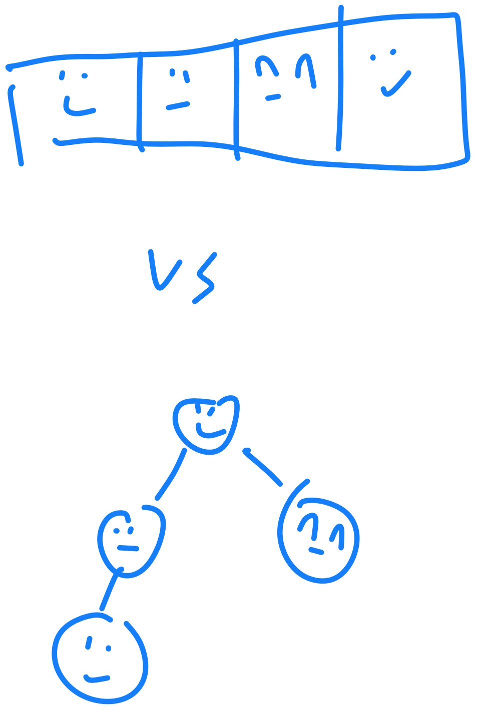
Data Structures
MST 2

CS 225
Brad Solomon & G Carl Evans

November 29, 2023



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Announcements

Project teams be sure to schedule your mid-project checkin soon!

Today lecture is good example for pseudo-code runtime

This week's lab is extra credit lab

ICES Evaluations are open! If enough students submit, extra credit!

Learning Objectives

Review the minimum spanning tree (with weights)

Discuss Kruskal’s MST Algorithm

Discuss Prim’s MST Algorithm

Minimum Spanning Tree Algorithms

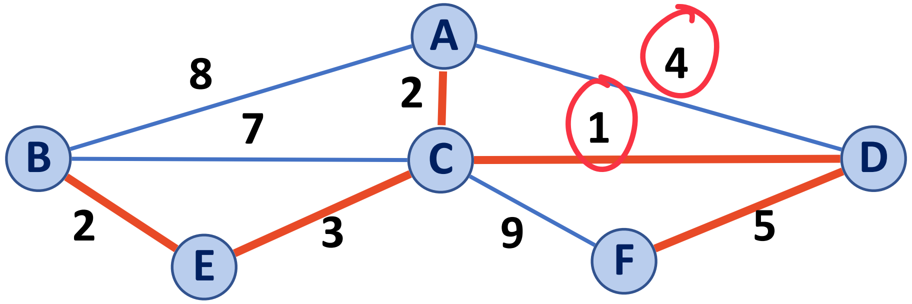
Input: Connected, undirected graph **G** with edge weights (unconstrained, but must be additive)

positive

DFS/BFS otherwise

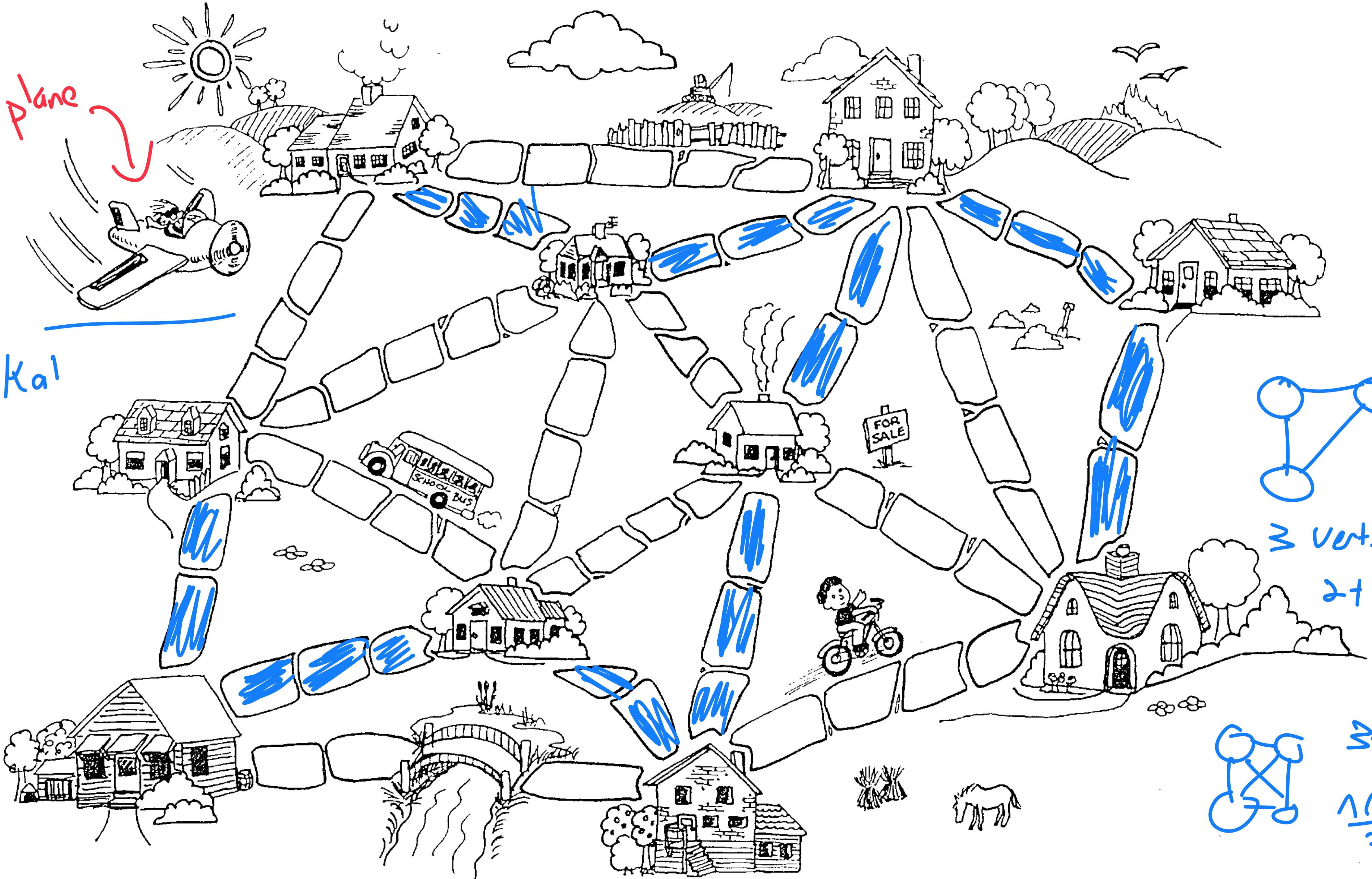
Output: A graph G' with the following properties:

- G' is a spanning graph of G – every node connected
- G' is a tree (connected, acyclic)
- G' has a minimal total weight among all spanning trees



Has airplane

Kruskal



3 vert.

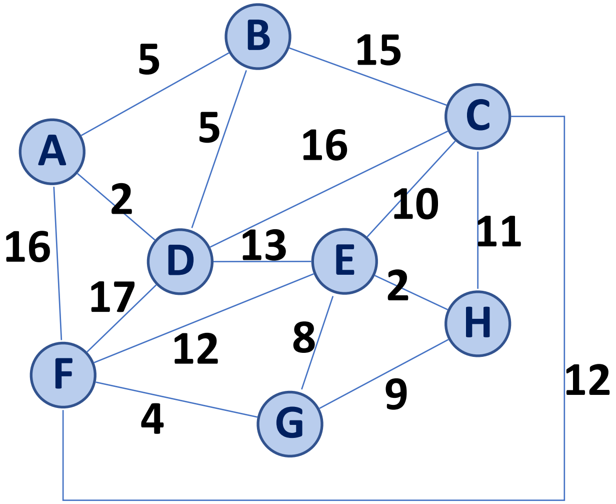
2+1

3+2+1

$\frac{n(n-1)}{2}$

Kruskal's Algorithm

What graph info do we need?



1) Min weights on edges

↳ Some data structure on edges

An array? Edge list (unsorted list of edges)

↳ A sorted array

Priority queue. → A Heap → We want the min edge

We want min edge quickly

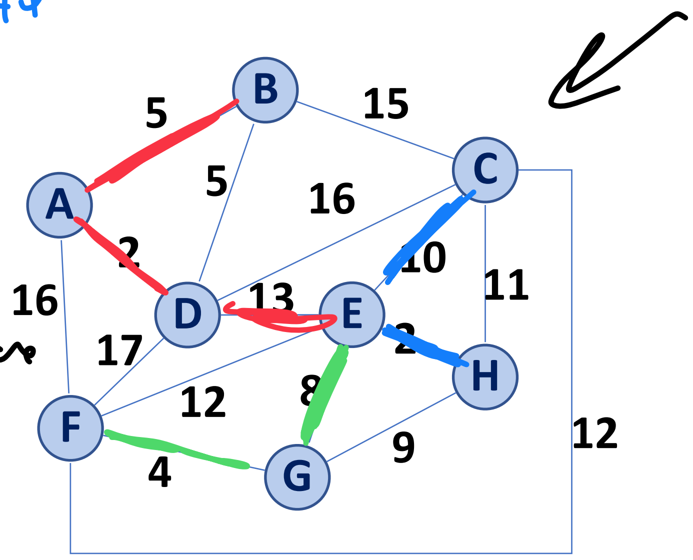
2) Use edges that connect unconnected vertices

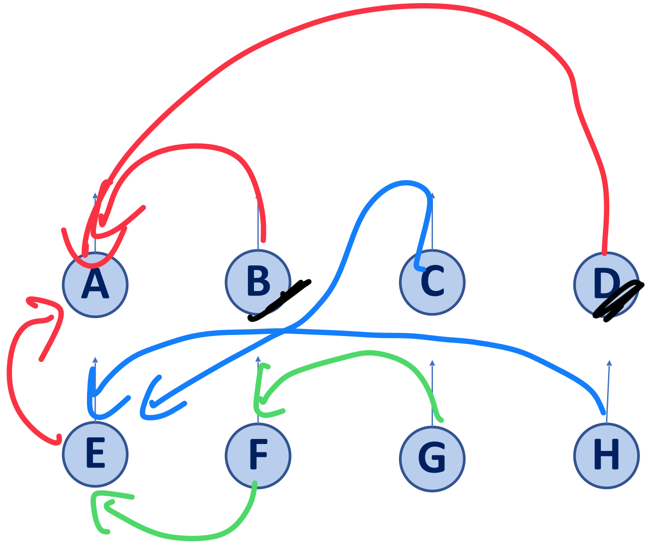
↳ Disjoint Set

Kruskal's Algorithm

i++

~~(A, D)~~
~~(E, H)~~
~~(F, G)~~
~~(A, B)~~
~~(B, D)~~ — ignore
~~(G, E)~~
~~(G, H)~~
~~(E, C)~~
~~(C, H)~~
~~(E, F)~~
~~(F, C)~~
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)





1) Build a **priority queue** on edges

↳ either min Heap or sorted array or ???

2) Build a **disjoint set** on vertices

3) Get repeated min edge
if connects two sets, union sets & record edge

When to stop?

1) size disjoint set is size |# of vertices|

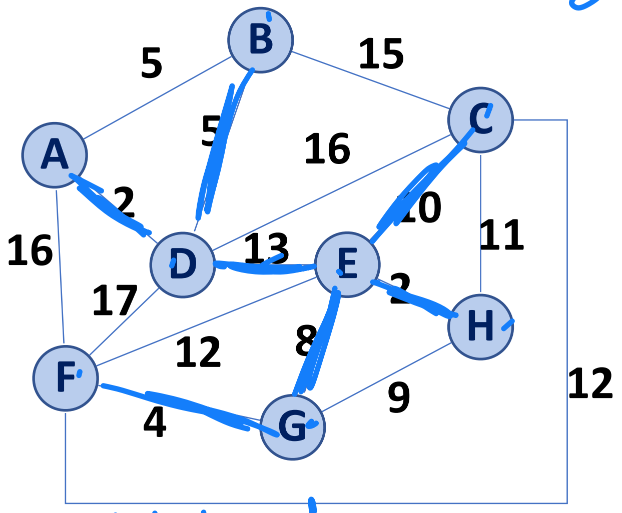
2) Added n-1 edges to our "record"

Kruskal's Algorithm

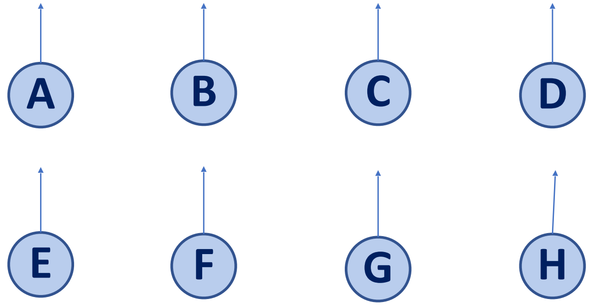
Runtime: O(n) + ___ + ___ + O(m)
build   remove min

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)

8 vertices



7 total edges




```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G.vertices()):
    forest.makeSet(v)

  PriorityQueue Q     // min edge weight
  Q.buildFromGraph(G.edges())

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) != forest.find(v):
      T.addEdge(u, v)
      forest.union( forest.find(u),
                    forest.find(v) )

  return T
```

make D.S. O(n)

???

MST

Stop when MST has size n-1

???

O(1)

O(m) not O(n)

loop runs at most m

Kruskal's Algorithm

Simple Connected graph

$O(n) \leq O(m) \leq O(n^2)$ → $O(\log n)$ ≤≤ $O(\log m)$

Priority Queue:	Heap	Sorted Array
Building :7	$O(m)$	$O(m \log m)$
Each removeMin :12	$O(\log m)$	$O(1)$

Heap: $O(m) + O(m \log m)$

Array: $O(m \log m) + O(m)$

$m \leq C * n^2$

$\log(m) \leq C * \log(n^2)$

$2C * \log(n)$

```
1  KruskalMST(G):
2    DisjointSets forest
3    foreach (Vertex v : G.vertices()):
4      forest.makeSet(v)
5
6    PriorityQueue Q    // min edge weight
7    Q.buildFromGraph(G.edges())   ← once
8
9    Graph T = (V, {})
10
11   while |T.edges()| < n-1:      ← m times
12     Vertex (u, v) = Q.removeMin()
13     if forest.find(u) != forest.find(v):
14        T.addEdge(u, v)
15        forest.union( forest.find(u),
16                      forest.find(v) )
17
18   return T
19
```

$n-1 \leq m \Rightarrow O(n) \leq O(m)$

Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	$O(n + m + m \log n)$
Sorted Array	$O(n + m + m \log n)$

```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G.vertices()):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  Q.buildFromGraph(G.edges())

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) != forest.find(v):
       T.addEdge(u, v)
       forest.union( forest.find(u),
                     forest.find(v) )

  return T
```

Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	O(n + m + m log(n))
Sorted Array	O(n + m log(n) + m)

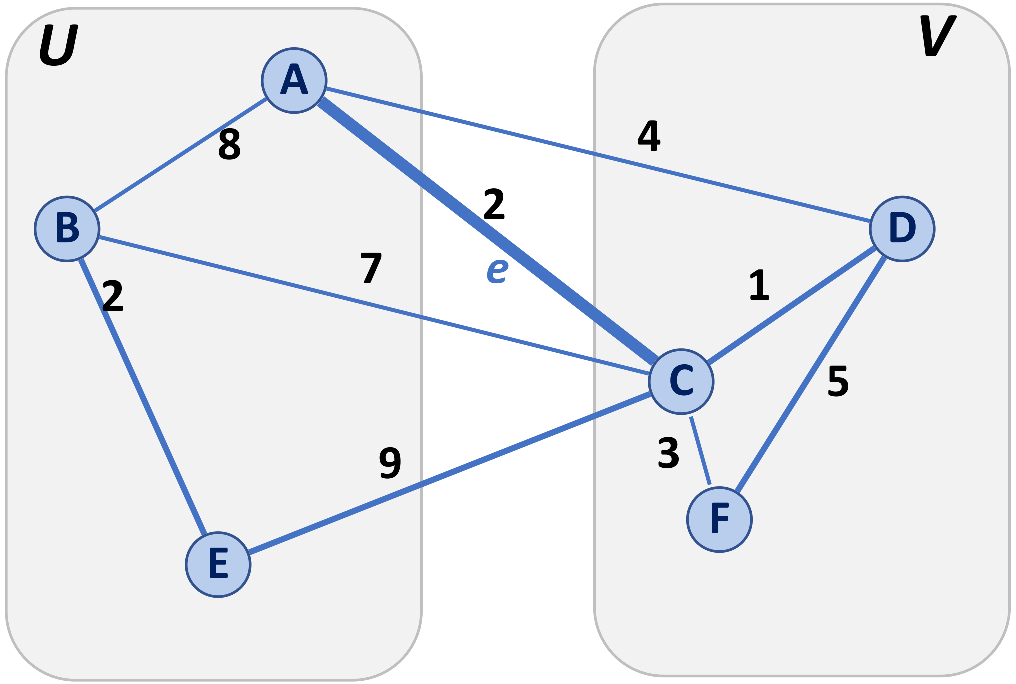
```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  Q.buildFromGraph(G)

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) != forest.find(v):
       T.addEdge(u, v)
       forest.union( forest.find(u),
                     forest.find(v) )

  return T
```

Partition Property

Consider an arbitrary partition of the vertices on **G** into two subsets **U** and **V**.

Let **e** be an edge of minimum weight across the partition.

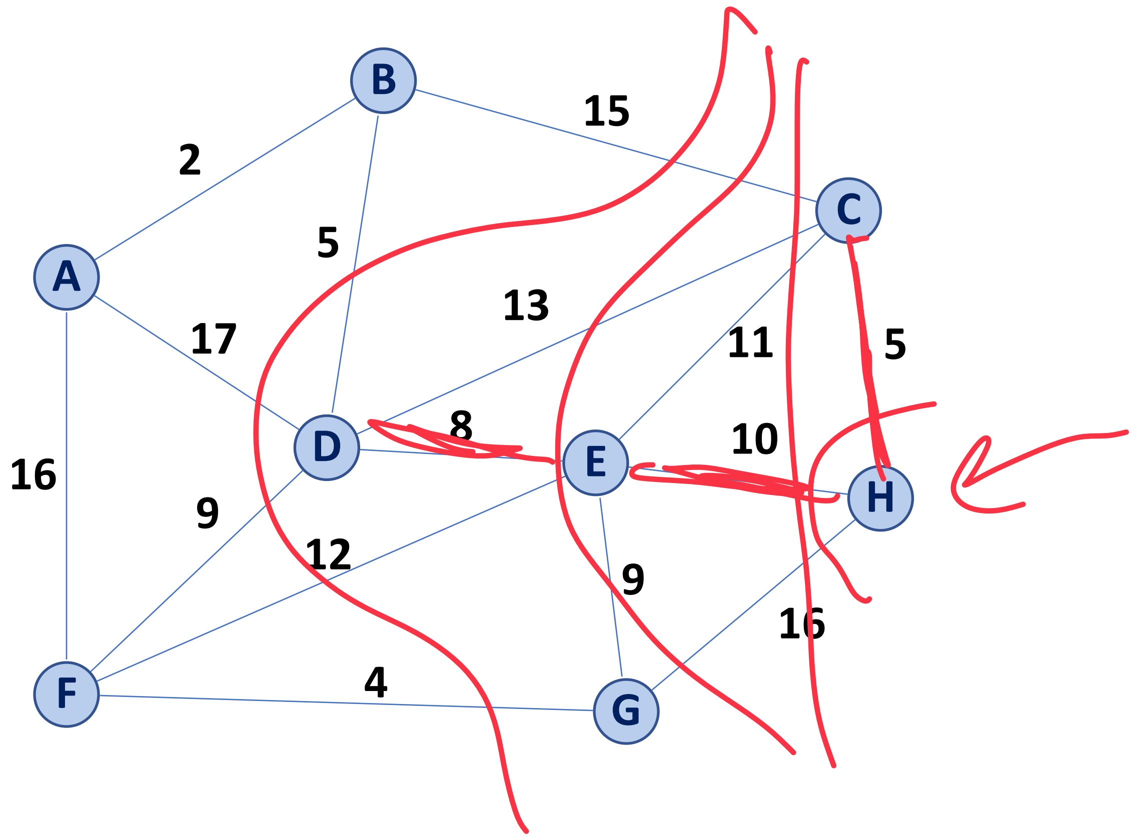
Then **e** is part of some minimum spanning tree.



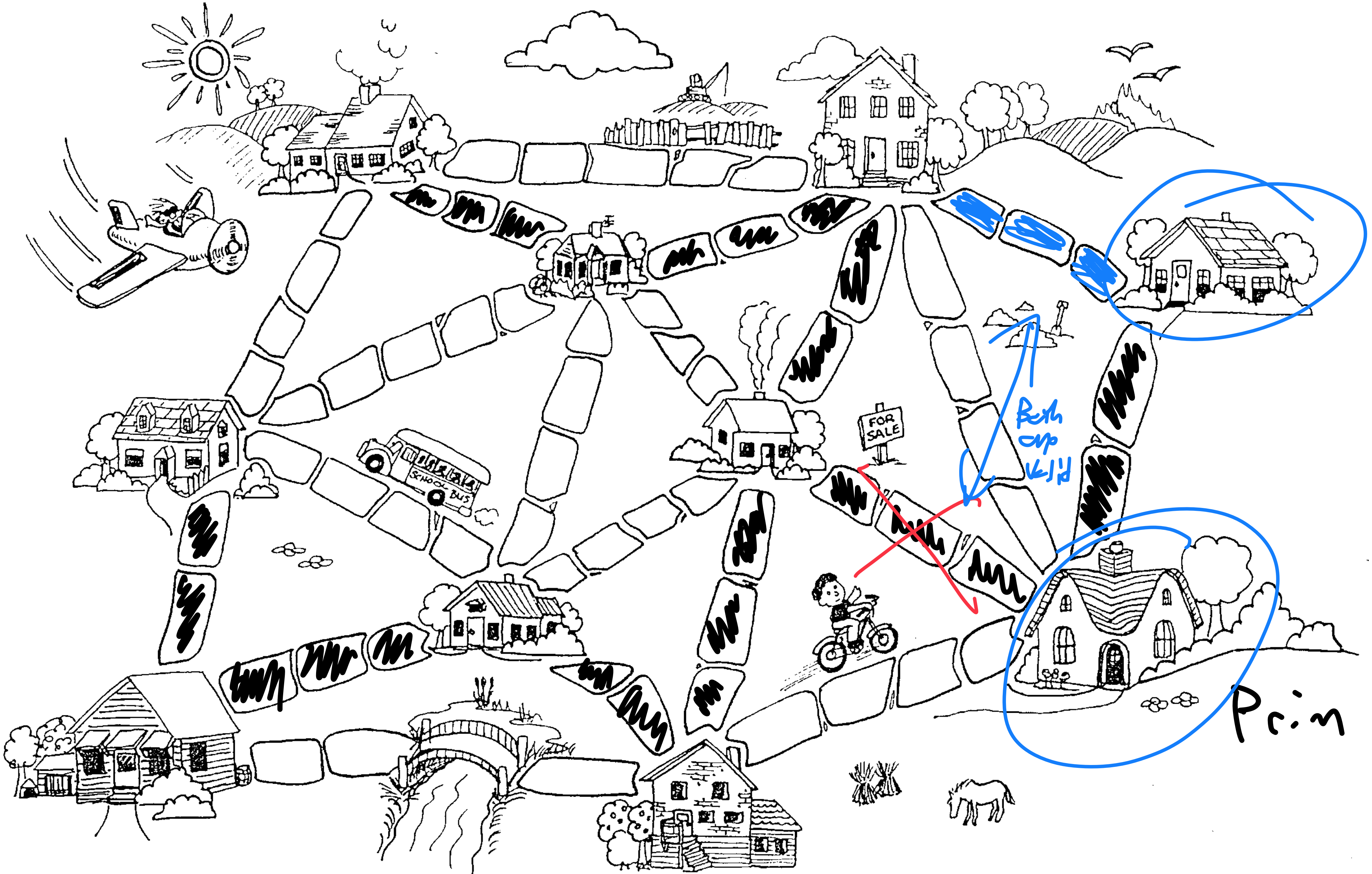
Partition Property

The partition property suggests an algorithm:

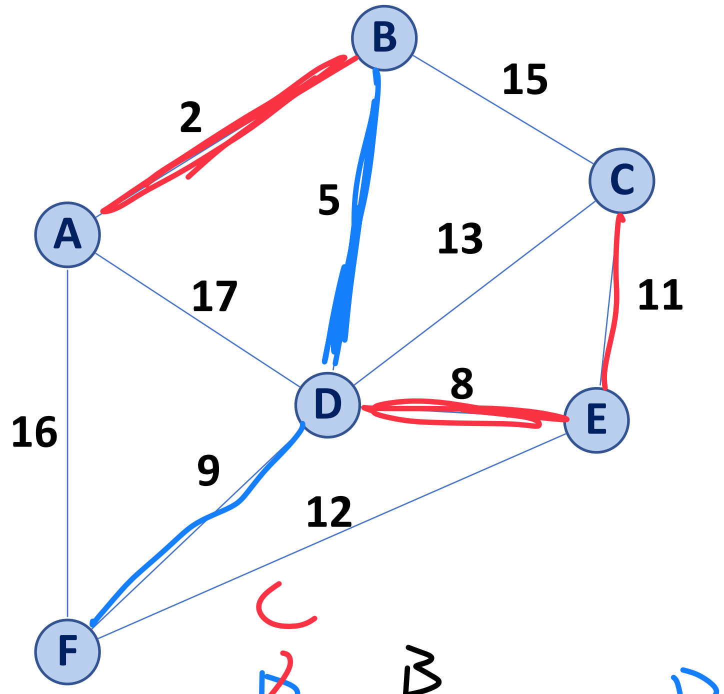
Out



In


FOR SALE
SCHOOL BUS
Both are valid
Prim

Prim's Algorithm



	A	B	C	D	E	F
p		A	~~B~~ ~~D~~ E	~~A~~ B	D	~~A~~ D
d	0	~~∞~~ 2	~~∞~~ ~~15~~ ~~13~~ 11	~~∞~~ ~~17~~ 5	~~∞~~ 8	~~∞~~ ~~16~~ 9

```
PrimMST(G, s):
  Input: G, Graph;
         s, vertex in G, starting vertex
  Output: T, a minimum spanning tree (MST) of G

  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q   // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T           // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m

  return T
```

Init step

cost to any member of in-group

choice of priority queue

minheap vs unsorted array???

choice of graph structure ???

```
PrimMST(G, s):
  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T           // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m
```

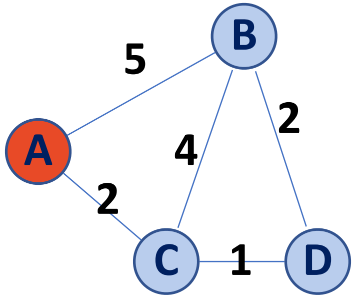
O(n)

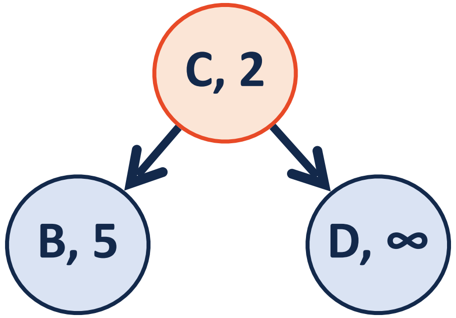
O(1)

???

O(n)

A	B	C	D
0	5	2	∞





On friday rebuilding heap vs unsorted array

```
PrimMST(G, s):
  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m

```

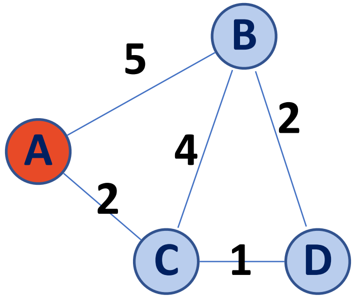
for one vertex is $O(n)$

$O(\deg(v))$

$\sum_v \deg(v) = 2|E| = 2m$

	Adj. Matrix	Adj. List
Heap	$O(n^2)$ + ???	$O(m)$ + ???

(A, 0)
(D, ∞)
(C, 2)
(B, 5)




```
PrimMST(G, s):
  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m
```

	Adj. Matrix	Adj. List
Heap	$O(n^2 + m \lg(n))$	$O(n \lg(n) + m \lg(n))$
Unsorted Array		

Prim’s Algorithm

Sparse Graph:

Dense Graph:

```
PrimMST(G, s):
  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m
```

	Adj. Matrix	Adj. List
Heap	$O(n^2 + m\lg(n))$	$O(n\lg(n) + m\lg(n))$
Unsorted Array	$O(n^2)$	$O(n^2)$

MST Algorithm Runtime:

Kruskal's Algorithm:

$O(n + m \log(n))$

Prim's Algorithm:

$O(n \log(n) + m \log(n))$

Sparse Graph:

Dense Graph:

Suppose I have a new heap:

	Binary Heap	Fibonacci Heap
Remove Min	O(lg(n))	O(lg(n))
Decrease Key	O(lg(n))	O(1)*

What's the updated running time?

```
PrimMST(G, s):
  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m
```