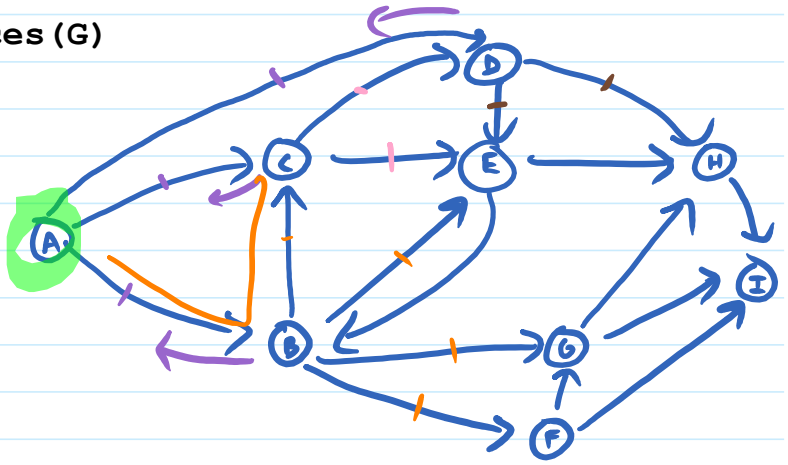
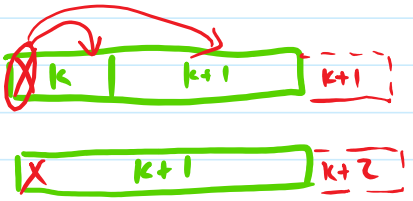
Breadth-First Search

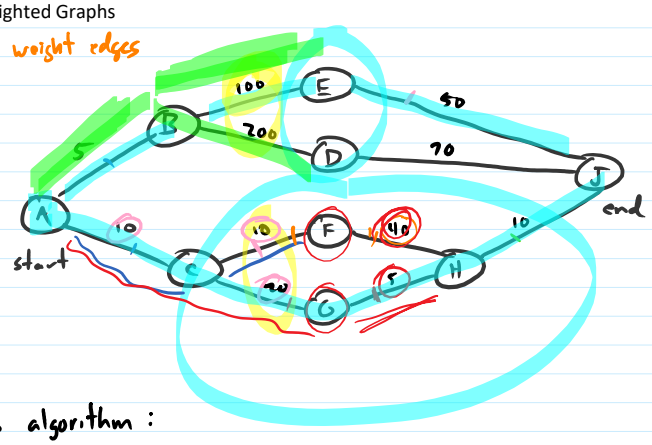
```
BFS(G,s)

for each vertex u in graph_vertices(G)
   color[u] <- WHITE
   d[u] <- infinity
   pred[u] <- NIL

color[s] <- GRAY
d[s] <- 0
pred[s] <- NIL

Q <- [s]

while not queue_is_empty(Q)
   u <- queue_dequeue(Q)
   for each v in graph_adjacent(u)
      if color[v] = WHITE
         color[v] <- GRAY
         d[v] <- d[u] + 1
         pred[v] <- u
         queue_enqueue(Q, v)
   color[u] <- BLACK
```





Invariant: verts on queue are in order of ↑ distance with those @ end at most one edge further than those at start

assume no neg weight edges



Dijkstra's algorithm:

```
Dijkstra(G,s)

Q <- pqueue_create()
for each vertex u in graph_vertices(G)
   d[u] <- infinity
   pred[u] <- NIL
   pqueue_add(Q, u, d[u])

d[s] <- 0
pred[s] <- NIL
pqueue_update(Q, s, 0)

while not pqueue_is_empty(Q)
   u <- pqueue_extract_min(Q)
   for each v in graph_adjacent(u)
      if pqueue_contains(Q, v) and d[u] + graph_weight(G, u, v) < d[v]
         d[v] <- d[u] + graph_weight(G, u, v)
         pred[v] <- u
         pqueue_update(Q, d[v])
```

adj list

O(n+m)

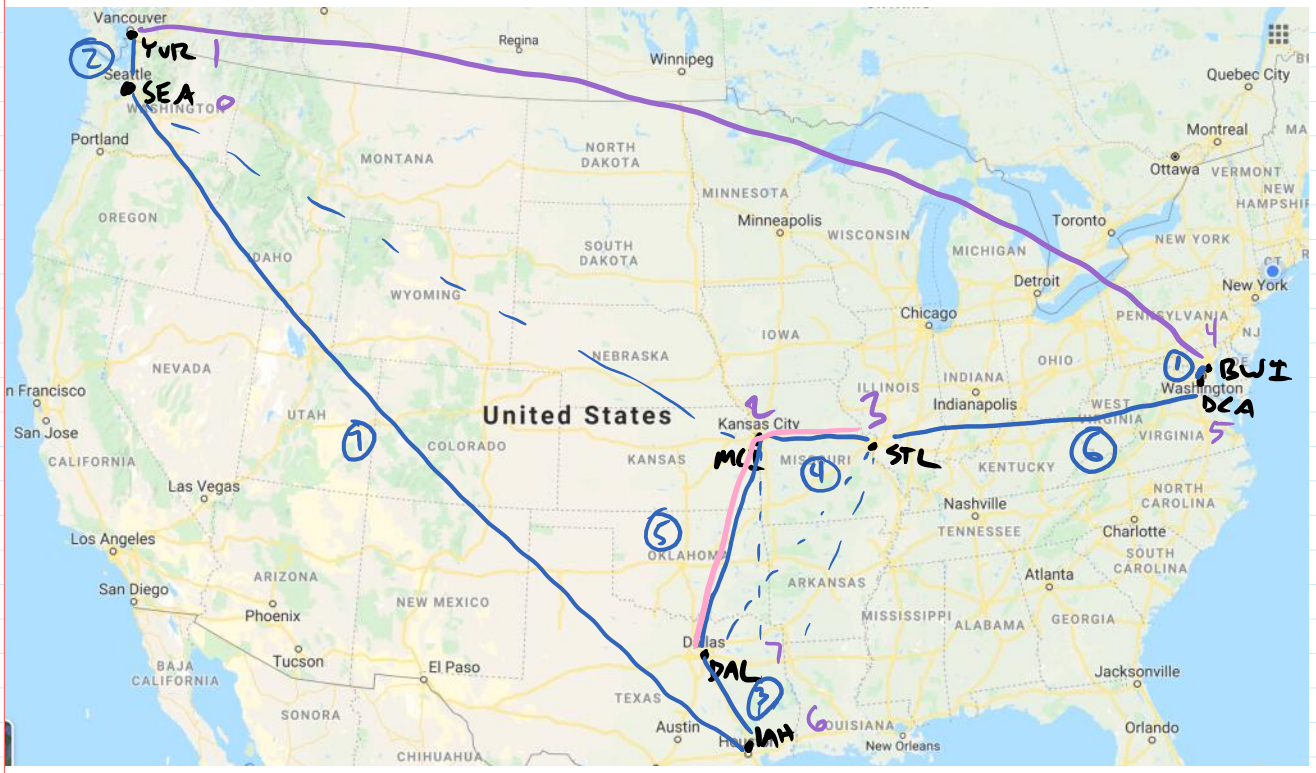
O(n) each, O(n) times → O(n²) total

O(n²) overall

O(1) each, O(m) time → O(m) total

Priority Queue: items + priorities
add(item, priority)
extract_min(): removes item w/ lowest priority
update(item, priority): changes priority of item

	unsorted array	heap
add	O(1)	O(log n)
extract_min	O(n)	O(log n)
update	O(1)	O(log n)



city/vertex indices

order edges added

for each pair of cities in order of increasing distance
if not connected yet and degree both $\in 1$
add edge to tour

BIS

find vertices u, v with degree 1
find path $u \rightsquigarrow v$
add edge (v, u)