CS380

Artificial Intelligence for Games

Dijkstra’s Search In Explicit Graph

Dijkstra’s Search

Edsger Wybe Dijkstra
1930 – 2002

Main contributions:

- Dijkstra's algorithm
- Semaphore

Recipient of Turing Award (1972)

Dijkstra's Search

- Finds the shortest path from the source node to **all** other nodes in the graph with **non-negative** edge costs.
- Is a **Best-first Search Algorithm** - makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem
- Similar to **Uniform-Cost Search**
  - Both use **openlist** implemented as a **priority queue**
  - DS finds shortest paths to all other nodes while UCS usually has one target node to reach
  - DS has slightly higher computational cost than UCS

Dijkstra’s Search. Initialization

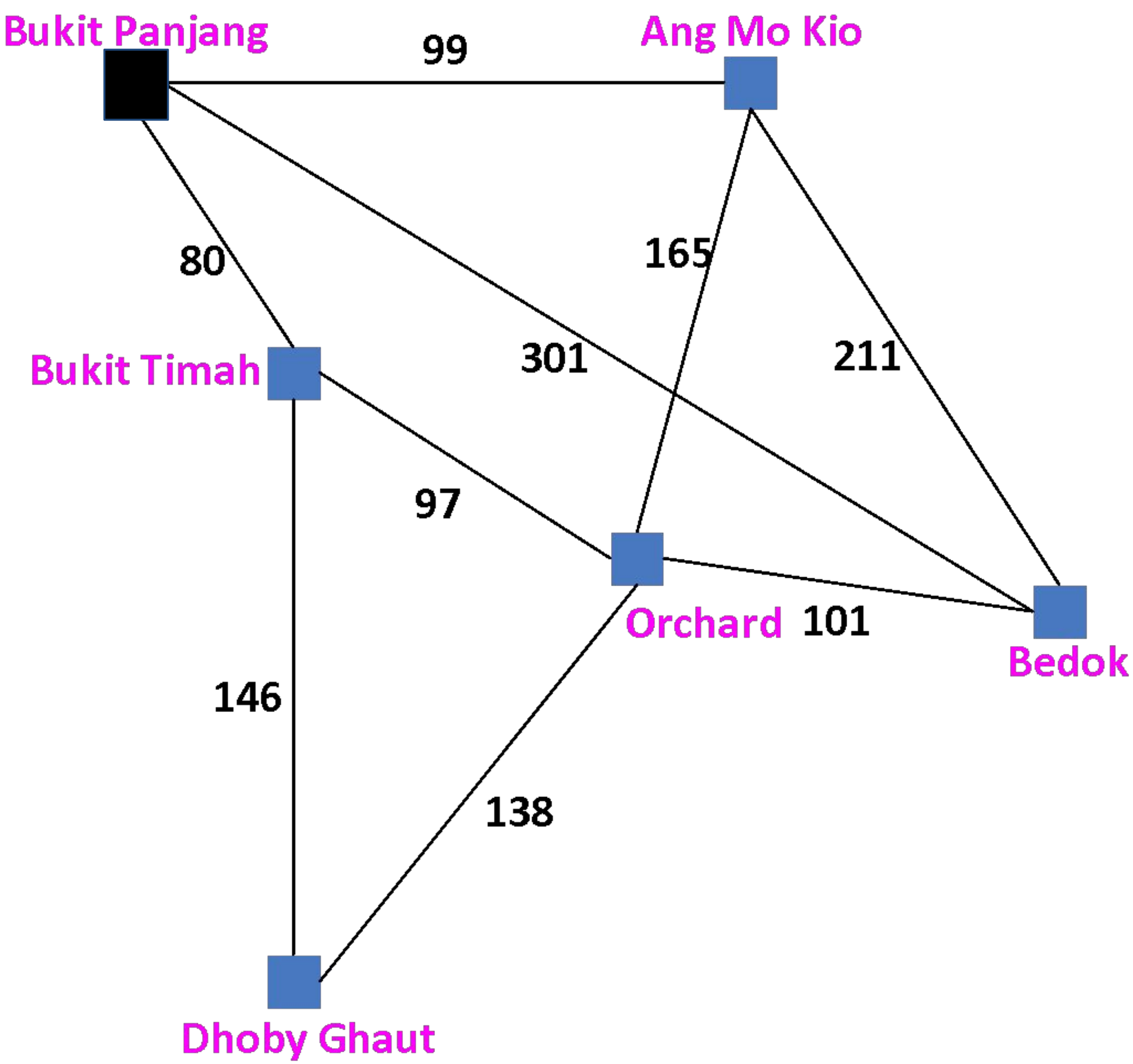
- Given:
  - graph,
  - starting and target nodes
  - empty openelist as priority queue
  - closelist as a hash table
- Initialization step:

```
for_each (node of graph) {
    node.g = +∞;   // Calculated distance is set as infinite
    node.parent = null; // Previous node is undefined
}
 starting.g = 0;
openlist.push(starting);
```

Dijkstra’s Search. Main loop

```
while (!openlist.empty()){
    current = openlist.pop();
    closedlist.add(current); // Set as “visited”
    for_each(adjacent of current)
        if (!closedlist.find(adjacent)) {
            if (!openlist.find(adjacent)) {
                adjacent.g = current.g + cost(current, adjacent);
                adjacent.parent = current;
                openlist.push(adjacent);
            } else {
                tentative_g = current.g + cost(current, adjacent);
                if (tentative_g  < adjacent.g){
                    adjacent.g = tentative_g;
                    adjacent.parent = current;
    }   }   }
```

Dijkstra's Search

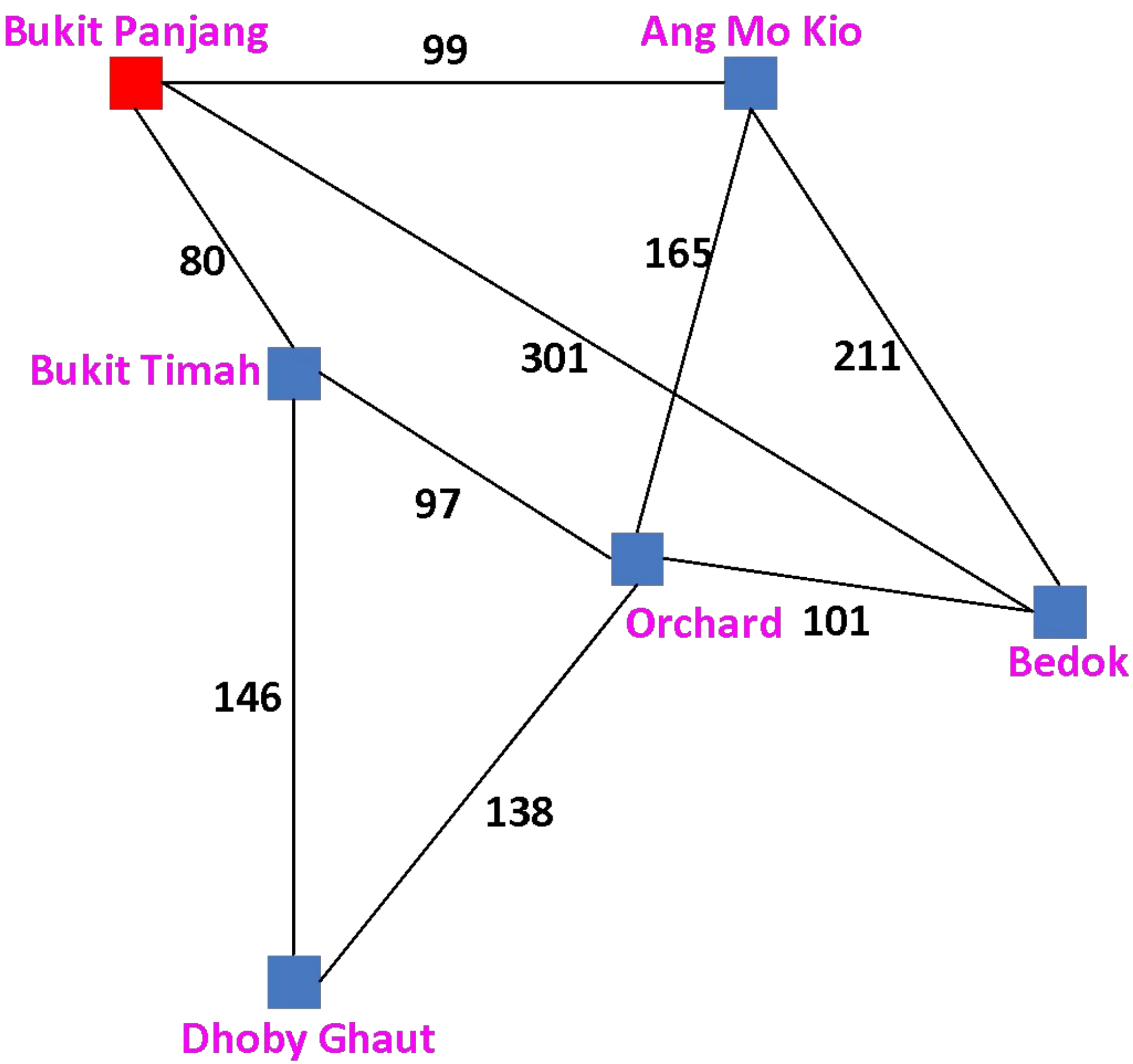



```
BP.parent  = null
AMK.parent = null
BT.parent  = null
O.parent   = null
B.parent   = null
DG.parent  = null

BP.g  = 0
AMK.g = +∞
BT.g  = +∞
O.g   = +∞
B.g   = +∞
DG.g  = +∞

openlist={BP}
closedlist={}
```

Dijkstra's Search

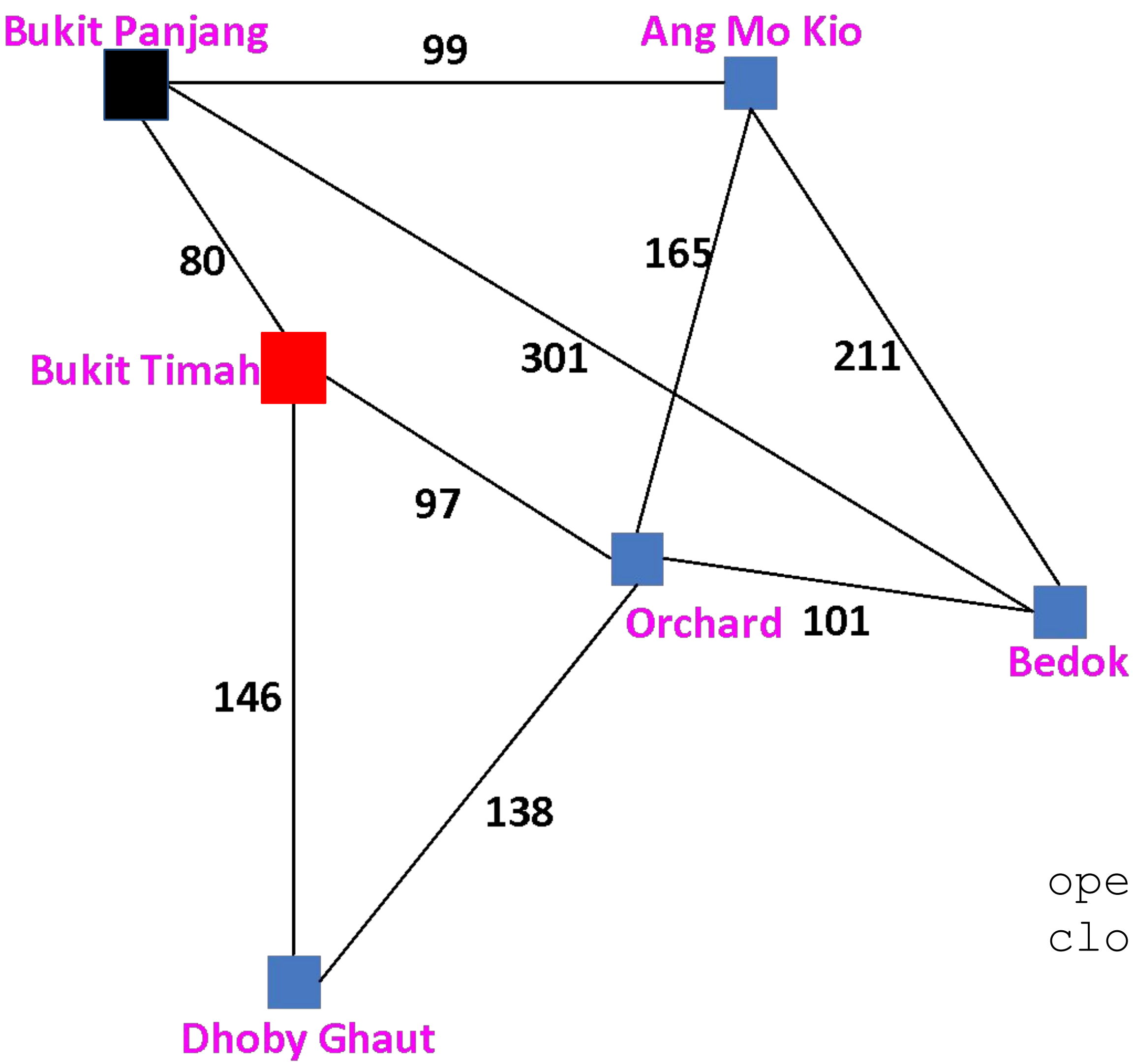



```
BP.parent  = null
AMK.parent = BP
BT.parent  = BP
O.parent   = null
B.parent   = BP
DG.parent  = null
```

```
BP.g  = 0
AMK.g = 99
BT.g  = 80
O.g   = +∞
B.g   = 301
DG.g  = +∞
```

```
openlist={BT,AMK,B}
closedlist={BP}
```

Dijkstra's Search

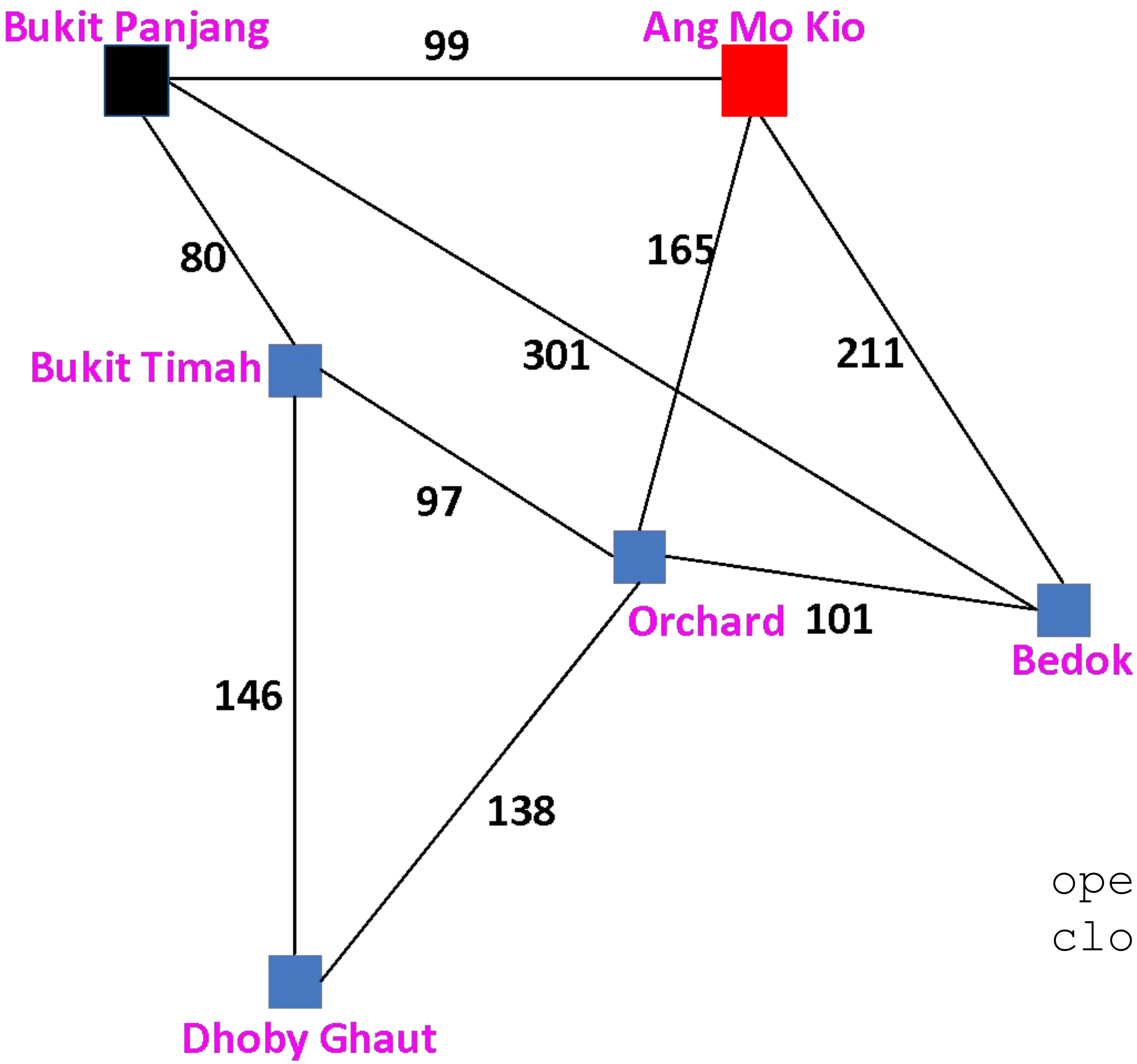



```
BP.parent  = null
AMK.parent = BP
BT.parent  = BP
O.parent   = BT
B.parent   = BP
DG.parent  = BT

BP.g  = 0
AMK.g = 99
BT.g  = 80
O.g   = 177
B.g   = 301
DG.g  = 226

openlist={AMK,O,DG,B}
closedlist={BP,BT}
```

Dijkstra's Search

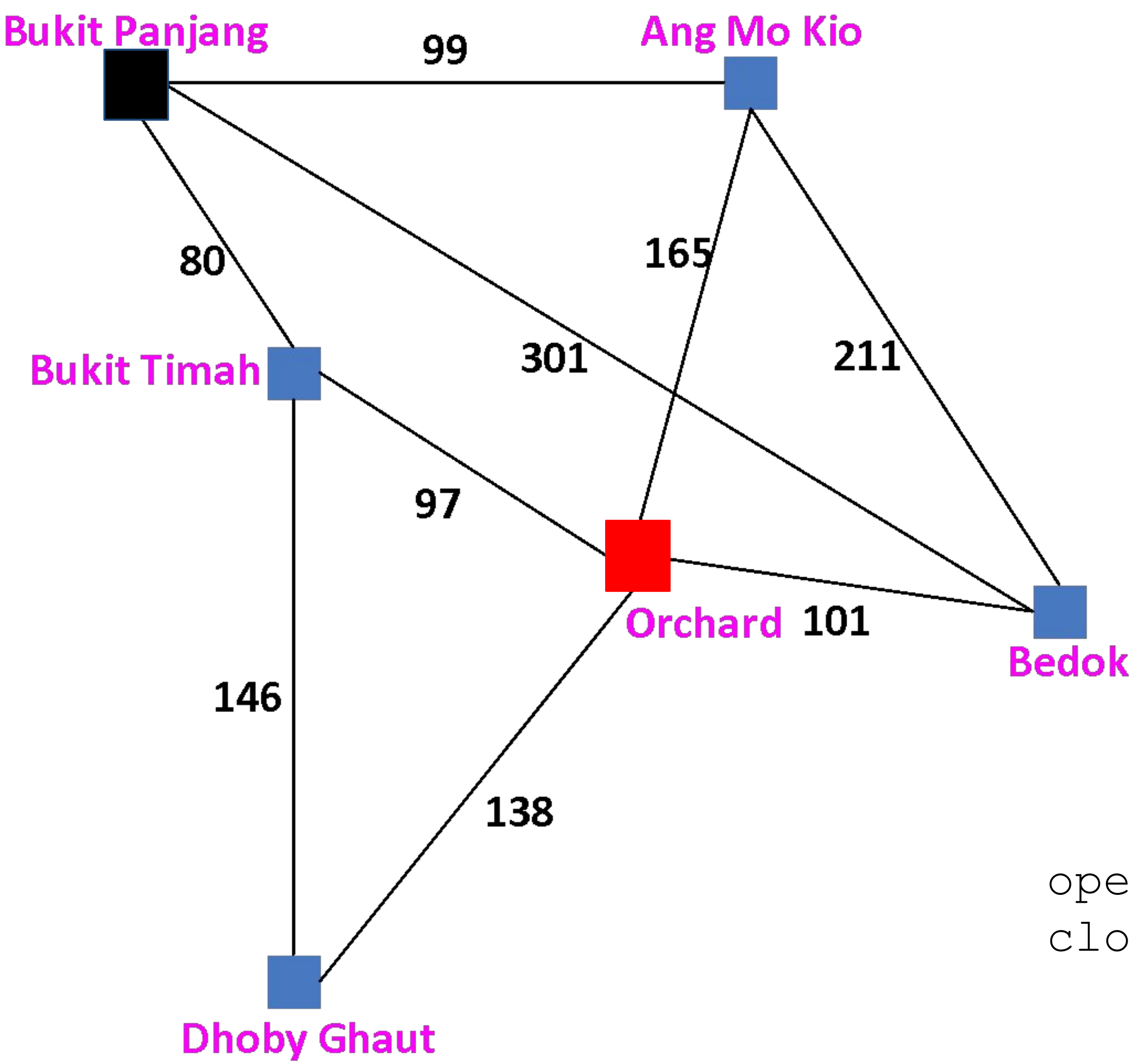



```
BP.parent  = null
AMK.parent = BP
BT.parent  = BP
O.parent   = BT
B.parent   = BP
DG.parent  = BT
```

```
BP.g  = 0
AMK.g = 99
BT.g  = 80
O.g   = 177
B.g   = 301
DG.g  = 226
```

```
openlist={O,DG,B}
closedlist={BP,BT,AMK}
```

Dijkstra’s Search

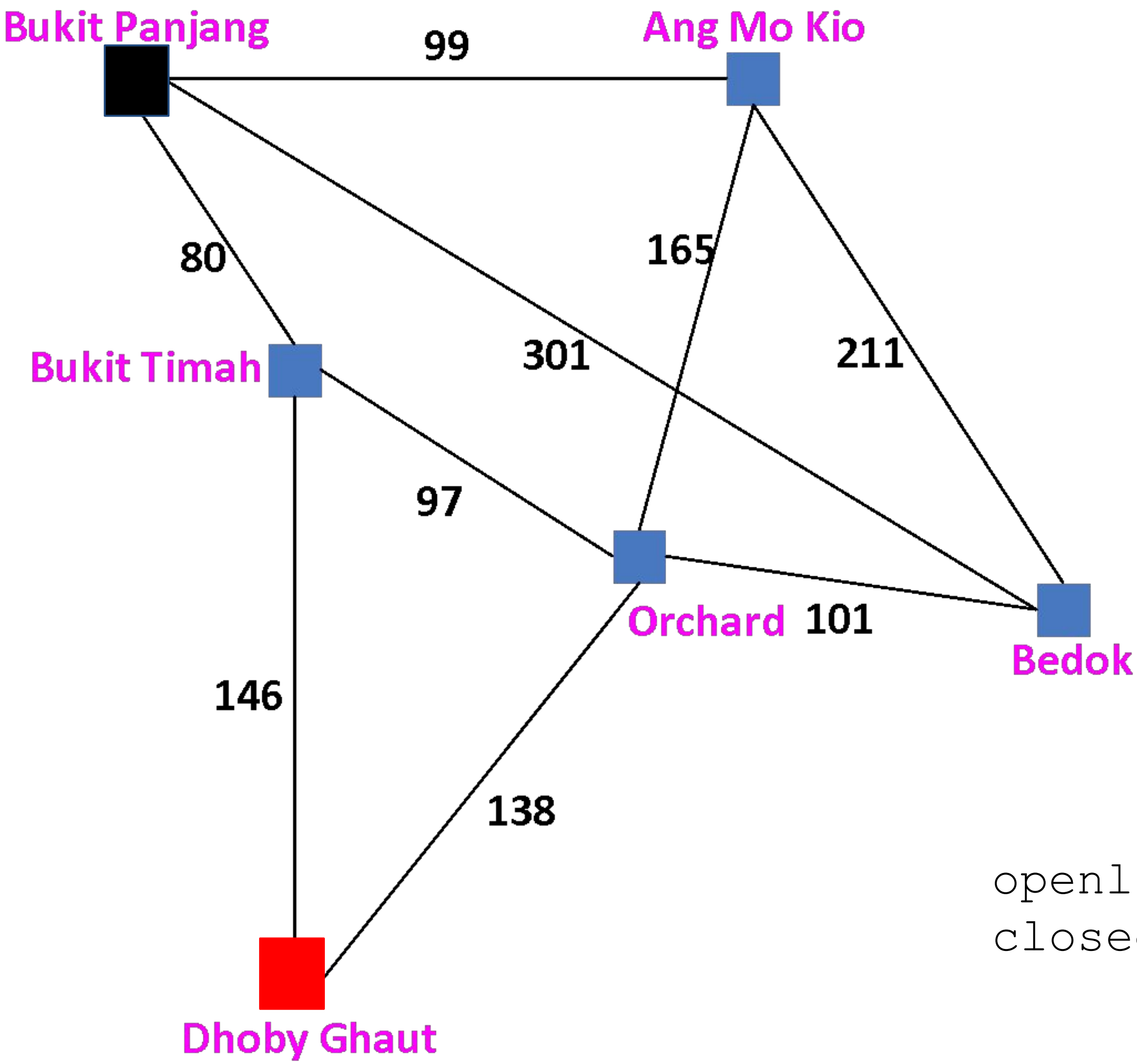



```
BP.parent  = null
AMK.parent = BP
BT.parent  = BP
O.parent   = BT
B.parent   = O
DG.parent  = BT

BP.g  = 0
AMK.g = 99
BT.g  = 80
O.g   = 177
B.g   = 278
DG.g  = 226

openlist={DG,B}
closedlist={BP,BT,AMK,O}
```

Dijkstra's Search

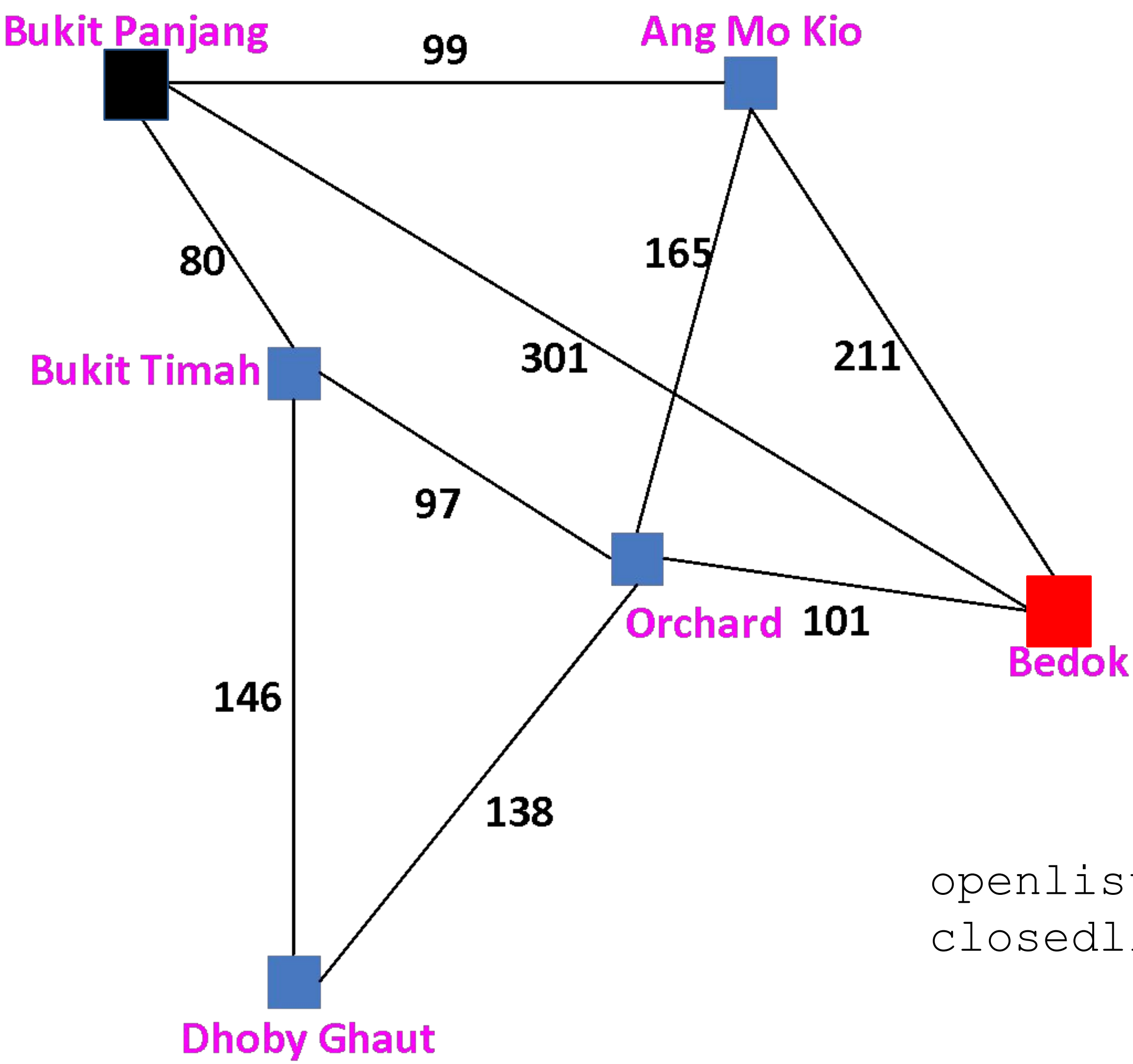



```
BP.parent  = null
AMK.parent = BP
BT.parent  = BP
O.parent   = BT
B.parent   = O
DG.parent  = BT

BP.g  = 0
AMK.g = 99
BT.g  = 80
O.g   = 177
B.g   = 278
DG.g  = 226
```

```
openlist={B}
closedlist={BP,BT,AMK,O,DG}
```

Dijkstra's Search




```
BP.parent  = null
AMK.parent = BP
BT.parent  = BP
O.parent   = BT
B.parent   = O
DG.parent  = BT

BP.g  = 0
AMK.g = 99
BT.g  = 80
O.g   = 177
B.g   = 278
DG.g  = 226
```

```
openlist={}
closedlist={BP,BT,AMK,O,DG,B}
```

Dijkstra’s Search

- If we are only interested in a shortest path between nodes starting and target, we can terminate the search after line

```
closedlist.add(current); // Set as “visited”
```

by testing current and target:

```
if (current == target)
   break;
```

Dijkstra’s Search

- Now we can read the shortest path from source to target by reverse iteration:

```
list path = {};
node = target;
while (node)
{
  path.push(node);
  node = node.parent;
}
path.pop(); // Optional, remove starting point
path.reverse();
```

Dijkstra’s Search

- **Complete?**
  - Yes.

- **Optimal?**
  - Yes, if all states’ actions have non-negative costs.
  - It expands nodes in order of their optimal path cost.

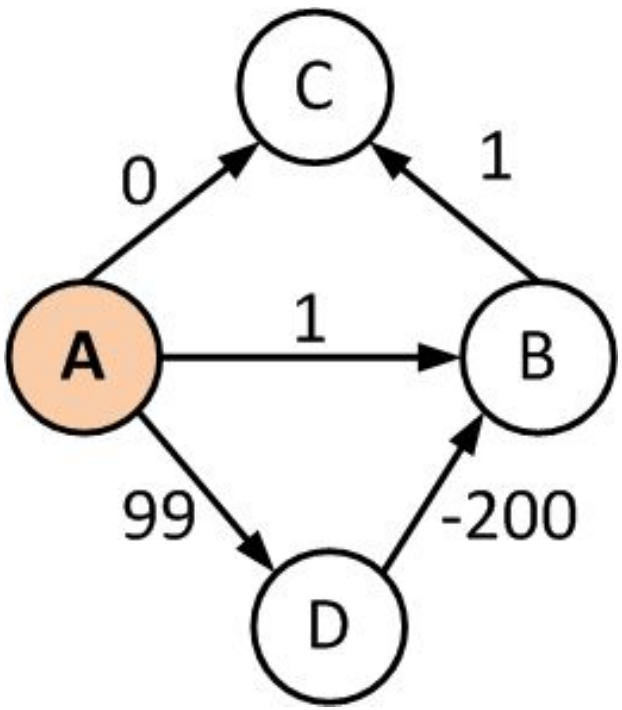
Dijkstra’s Search

- Time & Space complexity in terms of number of vertices n:

  O(n)

Dijkstra's Search. Negative Cost

- Note: it **may** not work on graph with negative cost values

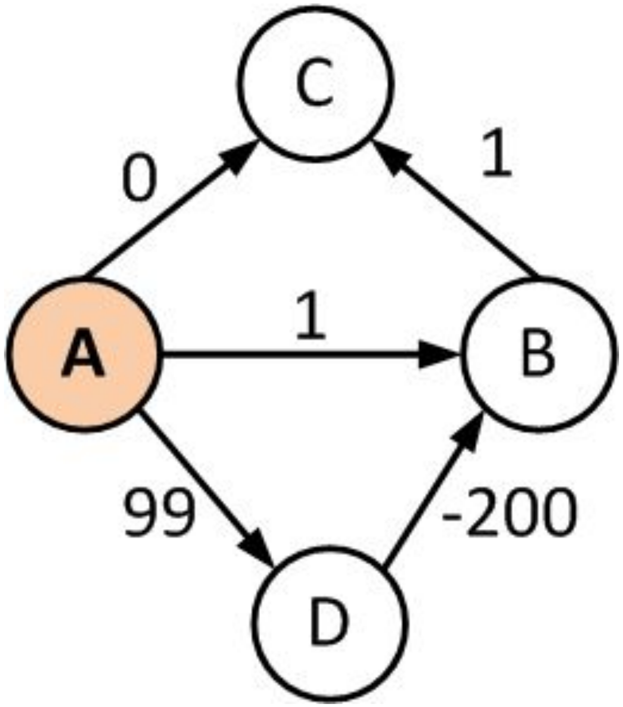



```
A.parent = null
B.parent = null
C.parent = null
D.parent = null

A.g = 0
B.g = +∞
C.g = +∞
D.g = +∞

openlist={A}
closedlist={}
```

Dijkstra's Search. Negative Cost

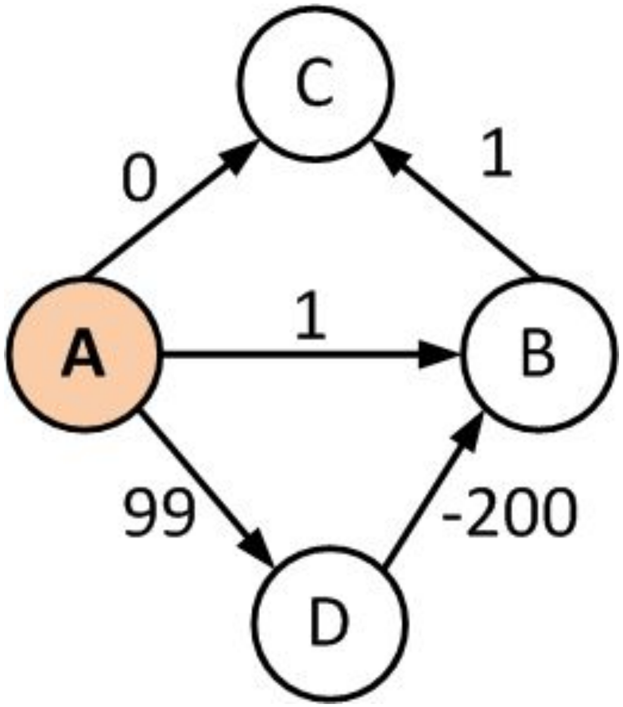



```
A.parent = null
B.parent = A
C.parent = A
D.parent = A

A.g = 0
B.g = 1
C.g = 0
D.g = 99

openlist={C,B,D}
closedlist={A}
```

Dijkstra’s Search. Negative Cost

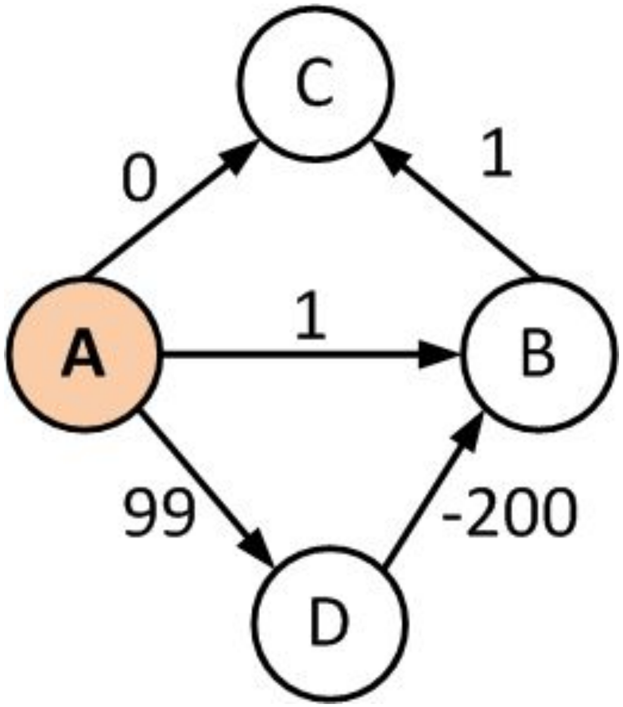



```
A.parent = null
B.parent = A
C.parent = A
D.parent = A

A.g = 0
B.g = 1
C.g = 0
D.g = 99

openlist={B,D}
closedlist={A,C}
```

Dijkstra's Search. Negative Cost

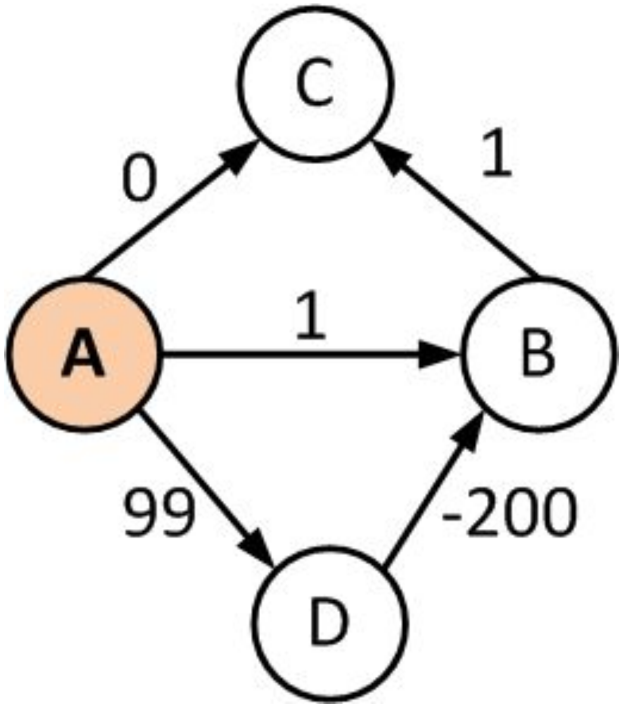



```
A.parent = null
B.parent = A
C.parent = A
D.parent = A

A.g = 0
B.g = 1
C.g = 0
D.g = 99

openlist={D}
closedlist={A,C,B}
```

Dijkstra’s Search. Negative Cost

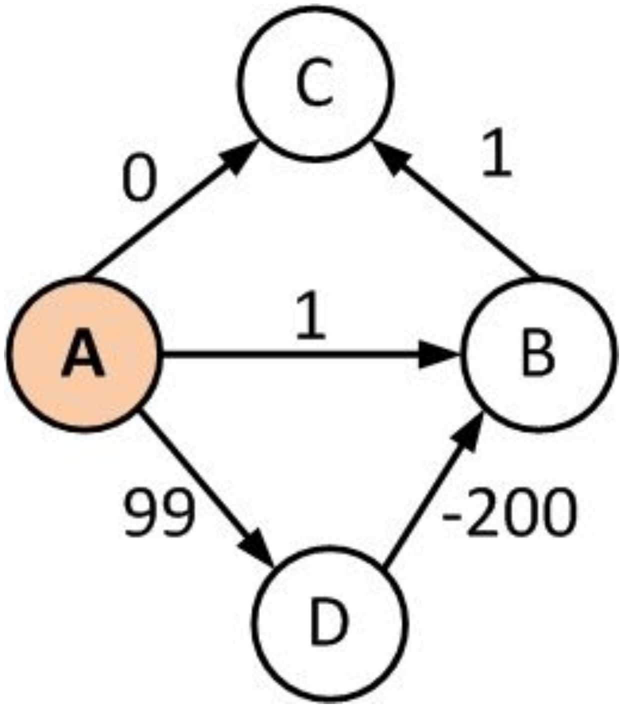



```
A.parent = null
B.parent = D
C.parent = A
D.parent = A

A.g = 0
B.g = -101
C.g = 0
D.g = 99

openlist={}
closedlist={A,C,B,D}
```

Dijkstra's Search. Negative Cost




```
A.parent = null
B.parent = D
C.parent = A
D.parent = A
```

```
A.g = 0
B.g = -101
C.g = 0 (should be -100)
D.g = 99
```

```
openlist={}
closedlist={A,C,B,D}
```

Dijkstra’s Search. Negative Cost

- **Question**: why not reduce computing shortest paths with negative edge lengths to the same problem with non-negative lengths? (by adding large constant to edge lengths)