CS106B
Spring 2016 Cynthia Lee

Section 8 (Week 9) Solution

1. **Dijkstra and A*.**

Dijkstra:
graph : {A:0,/, B:inf,/, C:inf,/, D:inf,/, E:inf,/, F:inf,/, G:inf,/}
pqueue: {A:0}
remove A, process neighbors B/C/E,
update B cost to 4, C to 5, E to 1
graph : {A:0,/, B:4,A, C:5,A, D:inf,/, E:1,A, F:inf,/, G:inf,/}
pqueue: {E:1, B:4, C:5}
remove E, process neighbor F, update F cost to 10
graph : {A:0,/, B:4,A, C:5,A, D:inf,/, E:1,A, F:10,E, G:inf,/}
pqueue: {B:4, C:5, F:10}
remove B, process neighbors C/F, update F cost to 6
graph : {A:0,/, B:4,A, C:5,A, D:inf,/, E:1,A, F:6,B, G:inf,/}
pqueue: {C:5, F:6}
remove C, process neighbors D/G, update D cost to 12, G cost to 8
graph : {A:0,/, B:4,A, C:5,A, D:12,C, E:1,A, F:6,B, G:8,C}
pqueue: {F:6, G:8, D:12}
remove F ... no unprocessed neighbors, no updates.
remove G, process neighbor D, update D cost to 9
graph : {A:0,/, B:4,A, C:5,A, D:9,G, E:1,A, F:6,B, G:8,C}
pqueue: {D:9}
remove D... no unprocessed neighbors, no updates.

Final paths:
A to B: {A, B}, cost=4
A to C: {A, C}, cost=5
A to D: {A, C, G, D}, cost=9
A to E: {A, E}, cost=1
A to F: {A, B, F}, cost=6
A to G: {A, C, G}, cost=8

A*:
graph : {A:0,/, B:inf,/, C:inf,/, D:inf,/, E:inf,/, F:inf,/, G:inf,/}
pqueue: {A:6}
remove A, process neighbors B/C/E,
update B cost to 4, C to 5, E to 1
graph : {A:0,/, B:4,A, C:5,A, D:inf,/, E:1,A, F:inf,/, G:inf,/}
pqueue: {E:3, B:9, C:9}
remove E, process neighbor F, update F cost to 10
graph : {A:0,/, B:4,A, C:5,A, D:inf,/, E:1,A, F:10,E, G:inf,/}
pqueue: {B:9, C:9, F:11}
remove B, process neighbors C/F, update F cost to 6
graph : {A:0,/, B:4,A, C:5,A, D:inf,/, E:1,A, F:6,B, G:inf,/}
pqueue: {F:7, C:9}
remove F ... no unprocessed neighbors, no updates.
remove C, process neighbors D/G, update D cost to 12, G to 8
graph : {A:0,/, B:4,A, C:5,A, D:12,C, E:1,A, F:6,B, G:8,C}
pqueue: {G:8, D:15}
remove G… destination vertex found! Use prev pointers to build path.

Final path:
A to G: {A, C, G}, cost=8

2. **Kruskal**.
   Edges: G-A, G-H, I-C, A-B, B-C, G-D (note that these are undirected, e.g., G-A is the same as A-G).
   Cost: 26

3. **isCyclic.**

```
bool isCyclic(BasicGraph& graph) {
    graph.resetData();
    Map<Node*, string> mark;
    for (Node* v : graph.getNodeSet()) {
        mark[v] = "UNVISITED";
    }
    for (Node* v : graph.getNodeSet()) {
        if (isCyclicHelper(graph, mark, v)) {
            return true;
        }
    }
    return false;
}

bool isCyclicHelper(BasicGraph& graph, Map<Node*, string>& mark, Node* v) {
    mark[v] = "PARTIAL";
    for (Arc* edge : graph.getArcSet(v)) {
        if (!edge->visited) {
            edge->visited = true;
            Node* neighbor = edge->finish;
            if (mark[neighbor] == "PARTIAL") {
                return true;
            } else if (mark[neighbor] == "UNVISITED") {
                if (isCyclicHelper(graph, mark, neighbor)) {
                    return true;
                }
            }
        }
    }
    mark[v] = "VISITED";
    return false;
}
```

4. **Inheritance and polymorphism**.

```
Lettuce* var1 = new Bacon();
Bacon* var2 = new Mayo();
Lettuce* var3 = new Hamburger();
Bacon* var4 = new Hamburger();
Lettuce* var5 = new Lettuce();
```

a.	var1->m1();	cout << endl;	// L1 / L2 / B1
b.	var1->m2();	cout << endl;	// L2
c.	var1->m3();	cout << endl;	// COMPILER ERROR
d.	var2->m1();	cout << endl;	// L1 / H2 / L2 / B1
e.	var2->m2();	cout << endl;	// H2 / L2
f.	var2->m3();	cout << endl;	// M3 / L1 / H2 / L2 / B1
g.	var2->m4();	cout << endl;	// COMPILER ERROR
h.	var3->m1();	cout << endl;	// L1 / H2 / L2 / B1
i.	var3->m2();	cout << endl;	// H2 / L2
j.	var4->m2();	cout << endl;	// H2 / L2
k.	var4->m3();	cout << endl;	// B3
l.	var4->m4();	cout << endl;	// COMPILER ERROR
m.	((Bacon*) var1)->m1();	cout << endl;	// L1 / L2 / B1 // (unchanged behavior)
n.	((Bacon*) var1)->m3();	cout << endl;	// B3 // (cast makes it compile)
o.	((Mayo*) var5)->m3();	cout << endl;	// CRASH // (cast too far down)
p.	((Lettuce*) var4)->m3();	cout << endl;	// COMPILER ERROR
q.	((Hamburger*) var2)->m4();	cout << endl;	// M4
r.	((Mayo*) var2)->m4();	cout << endl;	// M4