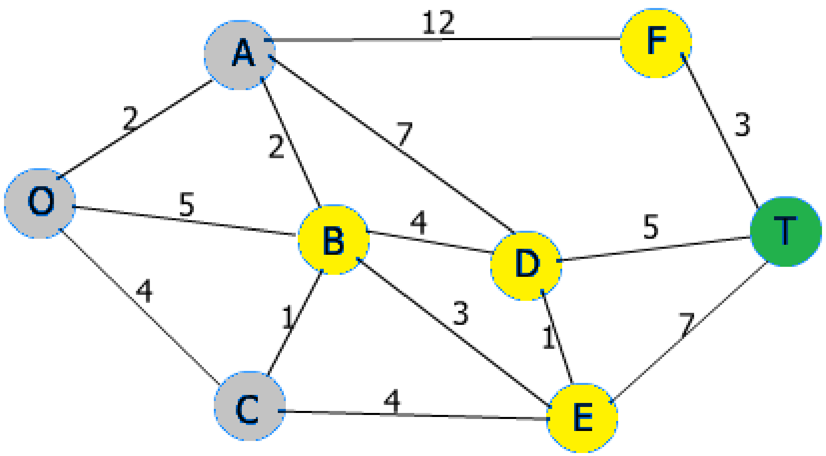



```
probingNode = C
openList    = [B,E,D,F]
closedList  = [O,A]

pathCost    = { O => 0               pathParent = { O => null
                A => 2                              A => O
                B => 4                              B => A
                C => 4                              C => O
                D => 9                              D => A
                F => 14                             F => A
                E => 8                              E => C

              }                                   }
```

Dijkstra's Algorithm

Visualization

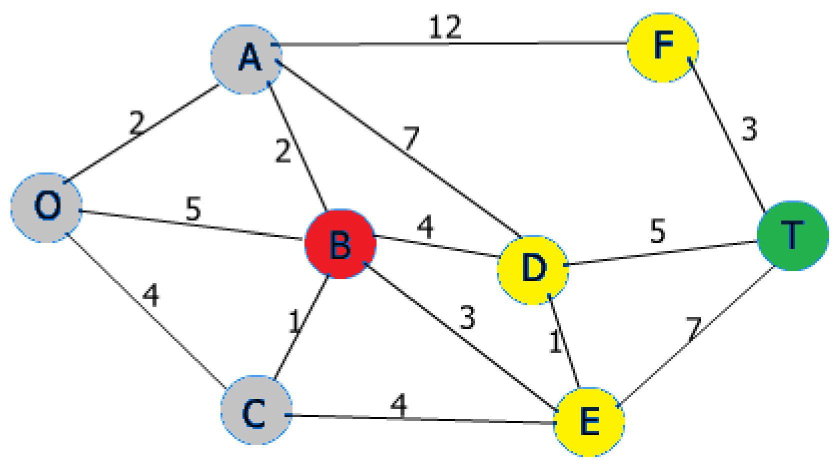
Step 3.4: Node **C** examined and moved to **closedList**




```
probingNode = null
openList    = [B,E,D,F]
closedList  = [O,A,C]

pathCost    = { O => 0            pathParent = { O => null
                A => 2                           A => O
                B => 4                           B => A
                C => 4                           C => O
                D => 9                           D => A
                F => 14                          F => A
                E => 8                           E => C

              }                                }
```

Dijkstra's Algorithm
Visualization

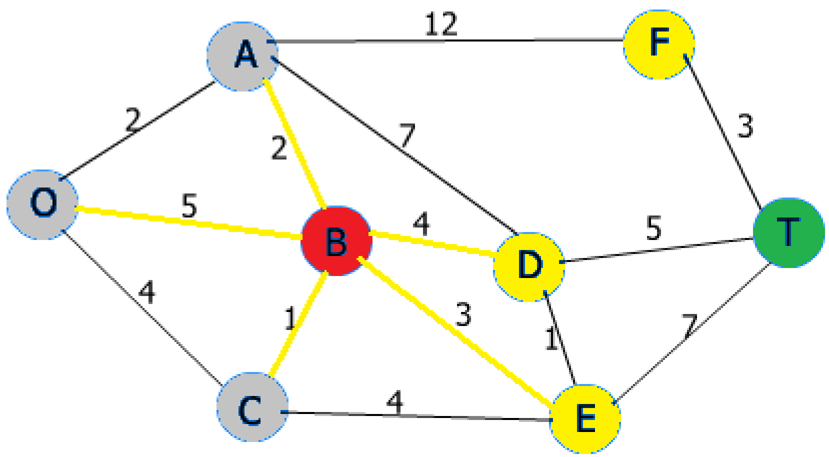
Step 4.1: Examining node **B**




```
probingNode = B
openList    = [E,D,F]
closedList  = [O,A,C]

pathCost    = { O => 0
                A => 2
                B => 4
                C => 4
                D => 9
                F => 14
                E => 8

              }

pathParent = { O => null
               A => O
               B => A
               C => O
               D => A
               F => A
               E => C

             }
```

Dijkstra's Algorithm
Visualization

Step 4.2: Probing links connecting node **B**

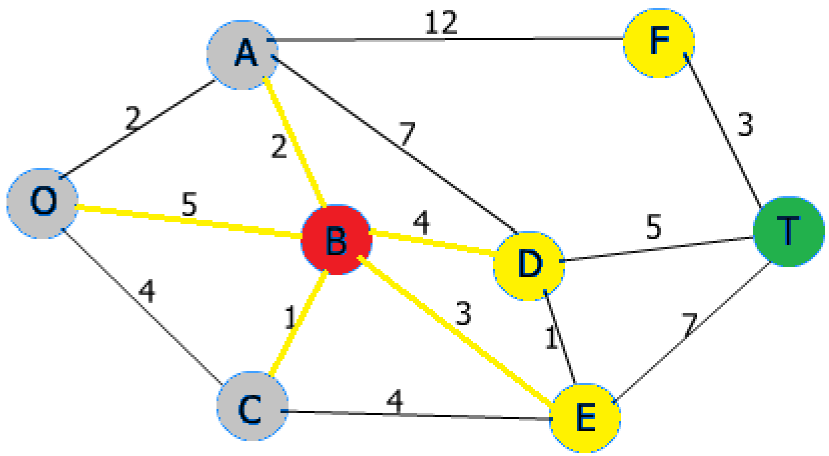



```
probingNode = B
openList    = [E,D,F]
closedList  = [O,A,C]

pathCost    = { O => 0           pathParent = { O => null
                A => 2                          A => O
                B => 4                          B => A
                C => 4                          C => O
                D => 9                          D => A
                F => 14                         F => A
                E => 8                          E => C

              }                               }
```

Dijkstra's Algorithm

Visualization

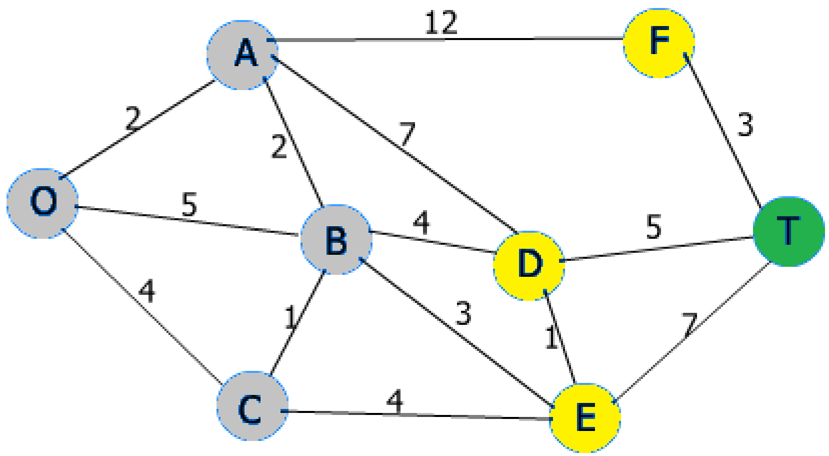
Step 4.3: Probed links connecting node **B**




```
probingNode = B
openList    = [E,D,F]
closedList  = [O,A,C]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 4                         B => A
                C => 4                         C => O
                D => 8                         D => B
                F => 14                        F => A
                E => 8                         E => C

              }                              }
```

Dijkstra's Algorithm
Visualization

Step 4.4: Node **B** examined and moved to **closedList**

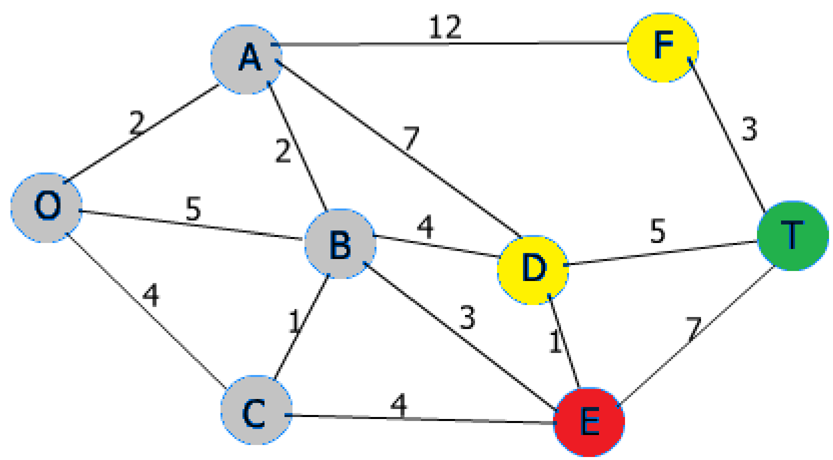



```
probingNode = null
openList    = [E,D,F]
closedList  = [O,A,C,B]

pathCost    = { O => 0             pathParent = { O => null
                A => 2                            A => O
                B => 4                            B => A
                C => 4                            C => O
                D => 8                            D => B
                F => 14                           F => A
                E => 8                            E => C

              }                                 }
```

Dijkstra's Algorithm

Visualization

Step 5.1: Examining node **E**



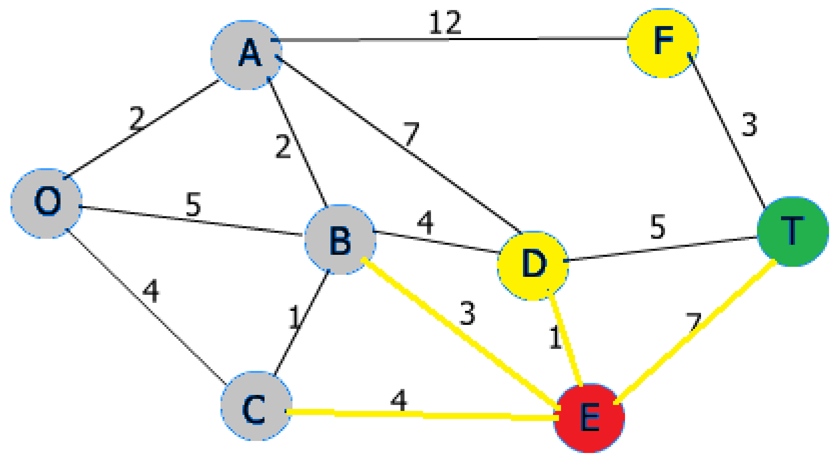

```
probingNode = E
openList    = [D,F]
closedList  = [O,A,C,B]

pathCost    = { O => 0
                A => 2
                B => 4
                C => 4
                D => 8
                F => 14
                E => 8

              }
```

```
pathParent = { O => null
               A => O
               B => A
               C => O
               D => B
               F => A
               E => C

             }
```

Dijkstra's Algorithm

Visualization

Step 5.2: Probing links connecting node **E**



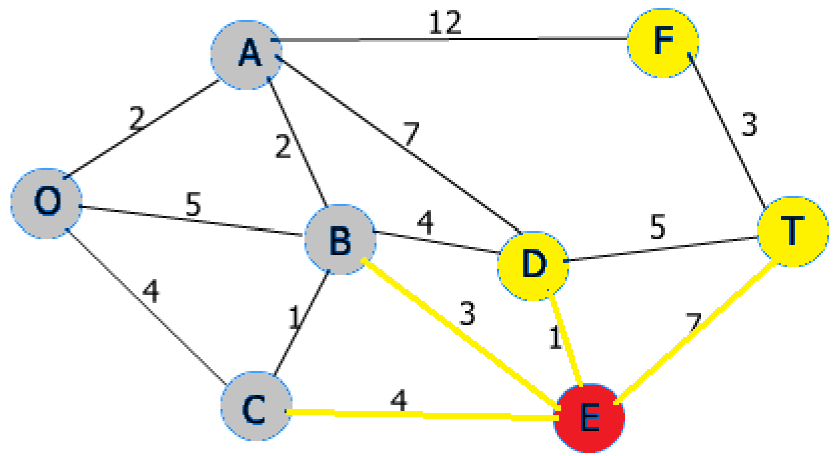

```
probingNode = E
openList    = [D,F]
closedList  = [O,A,C,B]
```

```
pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 8                             D => B
                F => 14                            F => A
                E => 8                             E => C

              }                                  }
```

Dijkstra's Algorithm

Visualization

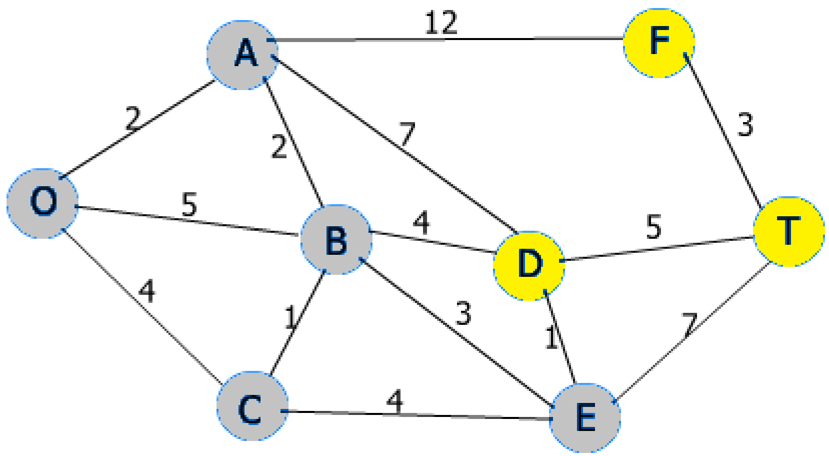
Step 5.3: Probed links connecting node **E**




```
probingNode = E
openList    = [D,F,T]
closedList  = [O,A,C,B]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 4                         B => A
                C => 4                         C => O
                D => 8                         D => B
                F => 14                        F => A
                E => 8                         E => C
                T => 15                        T => E
              }                              }
```

Dijkstra's Algorithm
Visualization

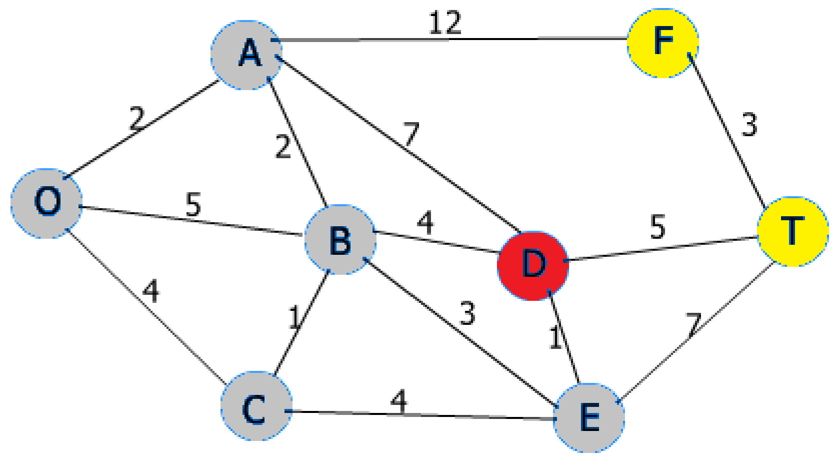
Step 5.4: Node **E** examined and moved to **closedList**




```
probingNode = null
openList    = [D,F,T]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 8                             D => B
                F => 14                            F => A
                E => 8                             E => C
                T => 15                            T => E
              }                                  }
```

Dijkstra's Algorithm
Visualization

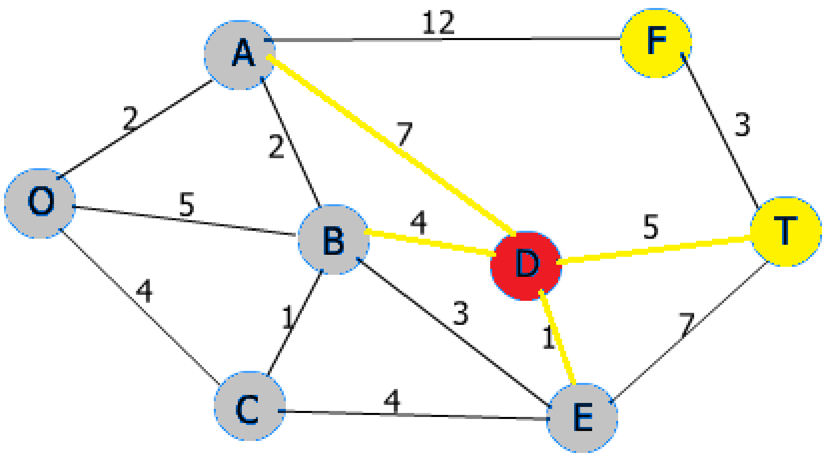
Step 6.1: Examining node **D**




```
probingNode = D
openList    = [F,T]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0           pathParent = { O => null
                A => 2                          A => O
                B => 4                          B => A
                C => 4                          C => O
                D => 8                          D => B
                F => 14                         F => A
                E => 8                          E => C
                T => 15                         T => E
              }                               }
```

Dijkstra's Algorithm
Visualization

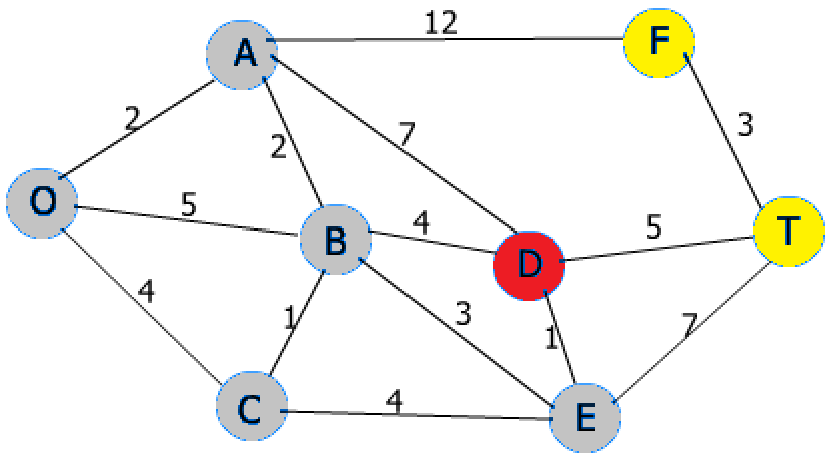
Step 6.2: Probing links connecting node **D**




```
probingNode = D
openList    = [F,T]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 4                         B => A
                C => 4                         C => O
                D => 8                         D => B
                F => 14                        F => A
                E => 8                         E => C
                T => 15                        T => E
              }                              }
```

Dijkstra’s Algorithm
Visualization

Step 6.3: Probed links connecting node **D**

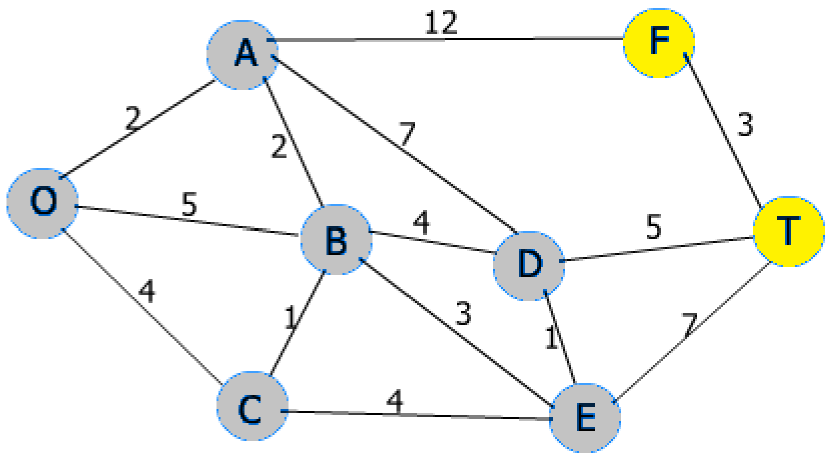



```
probingNode = D
openList    = [T,F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 8                             D => B
                F => 14                            F => A
                E => 8                             E => C
                T => 13                            T => D
              }                                  }
```

Dijkstra's Algorithm

Visualization

Step 6.4: Node **D** examined and moved to **closedList**

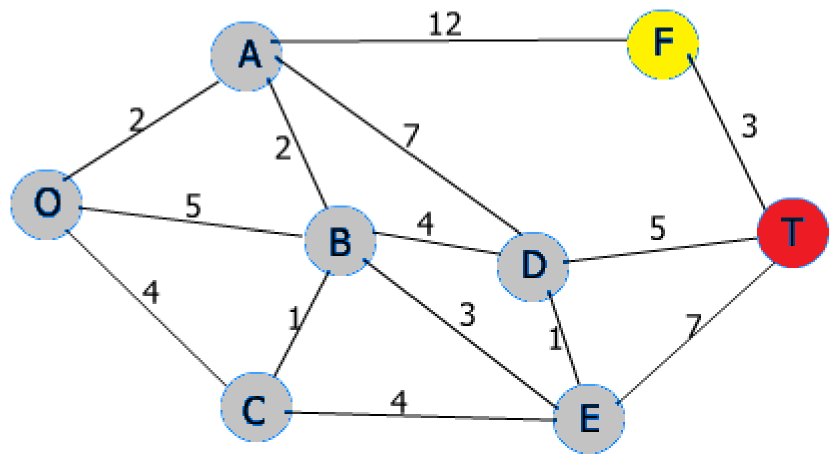



```
probingNode = null
openList    = [T,F]
closedList  = [O,A,C,B,E,D]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 4                         B => A
                C => 4                         C => O
                D => 8                         D => B
                F => 14                        F => A
                E => 8                         E => C
                T => 13                        T => D
              }                              }
```

Dijkstra's Algorithm

Visualization

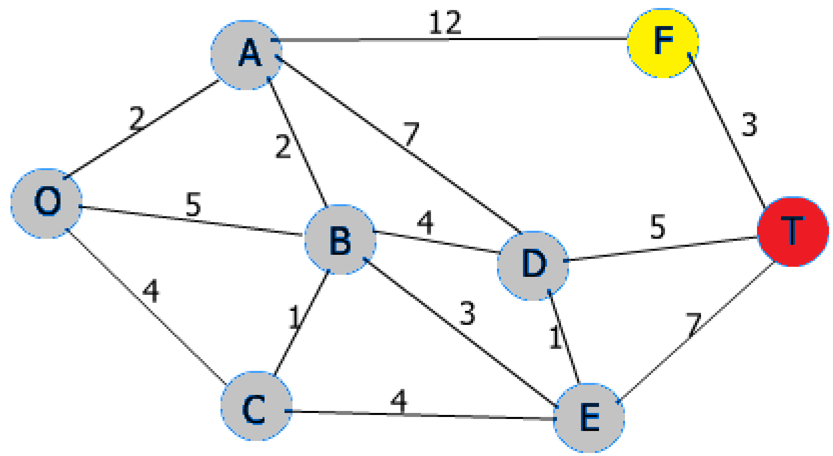
Step 7.1: Examining node T




```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 8                             D => B
                F => 14                            F => A
                E => 8                             E => C
                T => 13                            T => D
              }                                  }
```

Dijkstra's Algorithm
Visualization

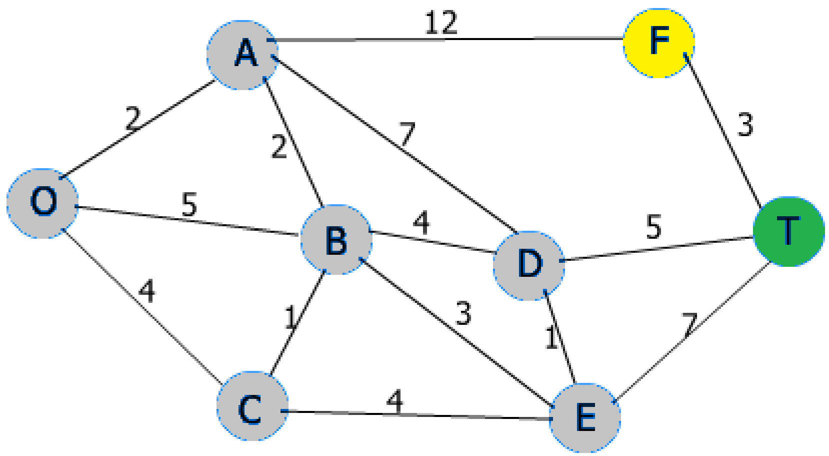
Step 7.1: Node T is our target node!




```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0            pathParent = { O => null
                A => 2                           A => O
                B => 4                           B => A
                C => 4                           C => O
                D => 8                           D => B
                F => 14                          F => A
                E => 8                           E => C
                T => 13                          T => D
              }                                }
```

Dijkstra’s Algorithm
Visualization

Step 7.1: Reconstructing path to **T** given **pathParent** map

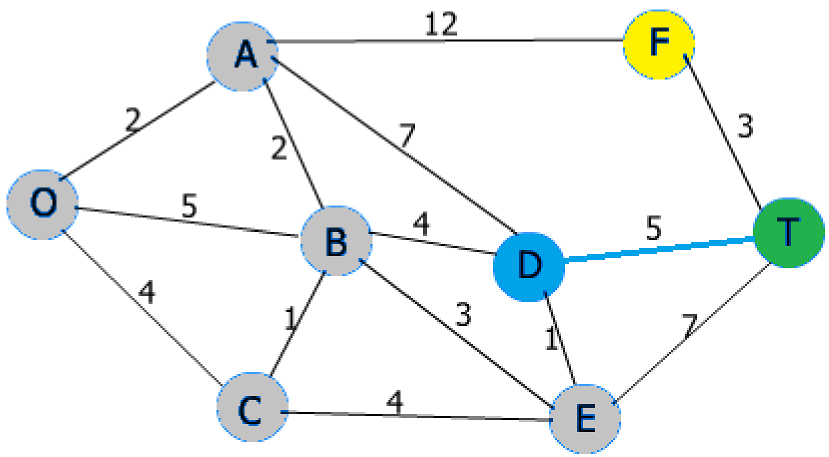



```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 8                             D => B
                F => 14                            F => A
                E => 8                             E => C
                T => 13                            T => D
              }                                  }
```

Dijkstra's Algorithm

Visualization

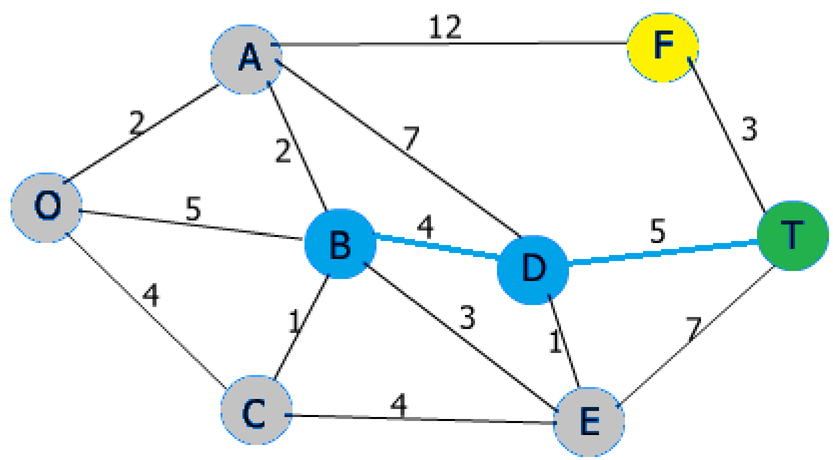
Step 7.1: Reconstructing path to **T** given **pathParent** map




```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0            pathParent = { O => null
                A => 2                           A => O
                B => 4                           B => A
                C => 4                           C => O
                D => 8                           D => B
                F => 14                          F => A
                E => 8                           E => C
                T => 13                          T => D
              }                                }
```

Dijkstra's Algorithm
Visualization

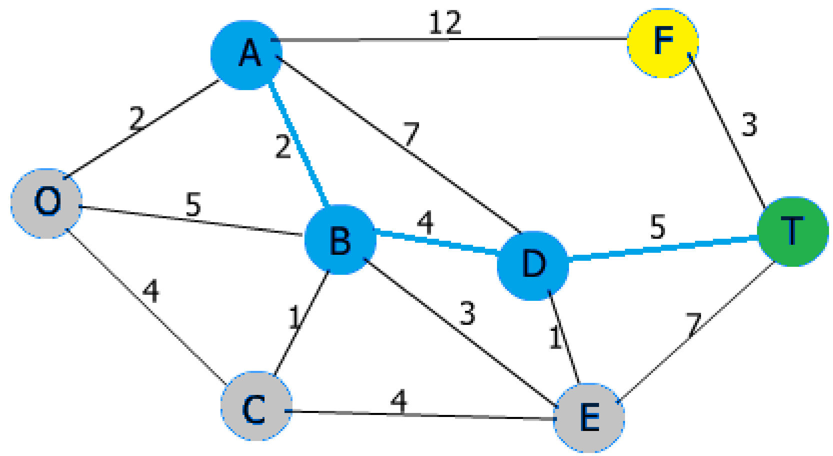
Step 7.1: Reconstructing path to **T** given **pathParent** map




```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0          pathParent = { O => null
                A => 2                         A => O
                B => 4                         B => A
                C => 4                         C => O
                D => 8                         D => B
                F => 14                        F => A
                E => 8                         E => C
                T => 13                        T => D
              }                              }
```

Dijkstra's Algorithm
Visualization

Step 7.1: Reconstructing path to **T** given **pathParent** map

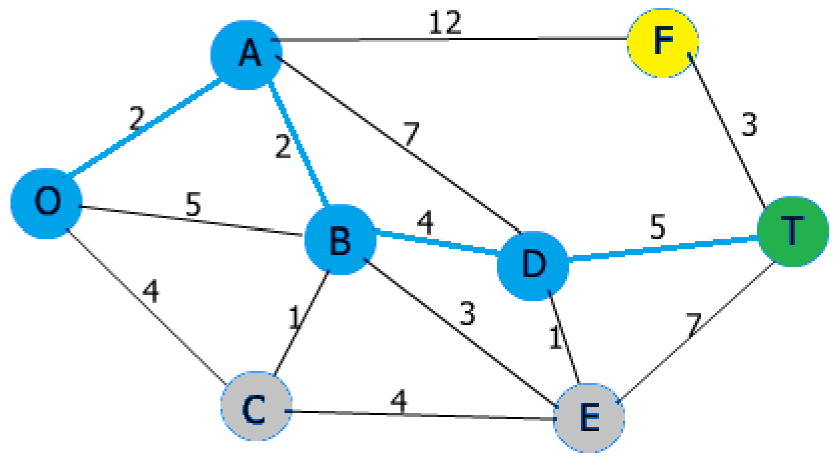



```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0              pathParent = { O => null
                A => 2                             A => O
                B => 4                             B => A
                C => 4                             C => O
                D => 8                             D => B
                F => 14                            F => A
                E => 8                             E => C
                T => 13                            T => D
              }                                  }
```

Dijkstra's Algorithm

Visualization

Step 7.1: Reconstructing path to **T** given **pathParent** map

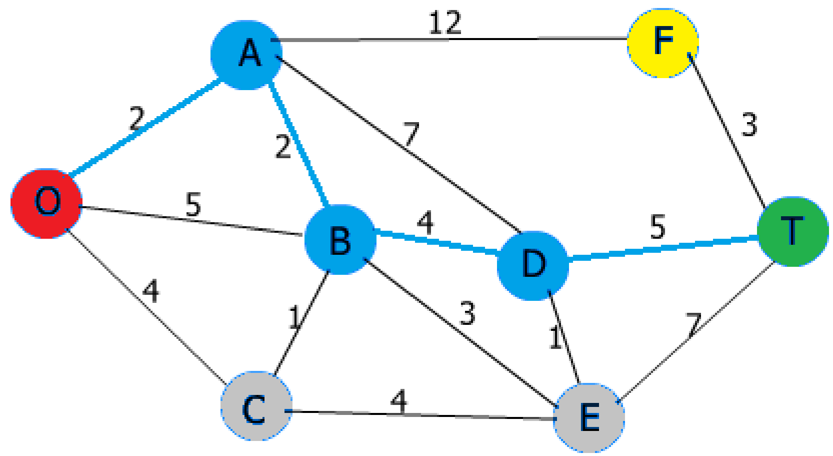



```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0           pathParent = { O => null
                A => 2                          A => O
                B => 4                          B => A
                C => 4                          C => O
                D => 8                          D => B
                F => 14                         F => A
                E => 8                          E => C
                T => 13                         T => D
              }                               }
```

Dijkstra's Algorithm

Visualization

Result: Shortest path found! O=>A=>B=>D=>T, path cost = 13




```
probingNode = T
openList    = [F]
closedList  = [O,A,C,B,E]

pathCost    = { O => 0                pathParent = { O => null
                A => 2                               A => O
                B => 4                               B => A
                C => 4                               C => O
                D => 8                               D => B
                F => 14                              F => A
                E => 8                               E => C
                T => 13                              T => D
              }                                    }
```

Topic
Navigation

Heap

Topic
Navigation

Dictionary

Tools of Trade

Dictionary + CompositeKeys

```
Dictionary<NodeType, Dictionary<int, INode>> nodes = new Dictionary<NodeType, Dictionary<int, INode>>();

Dictionary<NodeKey, INode> nodeByKey = new Dictionary<NodeKey, INode>();
```

```
6 references
class NodeKey
{

    private int id;
    private NodeType type;
    private int hashCode;

    0 references
    public NodeKey(int id, NodeType type)
    {
        this.id = id;
        this.type = type;
        this.hashCode = 7 * id + 23 * typeof(NodeType).GetHashCode();
    }

    0 references
    public override bool Equals(object obj)
    {
        if (!(obj is NodeKey)) return false;
        NodeKey nodeKey = (NodeKey)obj;
        if (id == nodeKey.id && type == nodeKey.type) return true;
        return false;
    }

    1 reference
    public override int GetHashCode()
    {
        return hashCode;
    }

}
```

Topic
Navigation

WalkLink vs. LiftLink

Debugging and ToString

Tools of Trade

Debugging + ToString()

- Default "String" representation of the object, e.g. Node

```
3 references
public override string ToString()
{
    return "Node[" + Enum.GetName(typeof(NodeType), type) + "-" + id + "]";
}
```

this	{Workshop05.Graph}
fromNode	{Workshop05.Node}
toNode	{Workshop05.Node}
person	{Workshop05.Patient}
item	{Workshop05.SearchItem}
heap	{Workshop05.Heap<Workshop05.SearchItem>}
opened	Count = 0
finished	Count = 1
pathFound	false

this	{Workshop05.Graph}
fromNode	{Node[ENTRANCE-1]}
toNode	{Node[INFODESK-1]}
person	{Workshop05.Patient}
item	{Workshop05.SearchItem}
heap	{Workshop05.Heap<Workshop05.SearchItem>}
opened	Count = 0
finished	Count = 1
pathFound	false

- To be used for DEBUGGING only! Do not misuse for "pretty printing that is handy for your billing application"!

Theme Hospital - Simulation

Theme Hospital Lite

Navigation - Time

- The link’s cost is in “seconds”
- So if lift’s cost is “10” it means it travels the link in 10 seconds.
- If person with speedMultiplier 2 is travelling through “walk” link of cost 20, then it means it will take them “2*20=40” seconds

Theme Hospital Lite

Navigation - Lifts

- Now you will have to simulate LIFTs!
- This means that you have to know where lift “begins”

Lift link: `[ <lift-left-link> | <lift-right-link> ]`

```
lift-left-link: ‘L<--(lift:c’ <capacity> ‘:t’<cost>
  ‘)-->’

lift-right-link: ‘<--(lift:c’ <capacity> ‘:t’<cost>
  ‘)-->L’
```

Theme Hospital Lite
Navigation - Lifts

- Person (patient or doctor) will always try to use the lift
- When the person arrives to the lift, following cases may occur
  1. Lift is there => Person will immediately use it
  2. Lift is not there & Waiting queue (of lift capacity length) is not full => Person will wait for the lift to arrive
  3. Lift is not there & Waiting queue is full => Person will take detour

Theme Hospital Lite

The Simulation

- Patient's route:
  - Own entrance (you cannot choose this!)

  -> nearest INFODESK

  -> nearest GP that has a doctor inside
    - If no such exist, than just "nearest GP"

  -> nearest special diagnose room that has a doctor inside
    - If no such exist, than just "nearest one"

  -> nearest GP that has a doctor inside
    - If no such exist, than just "nearest GP"

  -> nearest TREATMENT

  -> nearest ENTRANCE

Theme Hospital Lite

The Simulation

- INFODESK / TREATMENT
  - Each info desk / treatment has a „service speed associated“, that is, how much time it needs to “tell the patient how to navigate around the hospital”, resp. “cure the patient”
  - This speed is fixed
  - There can be any number of patients waiting in the queue of an infodesk / treatment
  - Path is determined by the “start service time”

Theme Hospital Lite

The Simulation

- GPs / Specific diagnose room
  - Similar to INFODESK/TREATMENT, but this time, the speed of service is determined by the doctor who is in the room
  - There can be any number of patients waiting in the queue of this room as well

Theme Hospital Lite

The Simulation

- Doctors & GPs
- While there are patients in the queue of the room, the doctor won’t leave his/her office
  - Whenever there is no queue, two cases may arrise
    1. [GP] There is no other room that has a patient trying to “use” or navigating to in order to “use” it => doctor stays in his/her current room
    2. There is such a room and

       2.1 There is a doctor who is navigating there => doctor ignores it

       2.2 There is no doctor travelling there =>

       2.2.1 And this doctor is the nearest one => travel there

       2.2.2 Is not the nearest one => stays in his/her current room

Theme Hospital Lite

The Simulation

- Doctors & Diagnostic rooms
- While there are patients in the queue of the room, the doctor won't leave his/her office
  - Whenever there is no queue, two cases may arrise
    1. [Diagnostic] There is no other room that has a patient trying to "use" or navigating to in order to "use" it => doctor goes to the nearest unoccupied GP
    2. There is such a room and

       2.1 There is a doctor who is navigating there => doctor ignores it

       2.2 There is no doctor travelling there =>

       2.2.1 And this doctor is the nearest one => travel there

       2.2.2 Is not the nearest one => stays in his/her current room

Theme Hospital Lite

The Simulation

- Doctors & GPs / Diagnoses
  - While there are patients in the queue of the room, the doctor won’t leave his/her office
  - Whenever there is no queue, two cases may arrise
    1. There is no other room that has a patient trying to “use” or navigating to in order to “use” it => doctor stays in his/her current room
    2. There is such a room and

       2.1 There is a doctor who is navigating there => doctor ignores it

       2.2 There is no doctor travelling there =>

       2.2.1 And this doctor is the nearest one => travel there

       2.2.2 Is not the nearest one => stays in his/her current room

Assignment 6
Theme Hospital Lite

INPUT: <int> '\n' [<node> ' ' <link> ' ' <node> '\n']+ <int> '\n' [<patient> '\n']+ <int> '\n' <int> [<infodesk/treatment> '\n']+ '\n' <int> [<doctor> '\n']+ '\n'

<node>: <node-type> '-' <id>

<node-type>: ['ENTRANCE' | 'INFODESK' | 'GP' | 'EEG' | 'SONO' | 'XRAY' | 'PSYCHO' | 'TREATMENT' | 'NODE']

<id>: <int>

<int>: [1-9][0-9]{0,1}

<link>: [<walk-link> | <lift-link>]

<walk-link>: [<non-oriented-walk-link> | <oriented-walk-link>]

<non-oriented-walk-link>: '<--(walk:' <int> ')-->'

<oriented-walk-link>: '--(walk:' <cost> ')-->'

<lift-link>: [<lift-left-link> | <lift-right-link>]

<lift-left-link>: 'L<--(lift:c' <capacity> ':t'<cost> ')-->'

<lift-right-link>: '<--(lift:c' <capacity> ':t'<cost> ')-->L'

<cost>: <int>

<capacity>: <int>

Assignment 6

Theme Hospital Lite Navigation

INPUT: <int> '\n' [<node> ' ' <link> ' ' <node> '\n']+ <int> '\n' [<patient> '\n']+ <int> '\n' <int> [<infodesk/treatment> '\n']+ '\n' <int> [<doctor> '\n']+ '\n'

<patient>: <name> ':' <speed-multiplier> ':' <health-problem> ':' <node> ':' <time>

<name>: [A-Z][a-zA-Z]+

<speed-multiplier>: <int>

<health-problem>: ['CARDIAC' | 'PNEUMONIA' | 'HIP-PAIN' | 'NEUROTIC']

<time>: [0-2][0-9] ':' [0-2][0-9] ':' [0-2][0-9]

Assignment 6

Theme Hospital Lite Navigation

INPUT: <int> '\n' [<node> ' ' <link> ' ' <node> '\n']+ <int> '\n' [<patient> '\n']+ <int> '\n' <int> [<infodesk/treatment> '\n']+ '\n' <int> [<doctor> '\n']+ '\n'

<infodesk/treatment>: <node> ':' <service-time>

<service-time>: <int>

<doctor>: <name> ':' <speed-multiplier> ':' <service-time>

Assignment 6

Theme Hospital Lite

Output:

Which doctors are you going to use and in which rooms they should begin + when the last patient leaves the hospital (reaches his/her exit ENTRANCE node).

The hospital opens at 08:00:00.

The hospital closes at 18:00:00.

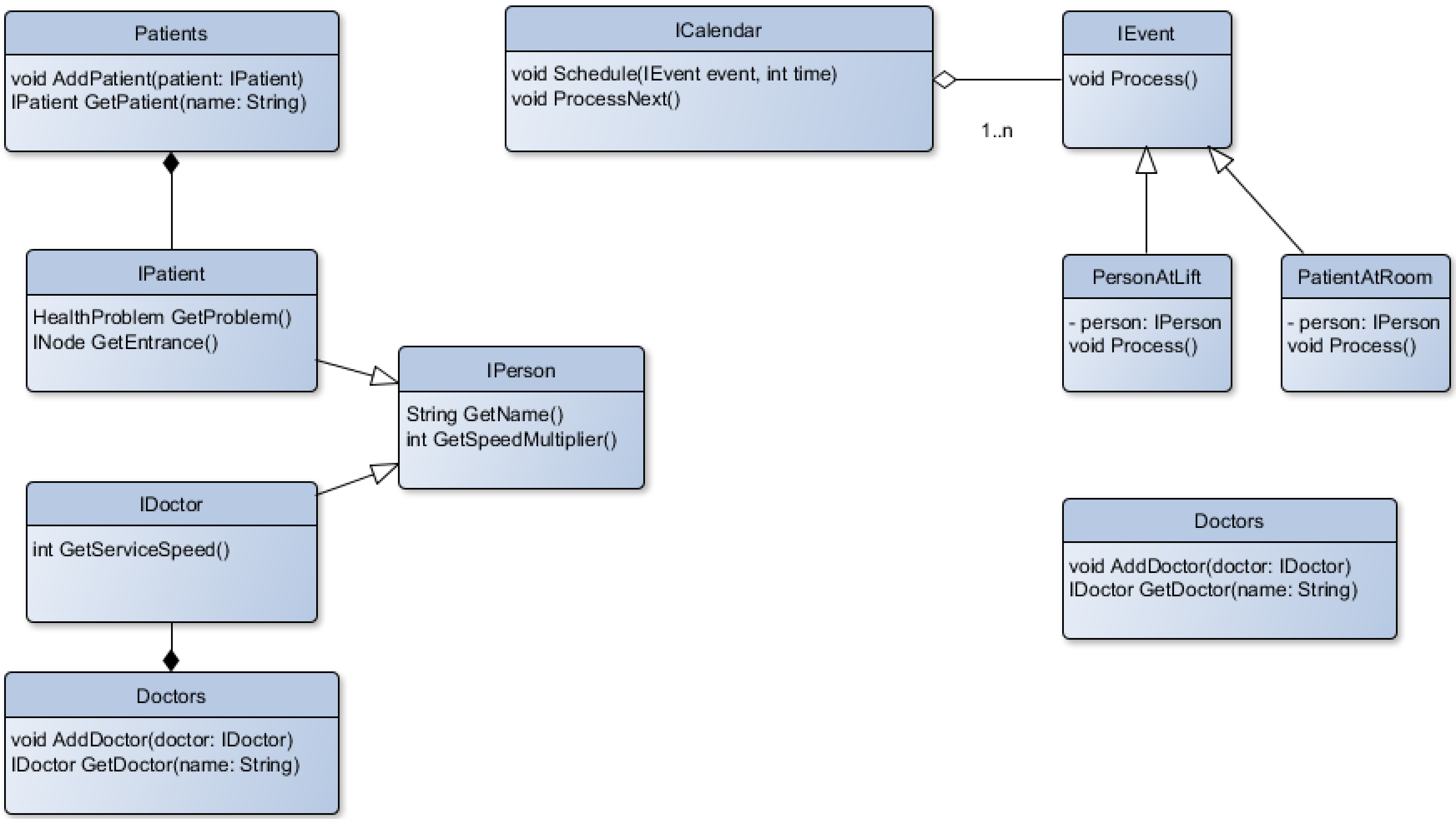
[<doctor-start> '\n']+ <finishing-time>

<doctor-start>: <name> ':' <node>

<finishing-time>: <time>

Assignment 6

Design time!



Assignment 06+07
Send me an email

- Email: jakub.gemrot@gmail.com
- Subject: **Programming II – 2016 – Assignment 07**
- Zip up the whole project and send it
- You WILL NOT find the assignment in CoDex!
- Deadline: **30.9.2016**

Questions?
I sense a soul in search of answers…

- In case of doubts about the assignment or some other problems don't hesitate to contact me!
  - Jakub Gemrot
    - gemrot@gamedev.cuni.cz