CSE 332

AUGUST 14TH – EFFICIENT REDUCTIONS

ADMINISTRIVIA

- P2 should be back and graded

ADMINISTRIVIA

- P2 should be back and graded
- P1 ClassNotFound fixed tonight

ADMINISTRIVIA

- P2 should be back and graded
- P1 ClassNotFound fixed tonight
- EX11 graded and back tonight

ADMINISTRIVIA

- **P2 should be back and graded**
- **P1 ClassNotFound fixed tonight**
- **EX11 graded and back tonight**
- **Parallelism exercises**
  - If you passed the tests on your home computer, you will get full marks (provided you actually used parallelism)

ADMINISTRIVIA

- **P2 should be back and graded**
- **P1 ClassNotFound fixed tonight**
- **EX11 graded and back tonight**
- **Parallelism exercises**
  - If you passed the tests on your home computer, you will get full marks (provided you actually used parallelism)
- **P3 due tonight at midnight**

EXERCISE TOKENS

- Redoing exercises with tokens

EXERCISE TOKENS

- **Redoing exercises with tokens**
  - Fill out token form

EXERCISE TOKENS

- **Redoing exercises with tokens**
  - Fill out token form
  - This should unlock the assignment so that you can resubmit

EXERCISE TOKENS

- **Redoing exercises with tokens**
  - Fill out token form
  - This should unlock the assignment so that you can resubmit
  - Additionally, for a brief paragraph, explain the mistakes that you made and how you learned from them (even if you just didn't submit at all)

COURSE EVALUATIONS

- **Course evaluations are out, please take 5 or 10 minutes to fill out the evaluations**
  - https://uw.iasystem.org/survey/179903

COURSE EVALUATIONS

- **Course evaluations are out, please take 5 or 10 minutes to fill out the evaluations**
  - https://uw.iasystem.org/survey/179903
- **These are very important, not just for me but for the department**

COURSE EVALUATIONS

- **Course evaluations are out, please take 5 or 10 minutes to fill out the evaluations**
  - https://uw.iasystem.org/survey/179903
- **These are very important, not just for me but for the department**
- **Summer quarter: what went well and what was difficult**

COURSE EVALUATIONS

- **Course evaluations are out, please take 5 or 10 minutes to fill out the evaluations**
  - https://uw.iasystem.org/survey/179903
- **These are very important, not just for me but for the department**
- **Summer quarter: what went well and what was difficult**
  - Prereq course, want to balance preparing you and not overworking you

EXAM REVIEW

- **Tuesday 1:30 PM 045**

EXAM REVIEW

- **Tuesday 1:30 PM 045**
  - Topics list will go out soon

EXAM REVIEW

- **Tuesday 1:30 PM 045**
  - Topics list will go out soon
- **Exam will be a two-period exam**

EXAM REVIEW

- **Tuesday 1:30 PM 045**
  - Topics list will go out soon
- **Exam will be a two-period exam**
  - Section: Sorting and Parallelism

EXAM REVIEW

- **Tuesday 1:30 PM 045**
  - Topics list will go out soon
- **Exam will be a two-period exam**
  - Section: Sorting and Parallelism
  - Friday: Graphs and Remaining

EXAM REVIEW

- **Tuesday 1:30 PM 045**
  - Topics list will go out soon
- **Exam will be a two-period exam**
  - Section: Sorting and Parallelism
  - Friday: Graphs and Remaining
  - Material from before the midterm is fair game for both days

TODAY'S LECTURE

- Graph algorithm review

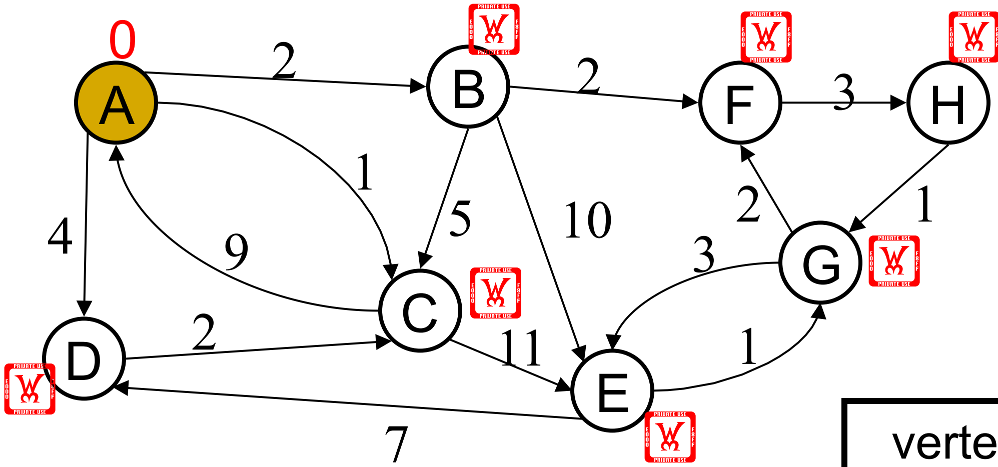
TODAY'S LECTURE

- Graph algorithm review
- Efficient reductions

DIJKSTRAS ALGORITHM

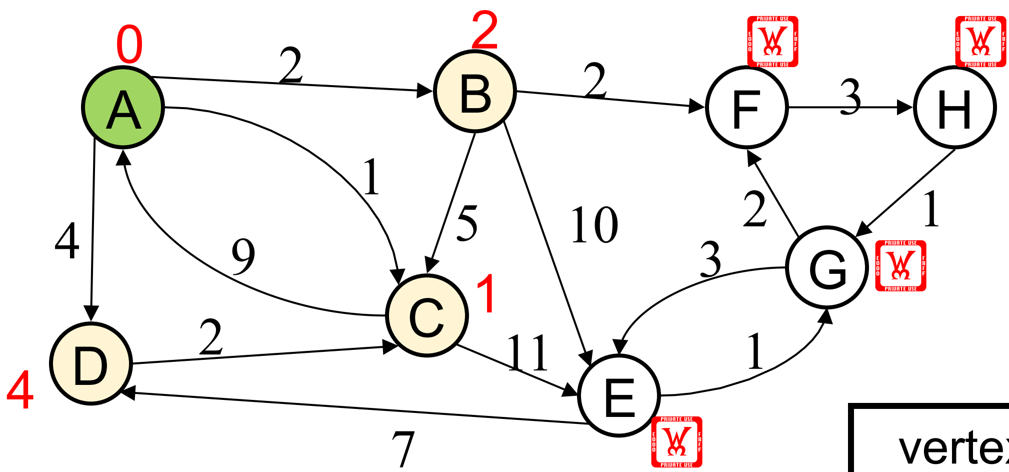
1. **For each node `v`, set `v.cost = ∞` and `v.known = false`**
2. **Set `source.cost = 0`**
3. **While there are unknown nodes in the graph**
   a) Select the unknown node **`v`** with lowest cost
   b) Mark **`v`** as known
   c) For each edge **`(v,u)`** with weight **`w`**,

```
c1 = v.cost + w // cost of best path through v  to u
c2 = u.cost   // cost of best path to u previously known
if(c1 < c2){ // if the path through v is better
  u.cost = c1
  u.path = v // for computing actual paths
}
```



vertex	known?	cost	path
A		0	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	
H		??	

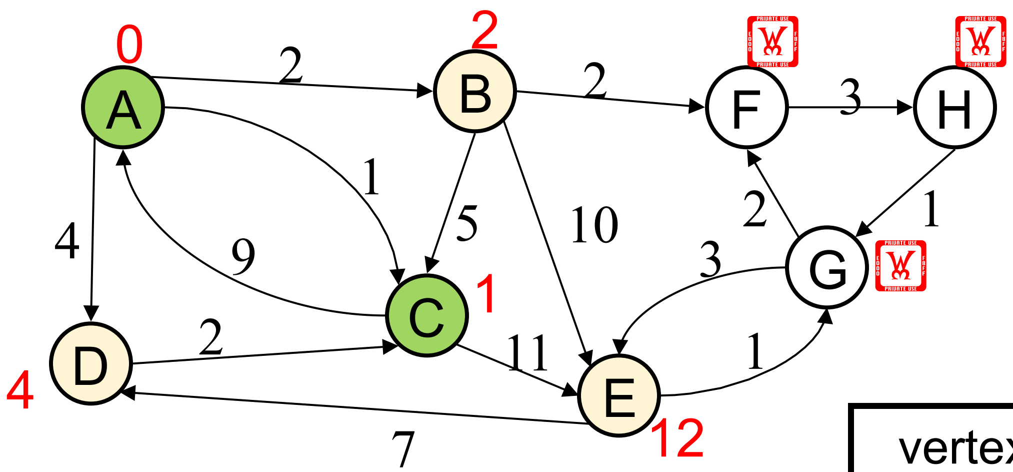
<u>Order Added to Known Set:</u>



vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		??	
F		??	
G		??	
H		??	

<u>Order Added to Known Set:</u>

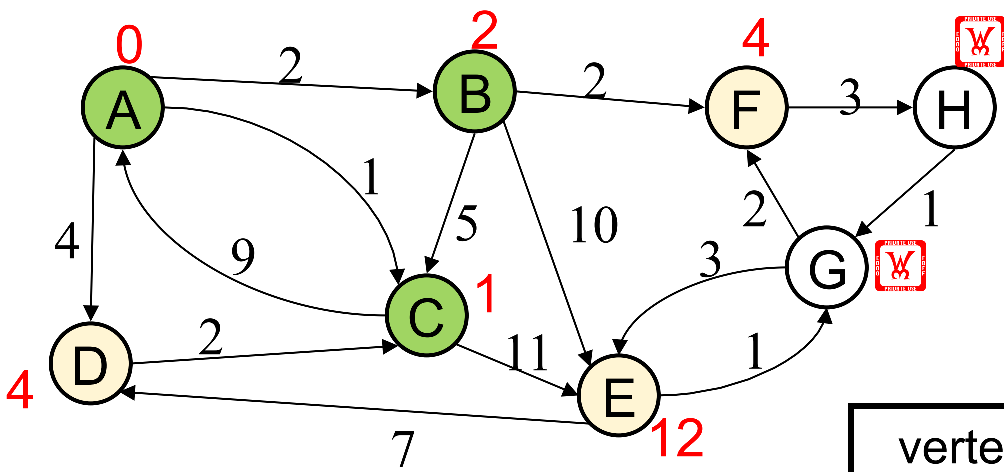
A



vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		??	
G		??	
H		??	

<u>Order Added to Known Set:</u>

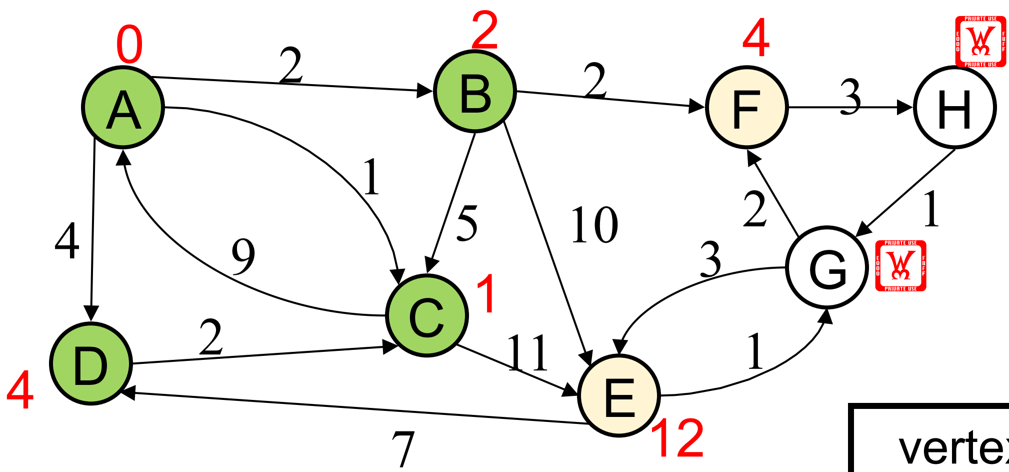
A, C



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		??	
H		??	

<u>Order Added to Known Set:</u>

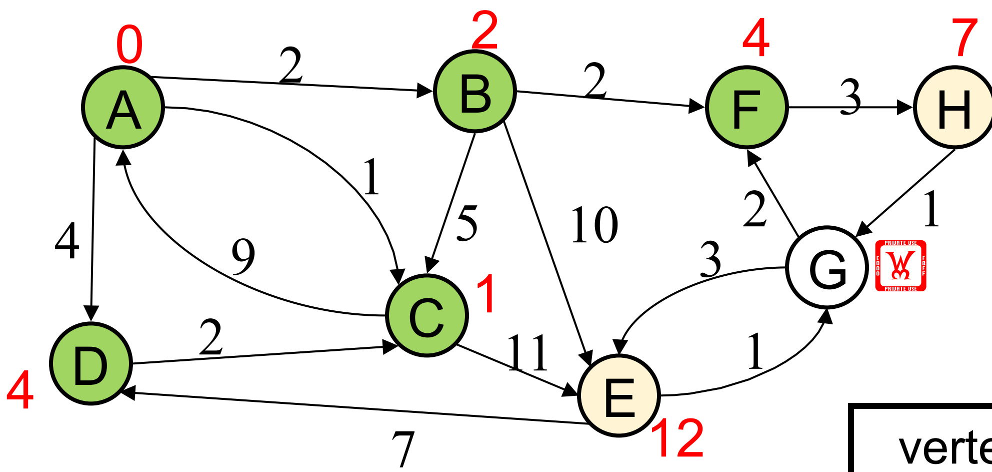
A, C, B



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		??	
H		??	

<u>Order Added to Known Set:</u>

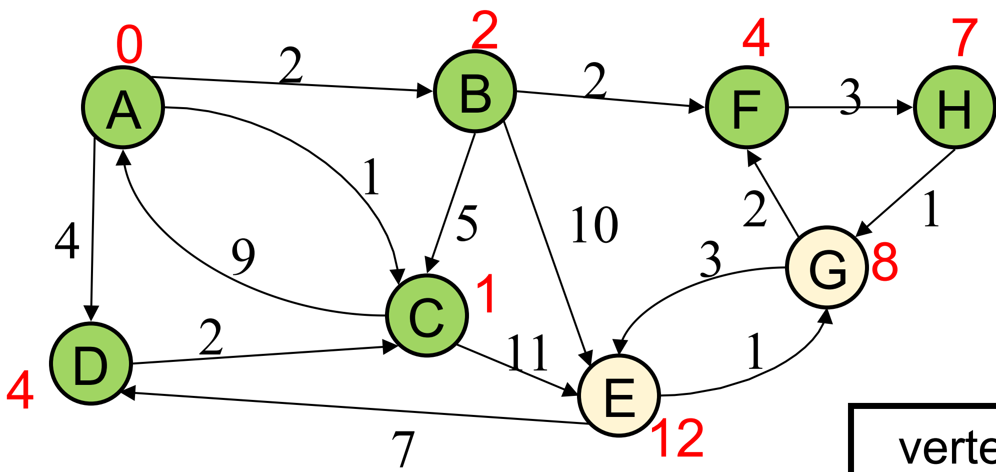
A, C, B, D



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		??	
H		≤ 7	F

Order Added to Known Set:

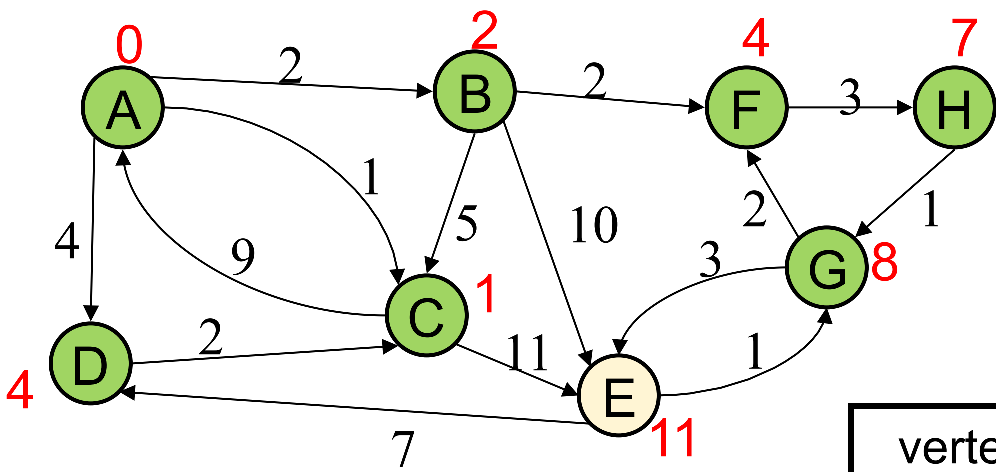
A, C, B, D, F



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

<u>Order Added to Known Set:</u>

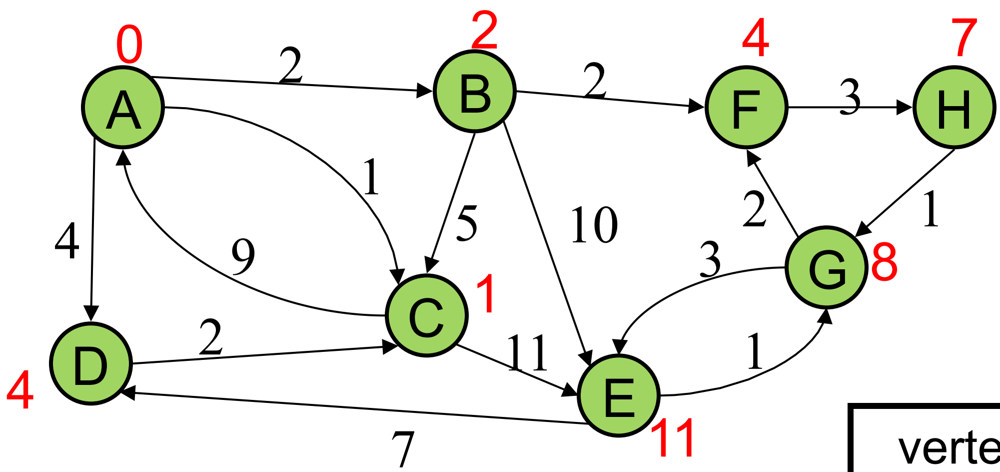
A, C, B, D, F, H



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

<u>Order Added to Known Set:</u>

A, C, B, D, F, H, G



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

<u>Order Added to Known Set:</u>

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

PRIM'S ALGORITHM

- A traversal

PRIM'S ALGORITHM

- **A traversal**
  - Pick a start node

PRIM'S ALGORITHM

- **A traversal**
  - Pick a start node
  - Keep track of all of the vertices you can reach

PRIM'S ALGORITHM

- **A traversal**
  - Pick a start node
  - Keep track of all of the vertices you can reach
  - Add the vertex that is closest (has the edge with smallest weight) to the current spanning tree.

PRIM'S ALGORITHM

- **A traversal**
  - Pick a start node
  - Keep track of all of the vertices you can reach
  - Add the vertex that is closest (has the edge with smallest weight) to the current spanning tree.
- **Is this similar to something we've seen before?**

PRIM'S ALGORITHM

- Modify Dijkstra's algorithm

PRIM’S ALGORITHM

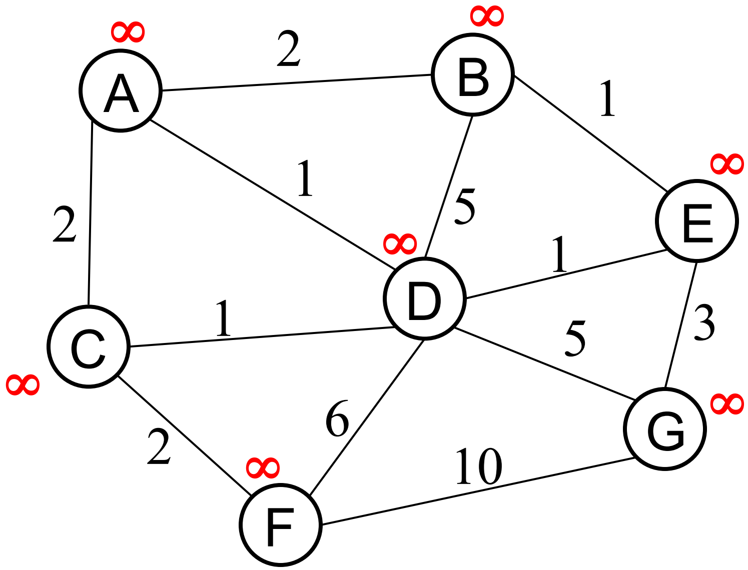
- **Modify Dijkstra’s algorithm**
  - Instead of measuring the total length from start to the new vertex, now we only care about the edge from our current spanning tree to new nodes

THE ALGORITHM

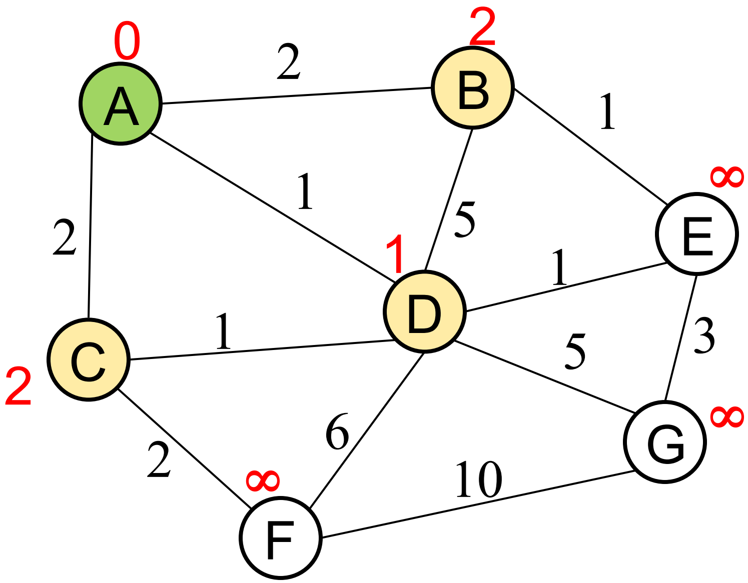
1. **For each node v, set v.cost = ∞ and v.known = false**
2. **Choose any node v**
   a) Mark **v** as known
   b) For each edge **(v,u)** with weight **w**, set **u.cost=w** and **u.prev=v**
3. **While there are unknown nodes in the graph**
   a) Select the unknown node **v** with lowest cost
   b) Mark **v** as known and add **(v, v.prev)** to output
   c) For each edge **(v,u)** with weight **w**,

```
if(w < u.cost) {
    u.cost = w;
  u.prev = v;
}
```

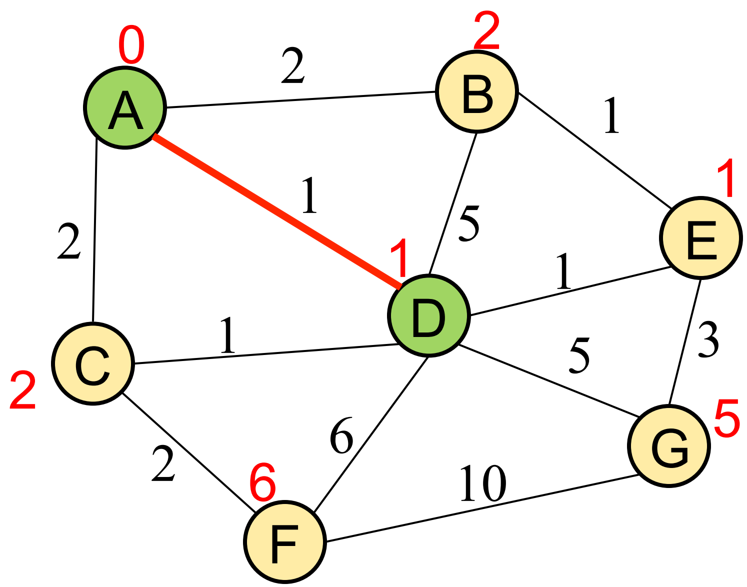
EXAMPLE



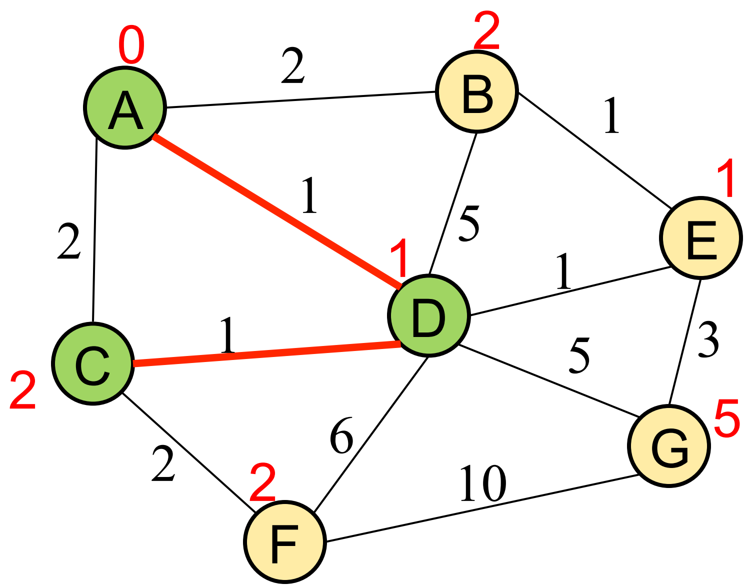
vertex	known?	cost	prev
A		∞	
B		∞	
C		∞	
D		∞	
E		∞	
F		∞	
G		∞	



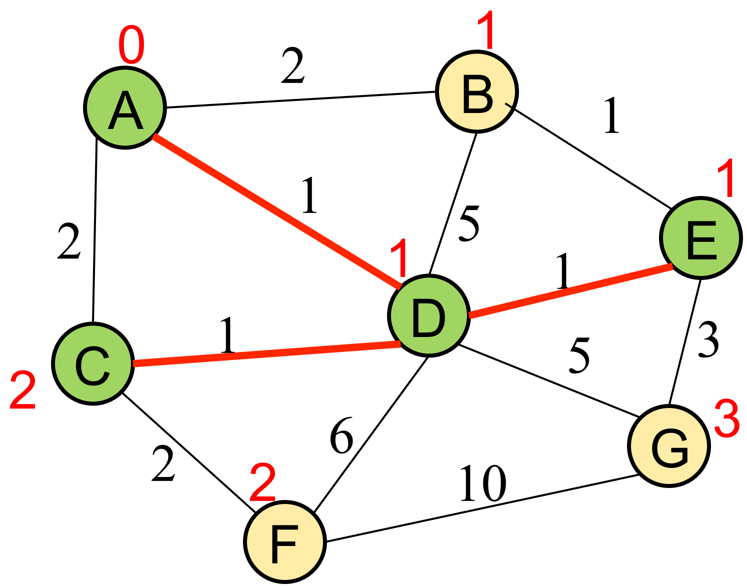
vertex	known?	cost	prev
A	Y	0	
B		2	A
C		2	A
D		1	A
E		∞	
F		∞	
G		∞	



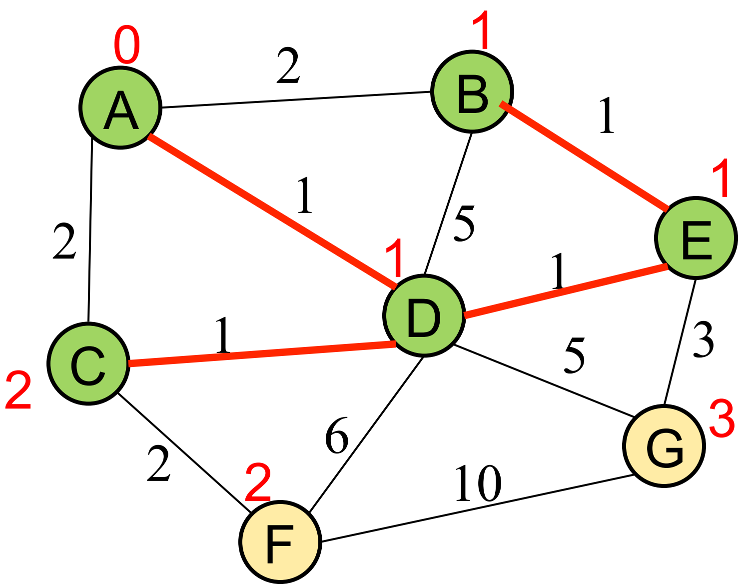
vertex	known?	cost	prev
A	Y	0	
B		2	A
C		1	D
D	Y	1	A
E		1	D
F		6	D
G		5	D



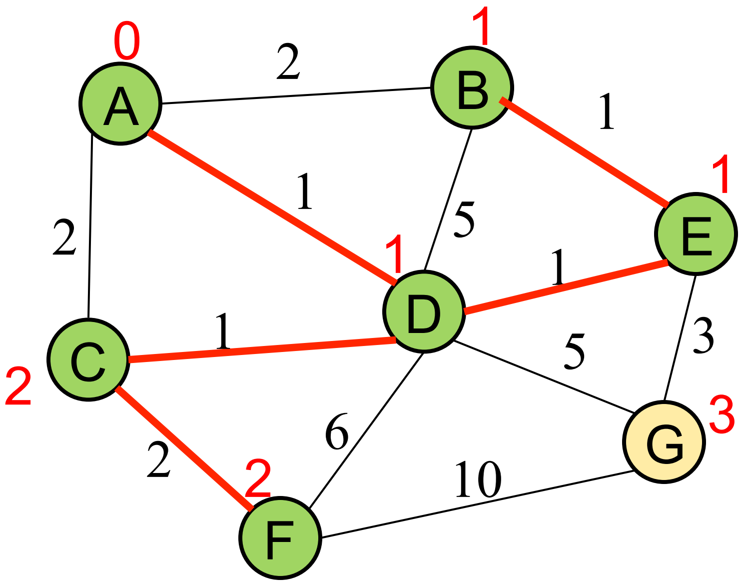
vertex	known?	cost	prev
A	Y	0	
B		2	A
C	Y	1	D
D	Y	1	A
E		1	D
F		2	C
G		5	D



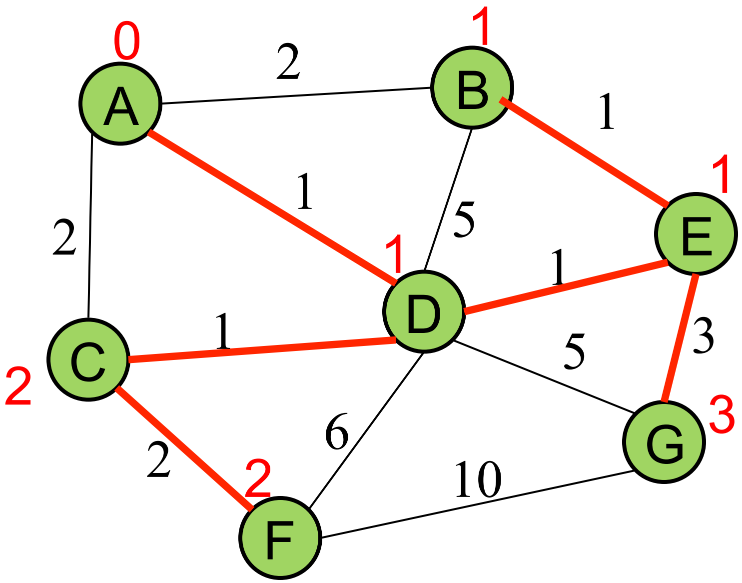
vertex	known?	cost	prev
A	Y	0	
B		1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E



vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E



vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F	Y	2	C
G		3	E



vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F	Y	2	C
G	Y	3	E

KRUSKAL'S ALGORITHM

- **Pseudocode:**

KRUSKAL'S ALGORITHM

- **Pseudocode:**
  - Sort the edges (or place them into a heap)
  - Create a union-find data structure with all separate vertices

KRUSKAL'S ALGORITHM

- **Pseudocode:**
  - Sort the edges (or place them into a heap)
  - Create a union-find data structure with all separate vertices
  - For each edge, add it to the minimum spanning tree if it does not form a cycle

KRUSKAL'S ALGORITHM

- **Pseudocode:**
  - Sort the edges (or place them into a heap)
  - Create a union-find data structure with all separate vertices
  - For each edge, add it to the minimum spanning tree if the two vertices don't have the same representative in the union find

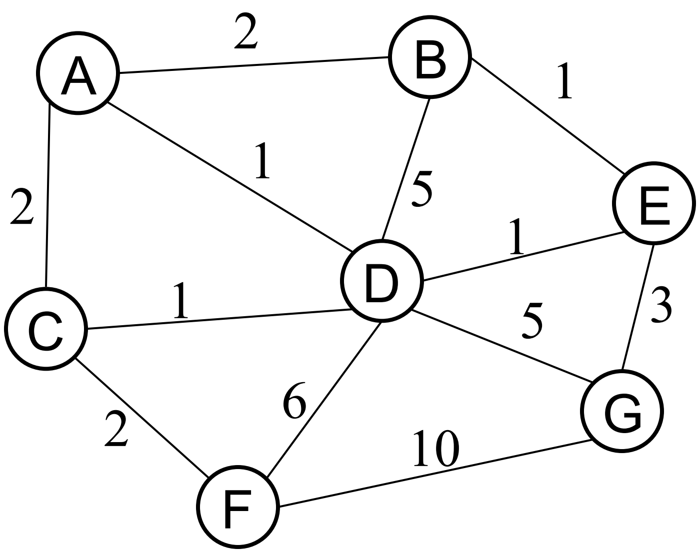
KRUSKAL'S ALGORITHM

- **Pseudocode:**
  - Sort the edges (or place them into a heap)
  - Create a union-find data structure with all separate vertices
  - For each edge, add it to the minimum spanning tree if the two vertices don't have the same representative in the union find
  - Union the two vertices in the union find

KRUSKAL'S ALGORITHM

- **Pseudocode:**
  - Sort the edges (or place them into a heap)
  - Create a union-find data structure with all separate vertices
  - For each edge, add it to the minimum spanning tree if the two vertices don't have the same representative in the union find
  - Union the two vertices in the union find
  - Stop after you've added |V|-1 edges

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

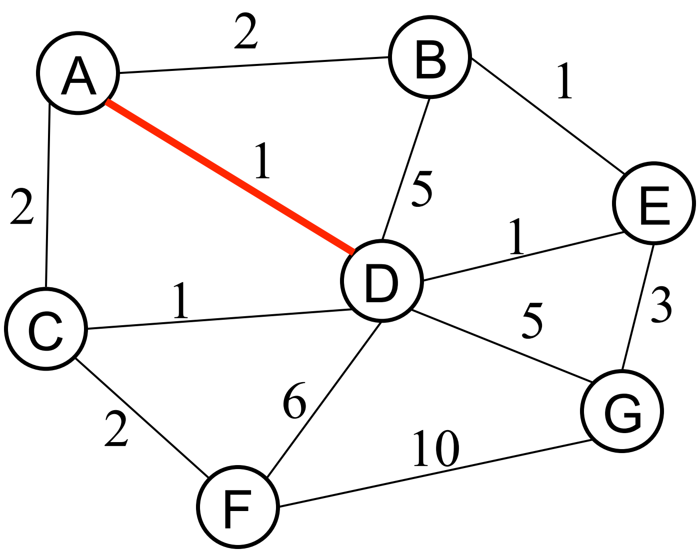
6: (D,F)

10: (F,G)

Output:

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

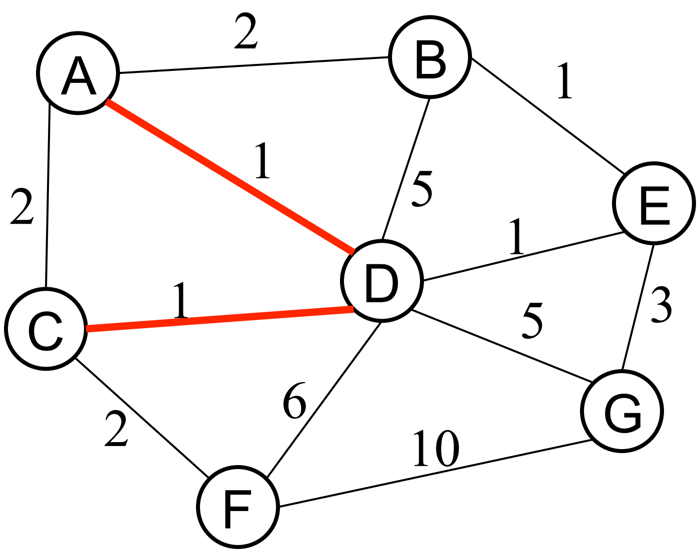
6: (D,F)

10: (F,G)

Output: (A,D)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

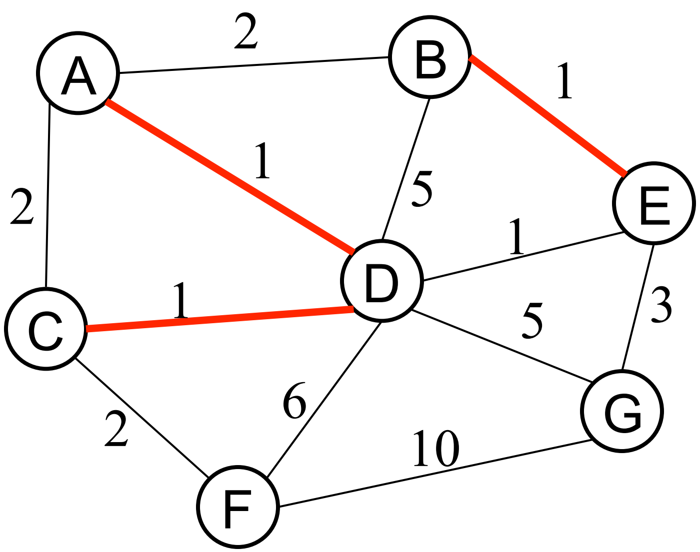
6: (D,F)

10: (F,G)

Output: (A,D), (C,D)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

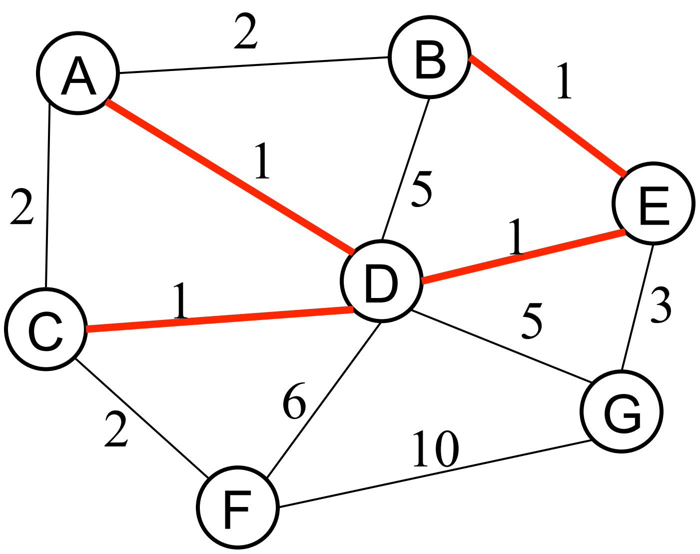
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

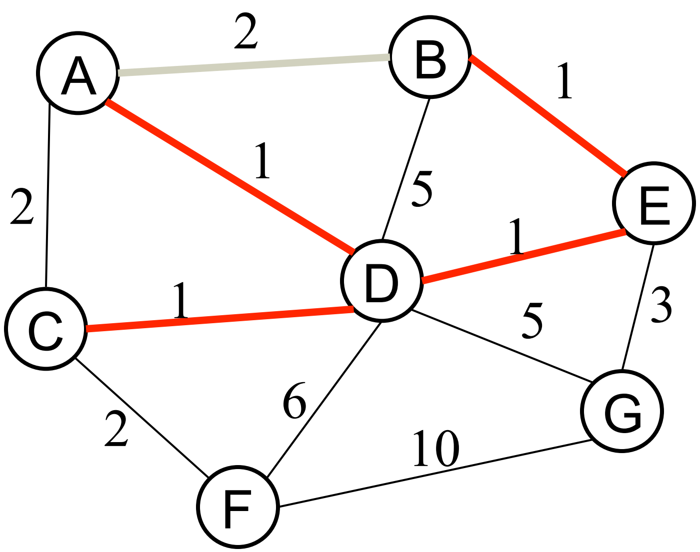
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

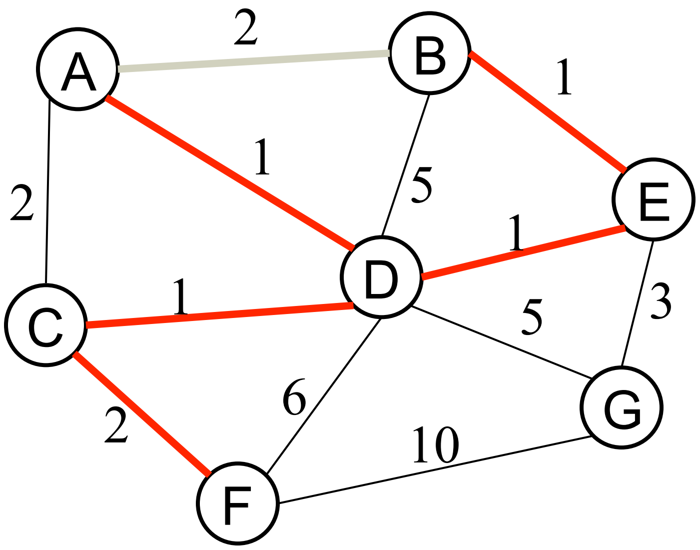
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

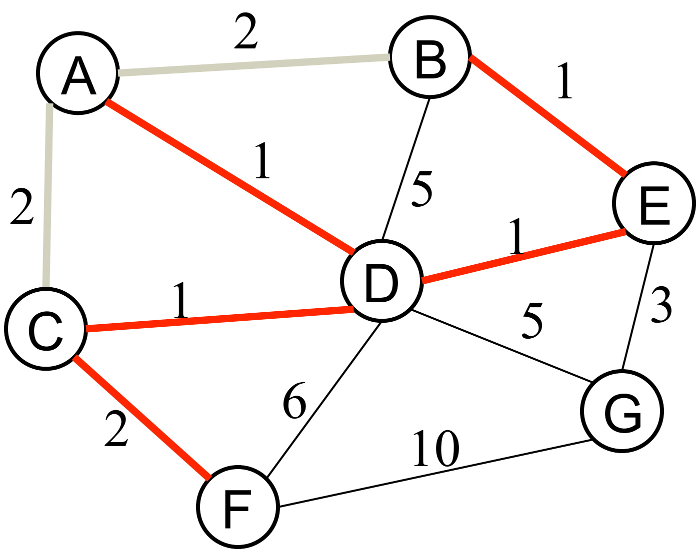
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

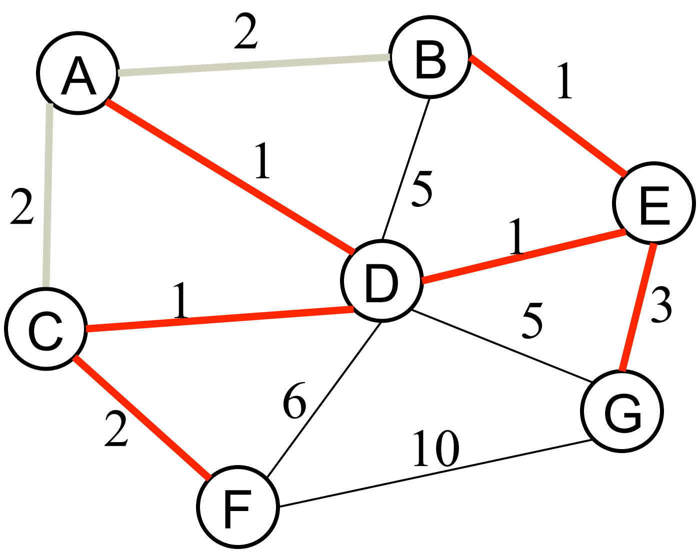
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

EXAMPLE



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F), (E,G)

Note: At each step, the union/find sets are the trees in the forest

EFFICIENT REDUCTIONS

- https://courses.cs.washington.edu/courses/cse332/17wi/lectures/p-np-1/efficient-reductions.pdf
- https://courses.cs.washington.edu/courses/cse332/17wi/lectures/p-np-2/p-np.pdf

NEXT CLASS

- **Randomization and Approximation**

NEXT CLASS

- Randomization and Approximation
- Exam review