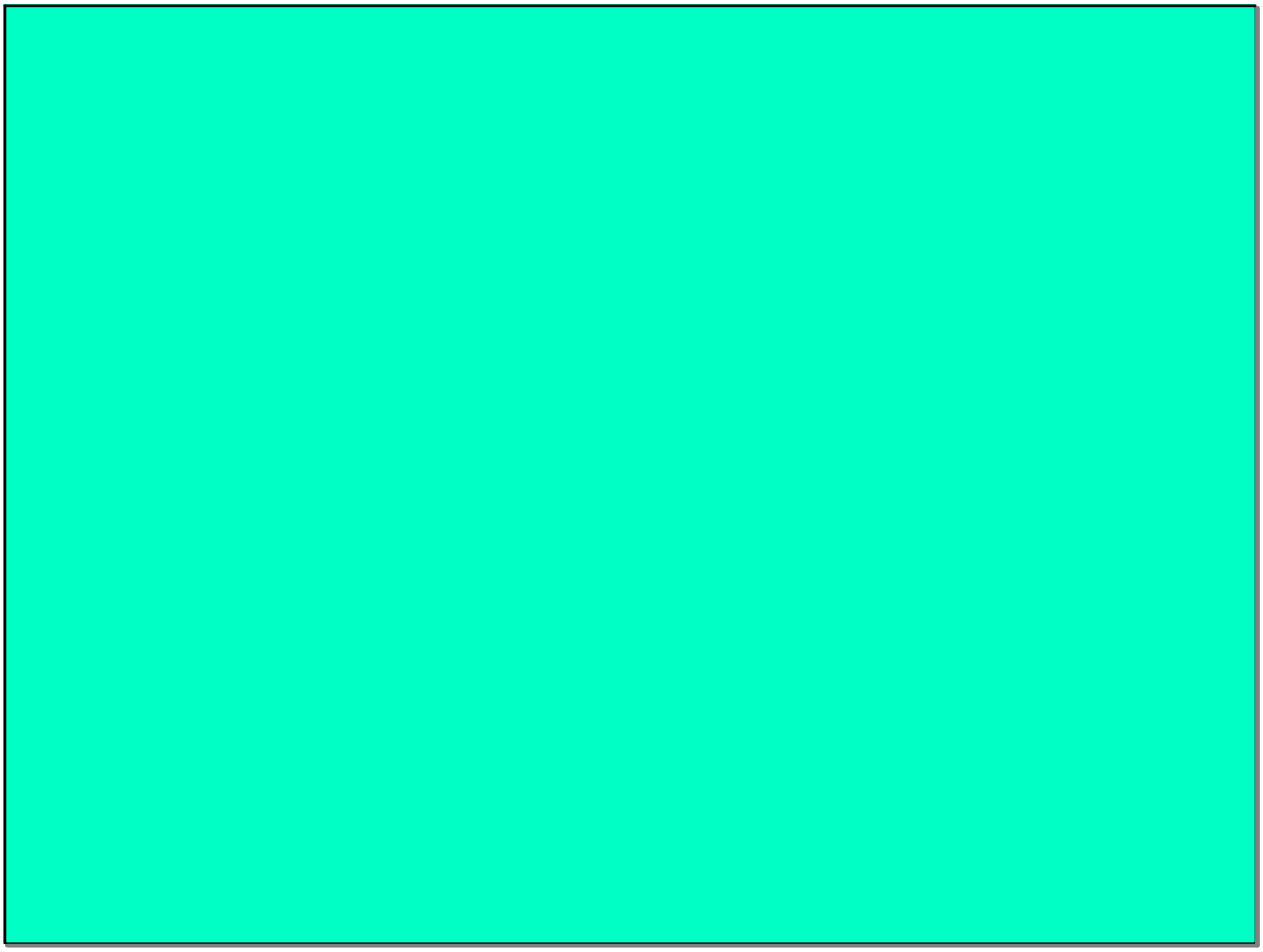


DFS application:
topological sorting

Given a directed acyclic graph (dag) G as input, arrange the vertices in a linear order, say, V[1], ..., V[n] so that if (V[i], V[j]) ∈ G.E then i < j.
vertices are tasks to be performed. An edge indicates a dependency:

means u must be completed before v starts.

Topological sort: G a dag with n vertices.

Initialize an array
V[1..n]
j := n
Run DFS on G,
as each vertex v finishes:
V[j] := v
j := j - 1

V stores vertices in decreasing order by finish time.

Time = $\Theta(|G.V| + |G.E|)$ (worst case; worst-case time of DFS)

Why does this work?
Claim: If there is an edge from u to v ($u \to v$) then u finishes after v finishing.
Why: when DFS is run, at some point DFSfrom(u) is called.
Key: When DFSfrom(u) traverses the edge (u,v), either v has not started yet (then DFSfrom(v) is called and returns) or v has already finished, If v has started but not finished, then there is a cycle involving (u,v). [we assume no cycles]

Minimum Spanning trees

Let G be a weighted (undirected) graph,
n = # nodes (vertices)
m = # edges

A spanning tree of G is a set of n-1 edges that connects the graph. For a spanning tree to exist, G must be connected. We will assume this.

A tree is a connected acyclic graph. A tree on n vertices must have exactly n-1 edges. In fact
$m \geq n$: there must be a cycle
$m < n-1$: G cannot be connected,

A forest is any acyclic graph (connected components are trees).

Generic algorithm
T := ∅ (T is a set of edges)
while |T| < n-1:
look for an edge e ∉ T is safe to add to T
T := T ∪ {e}
[MST = Minimum Spanning Tree]
T is a subset of some MST

e safe means T ∪ {e} is a subset of some MST.

Def: Let T be a subset of a MST. (G.V, T) is a spanning forest. Let C be some connected component of T. Any min weight edge one (but not both) of whose endpoints is in C is safe.

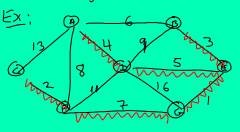
---

2 MST algos:
Kruskal's algo
Prim's algo

Kruskal(G)
// G weighted, connected graph
T := ∅
sort the edges into a list L, sorted by ascending weight

Kruskal(G)
// G weighted, connected graph
T := ∅
Sort the edges into a list L, sorted by ascending weight
for each edge e∈L in order:
if e's endpoints are not in the same component, then
T := T ∪ {e} // e is safe
Do this until |T| = n-1.
// e is safe because processing edges in ascending order by weight.

Ex:



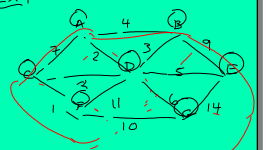
MST (red) has total wt 22

Maintain vertices in a disjoint set system whose sets are the connected components
Make-Set(v) for all vertices v
when e=(u,v) is processed:
if Find(u) ≠ Find(v):
add e to T
Union(u,v)

Time:
Sort the edges $\Theta(m \lg m)$
Union-Find $\Theta(m\,\alpha(n))$
since $\alpha(n) = O(\lg n)$ and $m \geq n-1$ (G is connected)
so $\alpha(m) = O(\lg m)$
Time spent is $\Theta(m \lg m)$ [worst case]
$= \Theta(m \lg n)$
because $n-1 \leq m \leq n^2$
dominated by the sorting.

Prim's Algo
Start at some vertex s (arbitrarily chosen).
T := ∅
C := {s} // the connected component containing s
Do n-1 times:
choose a light (minimum weight) edge e=(u,v)=(v,u) such that u∈C and v∉C
T := T ∪ {e}
C := C ∪ {v}

Ex:



T = {CF, AC, AD, BD, DE, DG}

Prim(s)
T := ∅
C := ∅
s.d := −∞
s.π := NIL
for each v ≠ s
v.d := ∞
v.π := NIL
Insert all vertices into a min-priority queue Q with d-attribute as key
While Q not empty
u := DeleteMin(Q)
C := C ∪ {u}
for each v∈Q such that (u,v)∈G
if w(u,v) < v.d
DecreaseKey(v, w(u,v))
v.π := u
end-while
T := {(v, v.π) : v ≠ s}