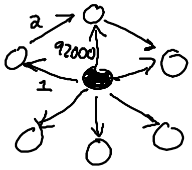
Dijkstra's Algorithm

Find, for a given source vertex, the shortest path to every destination.
path w/ least total weights of edges

```
Function sssp(g, src):
    frontier ← new MinHeap
    frontier.insert(0, src)
    cost ← new Dictionary
    cost.insert(src, 0)
    While frontier is not empty:
        currentCost ← frontier.peekPriority()
        current ← frontier.remove()
        For each neighbor of current:
            If neighbor is not a key in cost:
                cost.insert(neighbor, currentCost + weight)
                frontier.insert(currentCost + weight, neighbor)
            Else If cost.get(neighbor) > currentCost + weight:
                cost.update(neighbor, currentCost + weight)
                frontier.insert(currentCost + weight, neighbor)
            EndIf
        EndFor
    EndWhile
    Return cost
End Function
```

When a vertex is removed: if this is the first time we've explored it, it will have the lowest cost possible to reach it.

$O(E)$



$f(n)$ is $O(g(n))$

$g(n) \leq h(n)$

$f(n)$ is $O(h(n))$

$E \leq V^2$

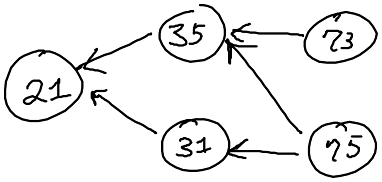
$O(E \cdot \log E)$

$O(E \cdot \log V^2) = O(E \cdot 2 \log V)$

$= O(E \cdot \log V)$

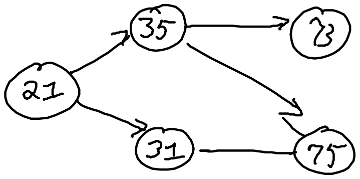
$O(V^2)$

Topological Sort

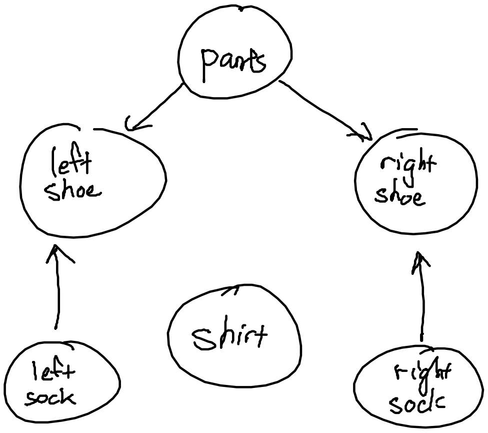


vertex – course
edge – "depends on"

dependency graph



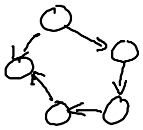
vertex – course
edge – "needed by"



Topological sort of a graph:
a sequence of vertices such that, if the edge $\langle V_1, V_2 \rangle$ is in the graph, V_1 appears before V_2 in the sequence.

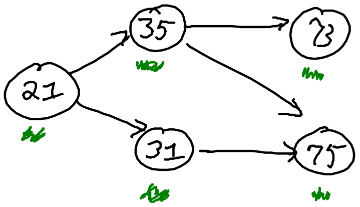
```
While graph is not empty:
    Find a vertex with in-degree of zero
    Add it to the result list
    Remove it (and all of its outgoing edges) from graph
```

21, 31, 35, 75, 73
21, 35, 73, 31, 75

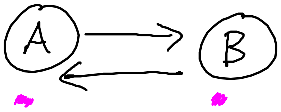


```
Function Toposort(g):
    For each vertex in g
        Explore(g, vertex)
    EndFor
EndFunction
Function Explore(g, vertex):
    If vertex is finished, return
    If vertex is exploring, !!
    Mark vertex as exploring
    For each neighbor:
        Explore(g, neighbor)
    End For
    Add vertex to the beginning of answer
    Mark vertex as finished
EndFunction
```

- ● Unexplored
- ● Exploring
- ● Finished



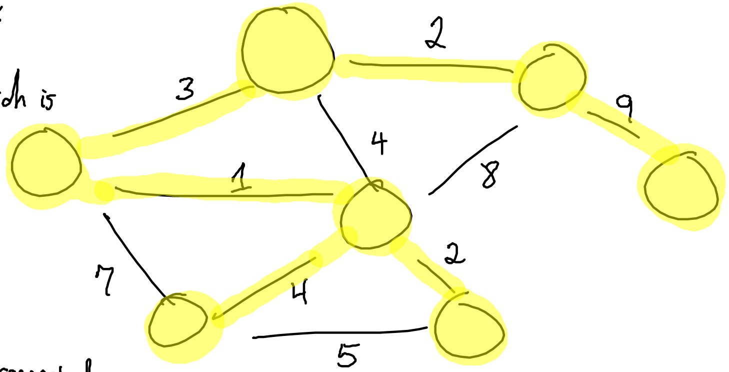
21, 31, 35, 75, 73



Minimum Spanning Tree

Prim's Algorithm

- Start at <u>any</u> vertex
- Add an edge which is cheapest among those which connect a new vertex into existing network
- Repeat until network connected


2
3
9
4
8
1
7
4
2
5