YEAH!

Trailblazer

Sahil Chopra - 3.03.2016
Adapted from SLs Brendon Go & Rishi Bedi

Graph Algorithms

- Depth First Search - Find a Path
- Breadth First Search - Finds the Shortest Path
- Dijkstra’s Algorithm - Finds Path of Least Cost
- A* Search - Finds Path of Least Cost w/ Heuristics → Speed Boost
- Kruskals Algorithm - Construct Minimum Spanning Trees

Depth First Search (DFS) - Recursive

```
function dfs(v1, v2){
    dfs(v1, v2, emptyPath)
}

function dfs(v1, v2, path) {
    add v1 to path and mark as visited
    if v1 == v2
        return true // we found a path
    for each unvisited neighbor n of v {
        if dfs(n, v2, path) finds a path
            we are done
    }
    remove v1 from path
}
```

Breadth First Search (BFS) - Iterative

```
function bfs(v1, v2) {
    enqueue(v1)
    mark v1 as visited
    while (q is not empty){
        dequeue vertex v from q
        if v1 == v2
            yay!
        for each unvisited neighbor n of v {
            mark n as visited and add it to q
        }
    }
}
```

Dikstra's Algorithm - Iterative

```
function dijkstras(v1, v2){
    initialize every node to have cost ∞
    set v1's cost to 0
    enqueue v1 with priority 0
    while (pq is not empty) {
        v = dequeue most urgent element
        mark v as visited
        if v == v2, stop
        foreach unvisited neighbor n of v {
            cost = v's cost + weight of edge from n to v
            if cost < n's cost
                set n's cost to cost and n's previous pointer to v
            if n is in pq
                update its priority to be cost
            else
                enqueue n with priority = cost
        }
    }
}
Follow previous pointers to reconstruct path
```

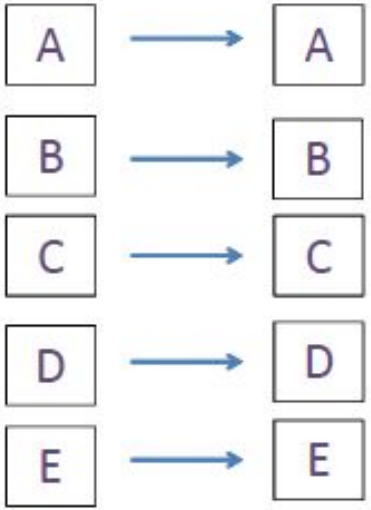
A* - Variant of Djikstra's

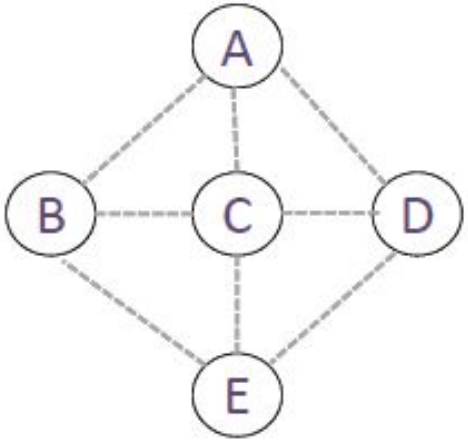
```
function dijkstras(v1, v2){
    initialize every node to have cost ∞
    set v1's cost to 0
    enqueue v1 with priority heuristic(v1, v2)
    while (pq is not empty) {
        v = dequeue most urgent element
        mark v as visited
        if v == v2, stop
        foreach unvisited neighbor n of v {
            cost = v's cost + weight of edge from n to v + heuristic(v, v2)
            if cost < n's cost
                set n's cost to cost and n's previous pointer to v
            if n is in pq
                update its priority to be cost
            else
                enqueue n with priority = cost
        }
    }
}
```

Kruskal's

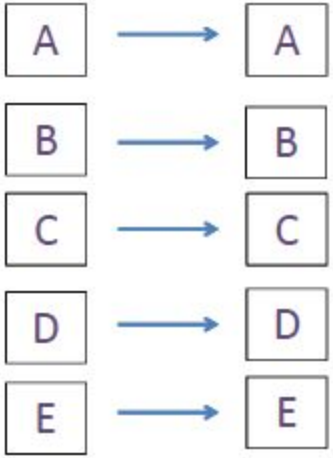
```
function kruskals(graph){
    put all nodes into their own cluster
    create a PQ of the edges order by their cost
    while pq is not empty {
        remove the edge
        if edge's endpts are not in the same cluster {
            choose this edge
            merge the two clusters
        }
    }
}
```

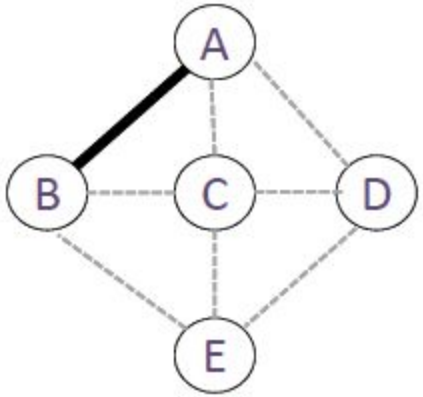
Clustering



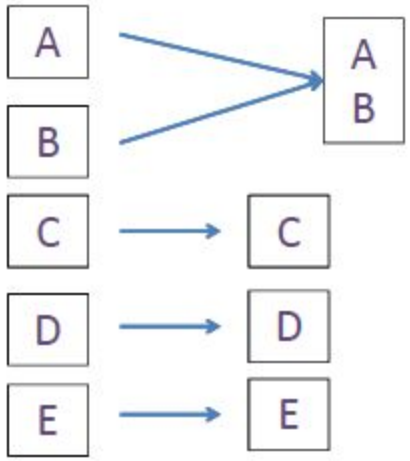


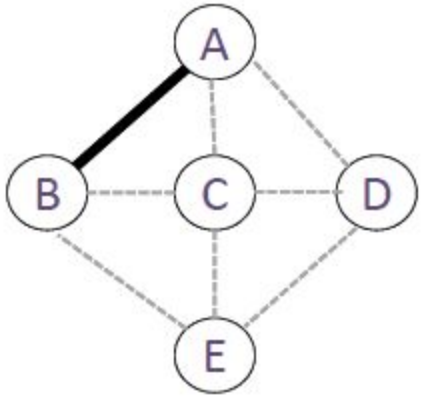
Clustering





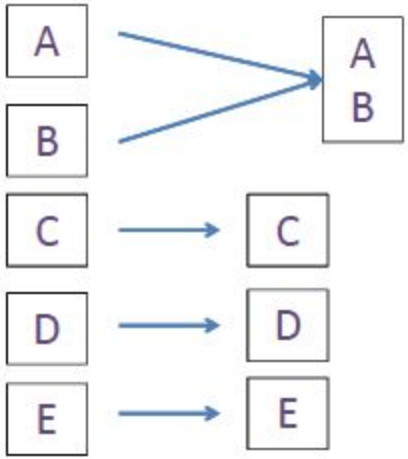
Clustering

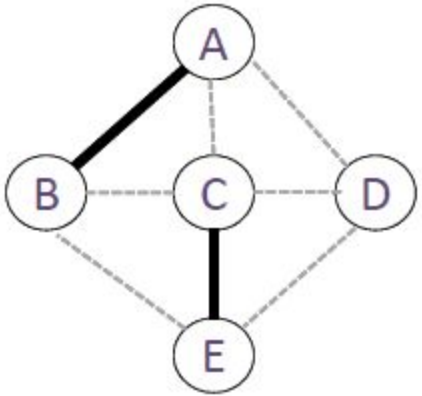




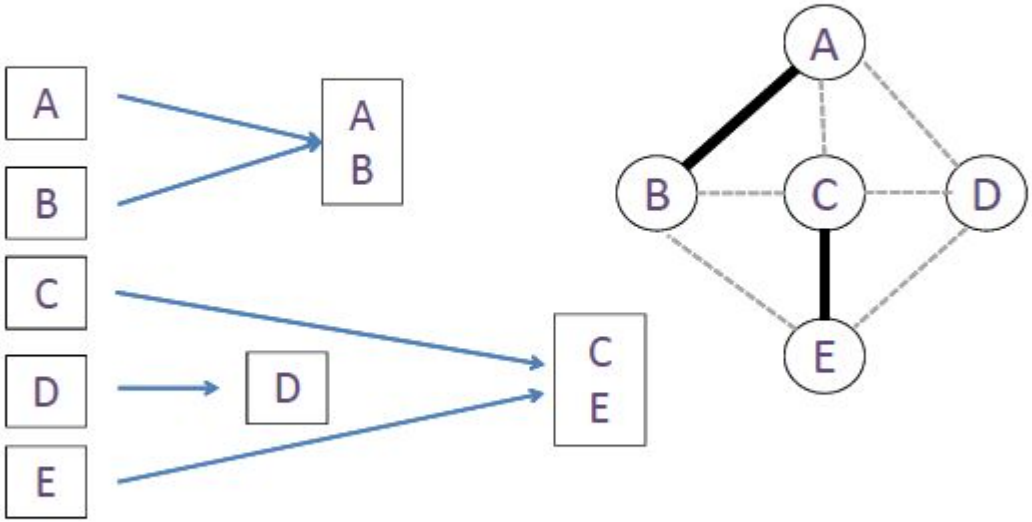
- Merge clusters

Clustering



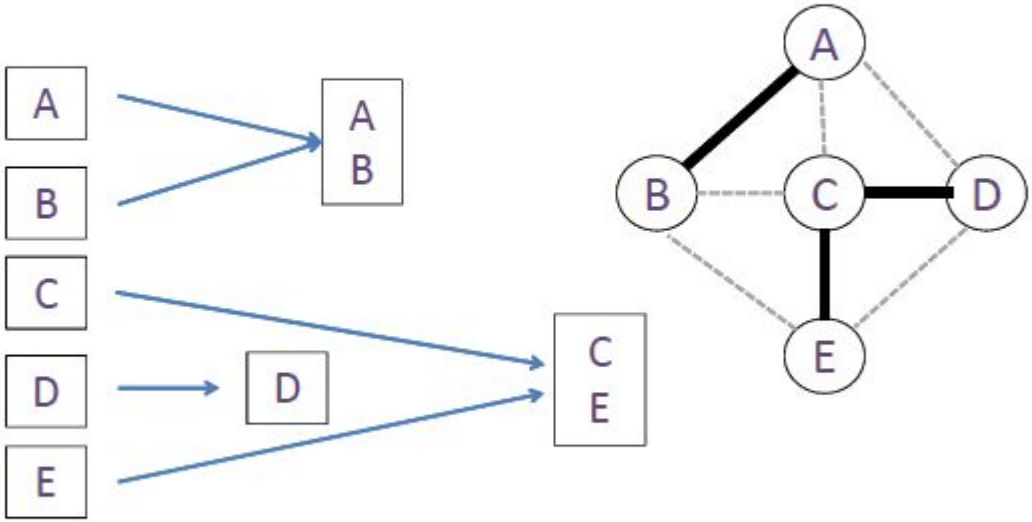


Clustering

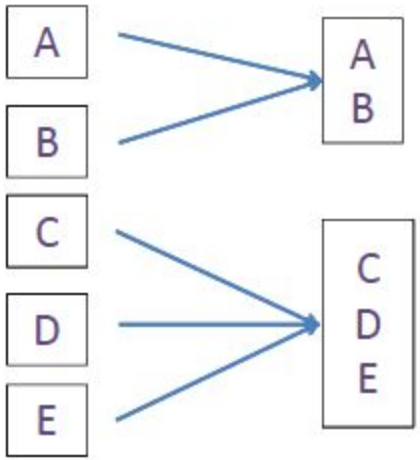


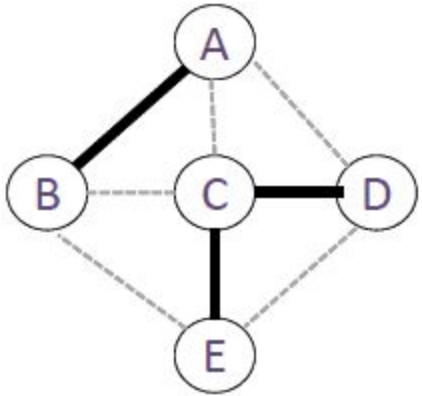
- Merge clusters

Clustering



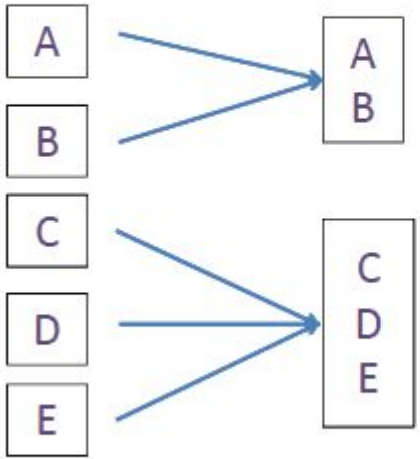
Clustering

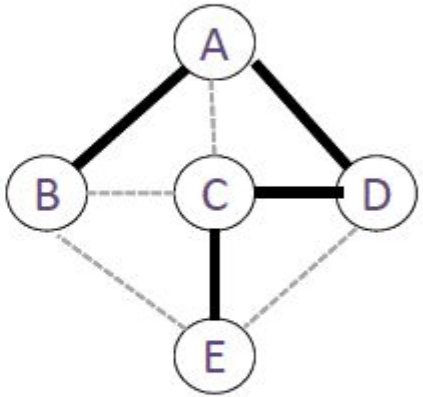




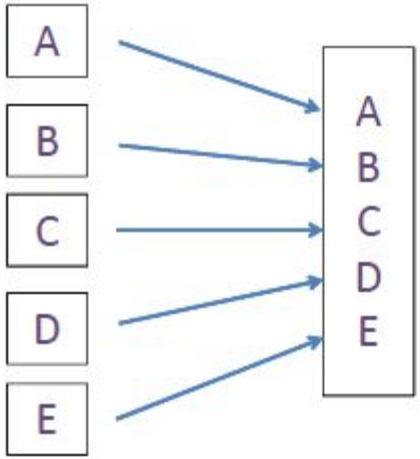
- Merge clusters

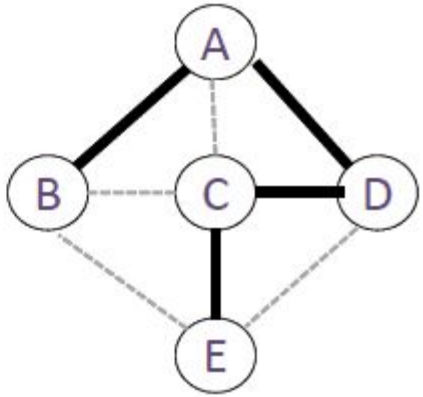
Clustering





Clustering





- Merge clusters

Relevant Code

```
graph.resetData()

graph.getEdge(v1,v2)

for(Vertex* neighbor: graph.getNeighbors(vertex))

for(Vertex* vertex : graph.getVertexSet())

for(Edge* edge: graph.getEdgeSet())

heuristicFunction(from, to)
```

Relevant Code

```
Vertex* vertex;

vertex->cost

vertex->visited

vertex->setColor(color)

vertex->getColor()

POSITIVE_INFINITY

Edge* edge

edge->cost
```

FAQ

How can I tell if I did it right?

DFS: finds any path

BFS: finds any path same length as sample

Dijkstras/A*: finds any path with same cost as sample

Kruskals: Maze is connected