**Final:**

1. Multi-choice (for same root, different queue order, bfs tree may various)
2. Graph problem as E3E4, proof, spanning tree, coloring
   1. DFS: topological sorting. no code
   2. BFS: know two properties. code
   3. Max-flow, know how to do it, no algorithm
3. Algorithm design
   1. Graph
   2. Graph or something else
   3. Is this graph connected, write code (run BFS, see if any node is un-visited)
4. Recursive, non-recursive back-tracking
5. Know all four Spanning Tree algorithm, how to write code. Note: slides algorithm not correct, check the book.
6. **Kruskal**: choose a min weight safe edge, one by one. code
7. **Prim’s**: mark start node, find a min safe edge around marked nodes, mark new node, looping. code
8. **Dijkstra**: mark start node, set min path value to 0. Check adj. nodes, update min path value for them, pick one adj. node with min path value, looping. code

Eurlerian path/cycles

Path/cycle that visits all vertices

Edges at most once (may exist edge that has never been visited)

Complete graph with n vertices, must has Hamiltonian cycle. E.g.: 1, 2, 3, 4, …, n, 1

**N-Queen problem**

NQ(Q[1..n])

r = 1;

while(true)

if (r == n+1) print Q, return true

if (r == 0) return false

for cols from Q[r] to n // check if there is a position for the current checking row

findLegal = false

for rows from 1 to r-1

checkLegal() // TODO

if findLegal

Q[r] = cols

r++

break

if findLegal

break

else if (cols == n)

Q[r] = 1

r--

Q =

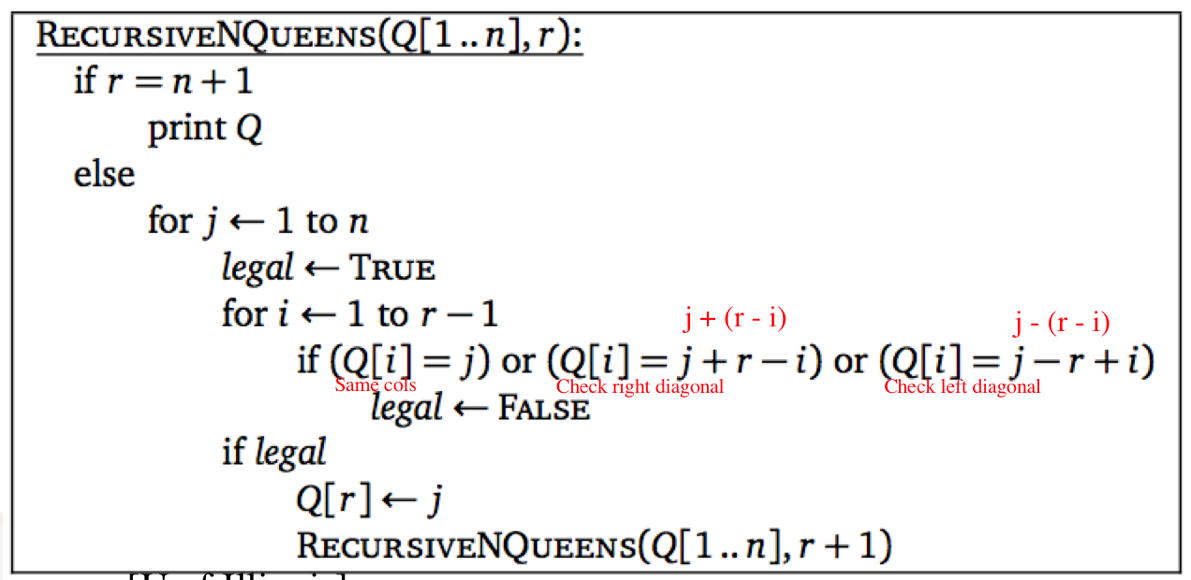
{1, 1, 1, 1} 🡪 {1, 3, 1, 1} 🡪 {1, 3, ?, 1}

🡪 {1, 4, 1, 1} 🡪 {1, 4, 2, 1} 🡪 {1, 4, 2, ?}

🡪 {1, 4, ?, 1}

🡪 {1, ?, 1, 1}

{2, 1, 1, 1} -> {2, 4, 1, 1} 🡪 {2, 4, 1, 1} 🡪 {2, 4, 1, 3}



**K-coloring problem**

colorGraph(G(V, E), k, c[1, …, n]) { // c[] is the list of color for all vertices

c = {1, 1, …, 1}

i = 1 // i is the back tracking variable, **IMPORTANT**

while(true) {

while(c[i] is invalid && c[i] ≤ k)

c[i]++

if (c[i] > k) { // out of color, back track

c[i] = 1 // reset current color

i-- // back tracking

if (i == 0) return false

c[i]++

else

i++

if (i > n) return c

**G(V, R), BFS-tree B**

**Show that a BFS tree has the property that e in E, e not in B, two vertices of e either on the same level or consecutive level of the tree**

Prove by contradiction, let e in E, e not in B, e connects nodes u and v such that level(u) = 1, level(v) – level(u) > 1.

Level(u) < level(v) means u is visited first. v will be added to queue as other children of u, it means level(v) = level(uchildren), means level(v) ≤ 1.

So level(v) – level(u) ≤ 1.

Contradiction

**Maze exit**

Maze\_exit(G, s, t, p[1..n])

u = s

r = 1

for each V.visited = false

A = empty set // contains sets of nodes, represent the exit path for the maze

// it’s a first-in-last-out set

Set all value in P to 1 // Define P[1..n], P[r] = k represent an edge (u, v) s.t. u is the rth node in

// A, and v is the kth neighbor in G.V.Adj(u)

While (true)

If (r == 0) // cannot find exit path

return false

k = |G.V.Adj(u)| // # of neighbor for u

if (p[r] == k) // back track

p[r] = 1

r--

u = v.π

v.visited = false

A.pop\_back()

else

for i from p[r] to k

v = G.V.Adj(u)[i]

if (v.visited == true or G.V.Adj(v) is empty)

p[r] = i

continue

if (v == t)

print A

return true

else

v.visited = true

v.π = u

A.push\_back(v)

p[r] = i

r++

u = G.V.Adj(u)[i]

**Midterm question:**

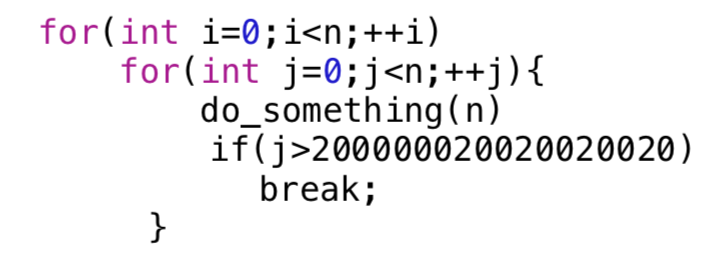
f(n) = n \* log(n) is not O(n)

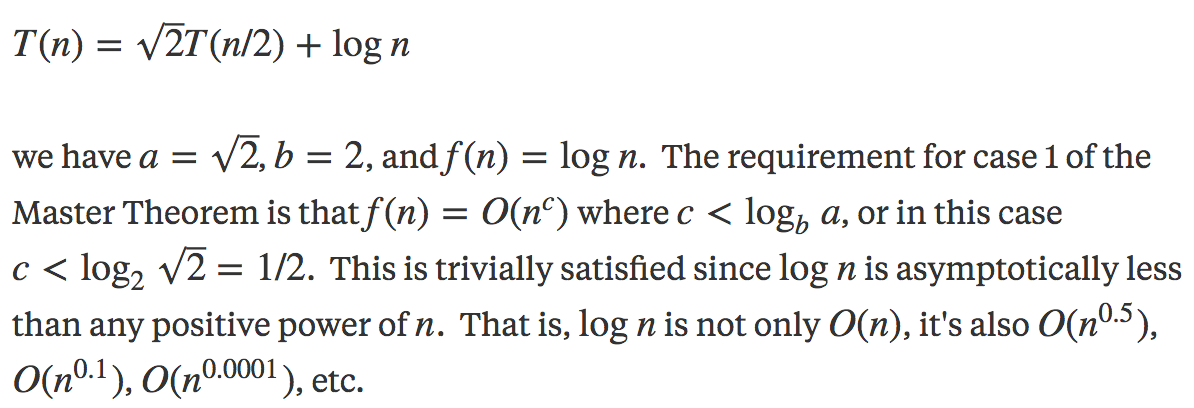
show nlogn ≤ cn

choose n = max(n0, 2c + 1)

prove or disprove O(kn) = O(n)

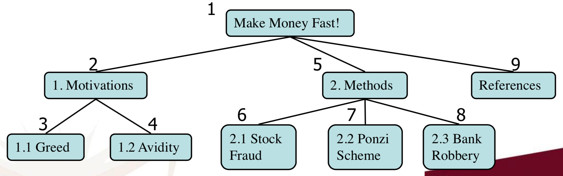
DO NOT LET K = 1 OR K = c, NOT ALLOWED





T(n) = 2T(n/2) + nlgn, neither case 1, 2, nor 3.

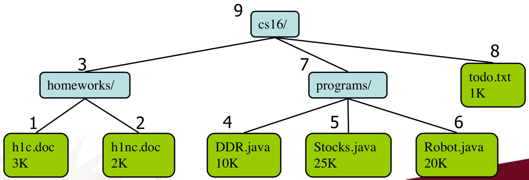
**Self-review**

|E| ≤ |V| \* |V-1| / 2, v-s is max degree for each vertex

A graph is planar iff it does not contain subdivision of K33 or K5

Euler formula

v – 3 + f = 2

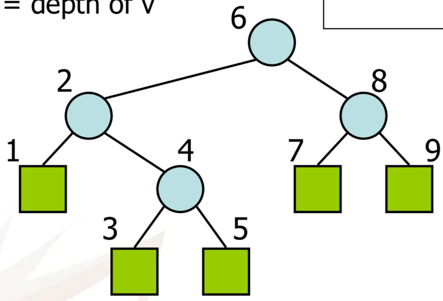
 e ≤ 3v – 6

f ≤ 2v – 4

Tree

Depth of a node: # of ancestors

Pre-order traversal: a node is visited before its descendants

 Post-order traversal: a node is visited after its descendants

In-order traversal: a node is visited its left subtree and before its right subtree

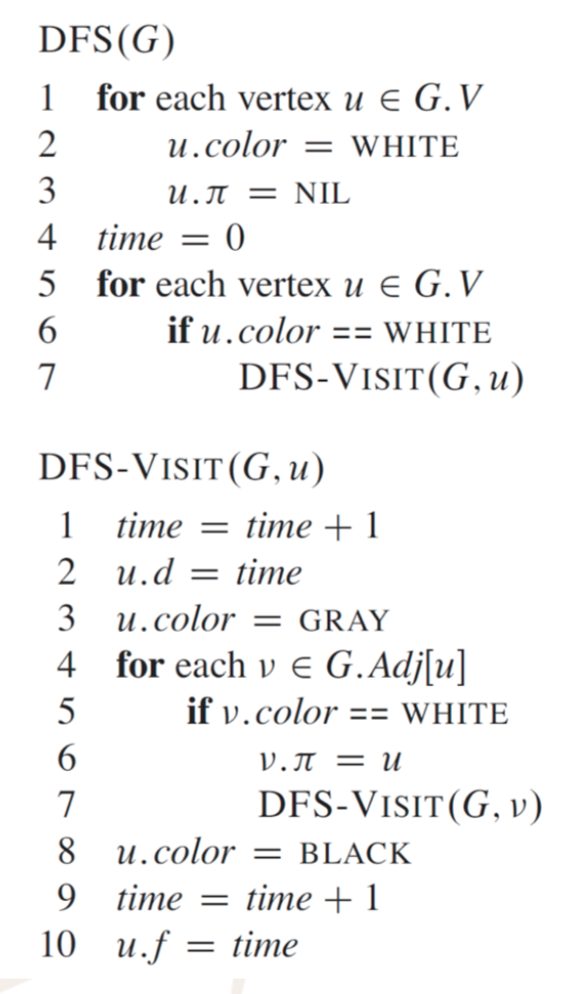
BFS

Assume the graph is connected

u.d is the shortest distance (path) between u and s

the path from s to u is unique

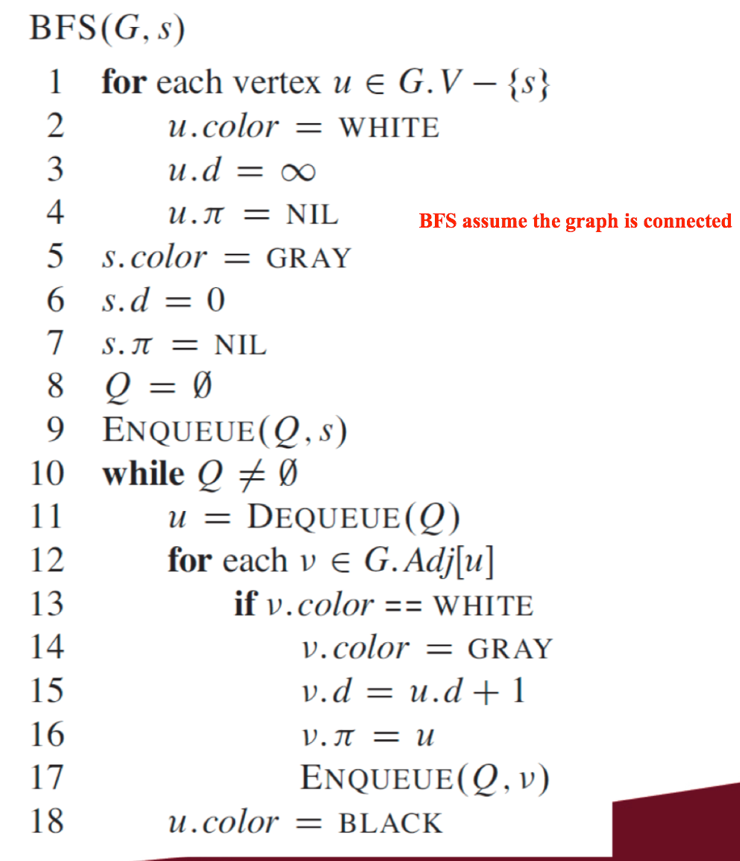
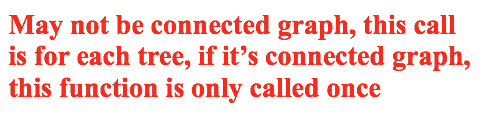
test bi-partitiness (proof no odd cycle)

DFS

No need to be connected

Topological sorting (task scheduling)

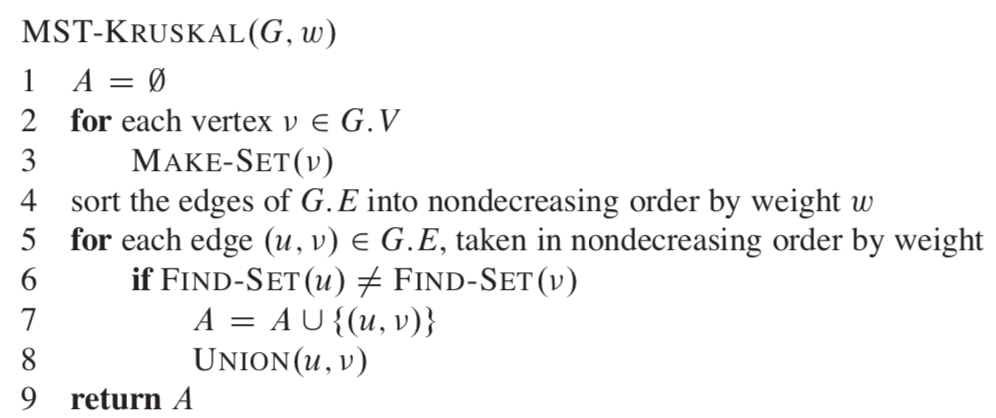
Directed acyclic graph



Minimum spanning trees

Kruskal (O(ElogE), O(ElogV))

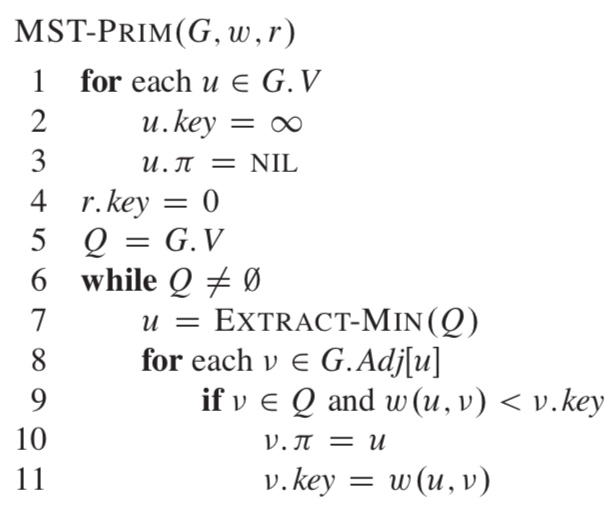
Each step current subtree is a minimum spanning tree of the subgraph

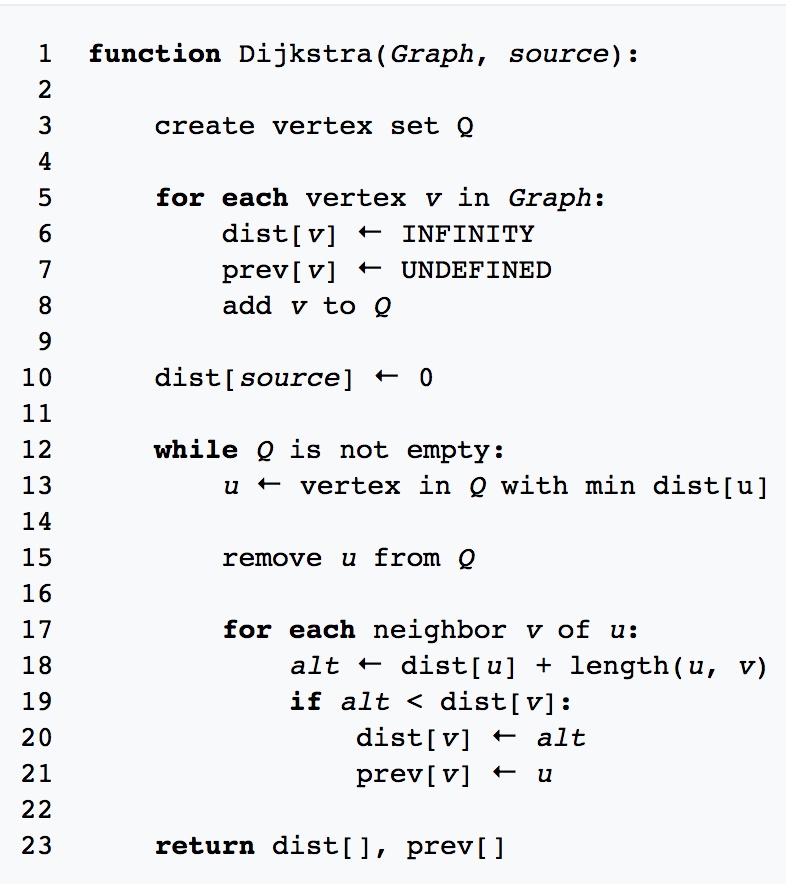


Prim’s (O(VlogV + ElogV) = O(ElogV))

Each step current subtree is a minimum spanning tree of the subgraph





 Dijkstra (O(V^2 + E), O(VlogV + E))

