CSCI 241

Lecture 25
Greedy Algorithms, MSTs, Prim's Algorithm

Happenings

- Tuesday, 12/4 – AWC Bake Sale – 9 am – 2 pm in the CF 1st Floor Lobby
- Tuesday, 12/4 – CS Study Break! – 3 pm in the CF 4th Floor Lobby
- Wednesday, 12/5 – Cybersecurity Career Taxonomy with Jeff Costlow, Deputy CISO of ExtraHop – 5 pm in CF 125
- Thursday, 12/6 – CS Internship Poster Session – 3 – 5 pm in the CF 1st Floor Hallways
- Then on Saturday, 12/10, Aran Clauson invites you to Buffalo Wild Wings for a fundraiser to support our local CAP squadron!
- 10% of the proceeds from your meal will be donated if you show your server this coupon!

Announcements

- Grades: Canvas averages do not reflect your current grade in absolute terms. As a *very* rough guideline, add 10% to your current average.
  - A4 (10%) and the final exam (20%) are still to come.
  - A3 is yet to be graded but I anticipate higher scores than A1 and A2 because you had the tests.
  - A2: You can earn back half of any unit test correctness points you lost. Deadline to submit revised code is in a week.

Goals

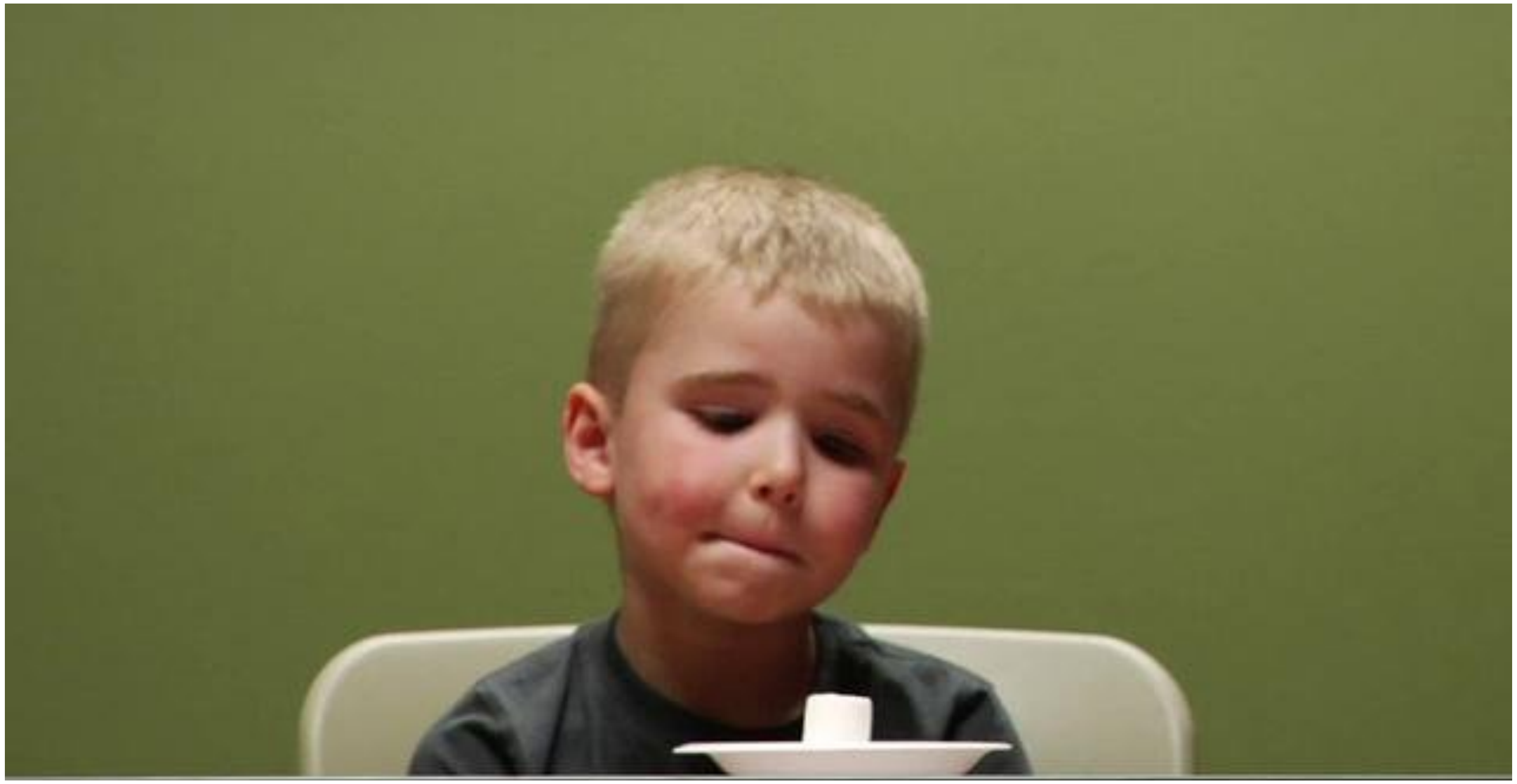
- Be able to run Kruskal’s algorithm and Prim’s algorithm on a graph on paper.
- Understand how to implement Prim’s algorithm using a priority queue of edges.

Finding a MST: Greedy algorithms

- Kruskal's Algorithm:
  - Add the **minimum-weight edge** that does not form a cycle.
- Prim's Algorithm:
  - Add the **minimum-weight edge** from the current spanning tree that does not form a cycle.

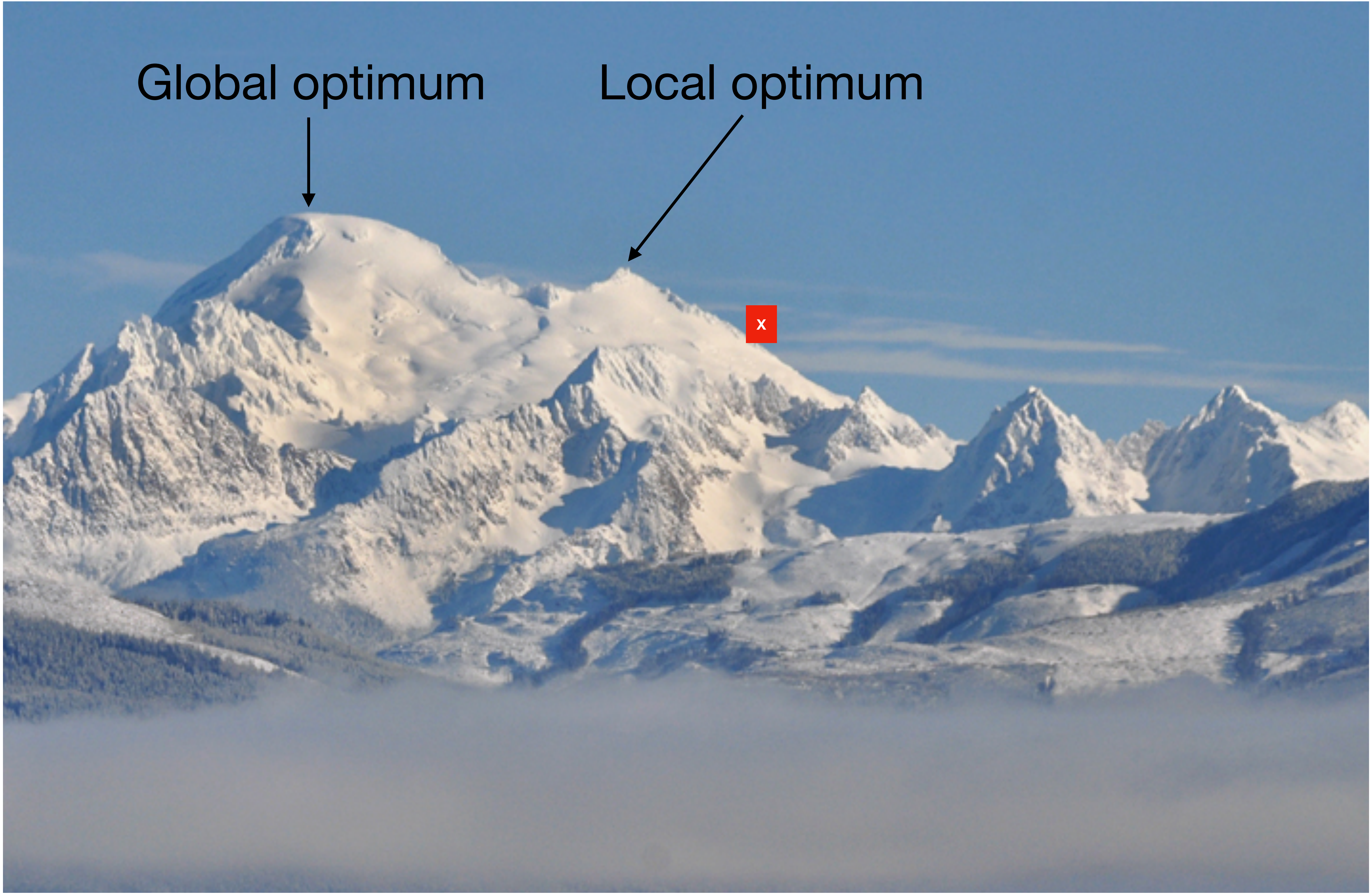
These are greedy algorithms.

Greedy Algorithms



Greedy Algorithms

- Algorithms that fail the marshmallow test.



Starting at x, climb to the highest point.
Greedy algorithm: walk uphill until you can't go up any more.

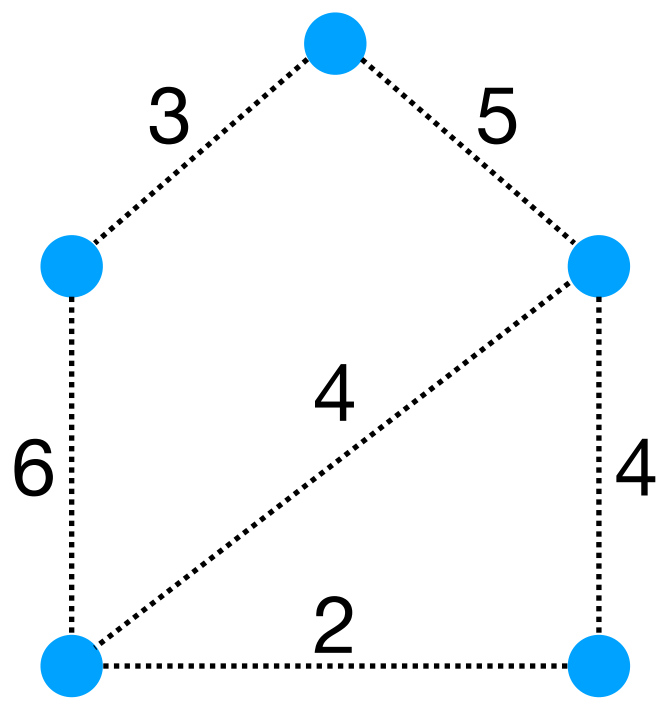
Greedy algorithms

```
/** Returns the smallest possible Set of
 *  coins with total value n */
makeChange(n):
  Set<Coin> change;
  while n > 0:
    Coin c = the largest Coin with value < n
    change.add(c)
    n = n - c
  return change;
```

This is a greedy algorithm.

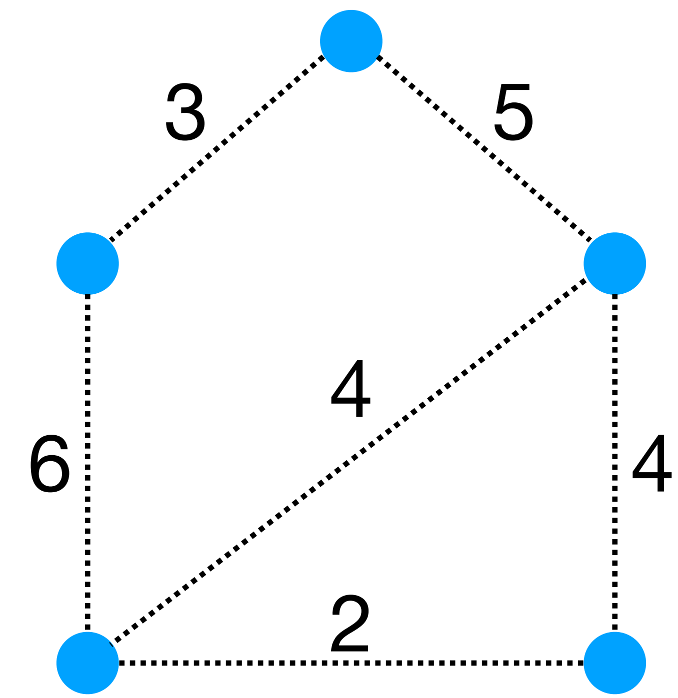
- With US coin values (1, 5, 10, 25), it works.
- With other coin systems, it doesn’t: try out (1,5,7)
- <u>For what coin systems does the greedy algorithm work?</u>

Kruskal and Prim



Kruskal

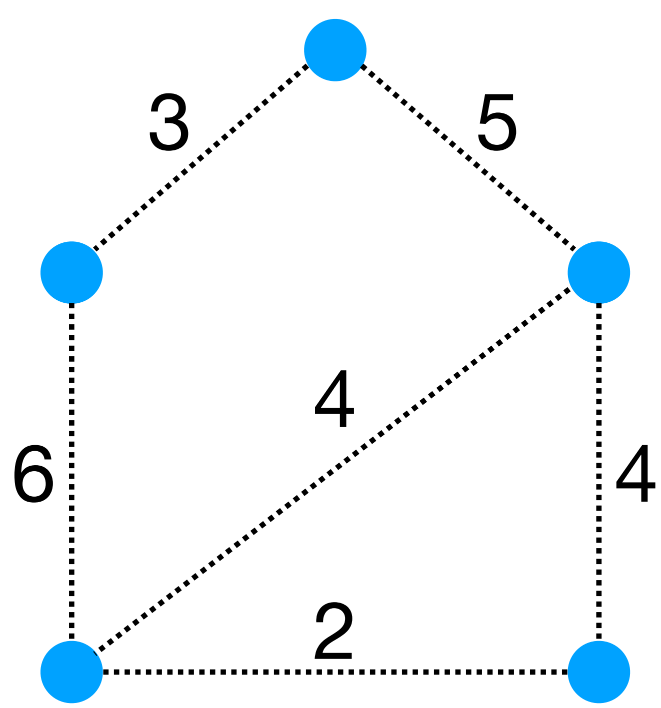
Add the smallest-weight edge that does not introduce a cycle.



Prim

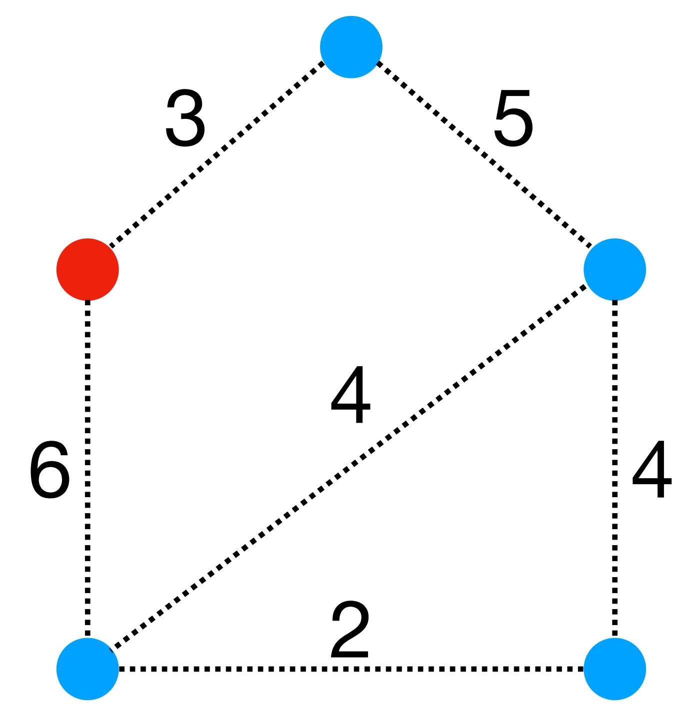
Add the smallest-weight edge with one endpoint in the current tree.

Kruskal and Prim



Kruskal

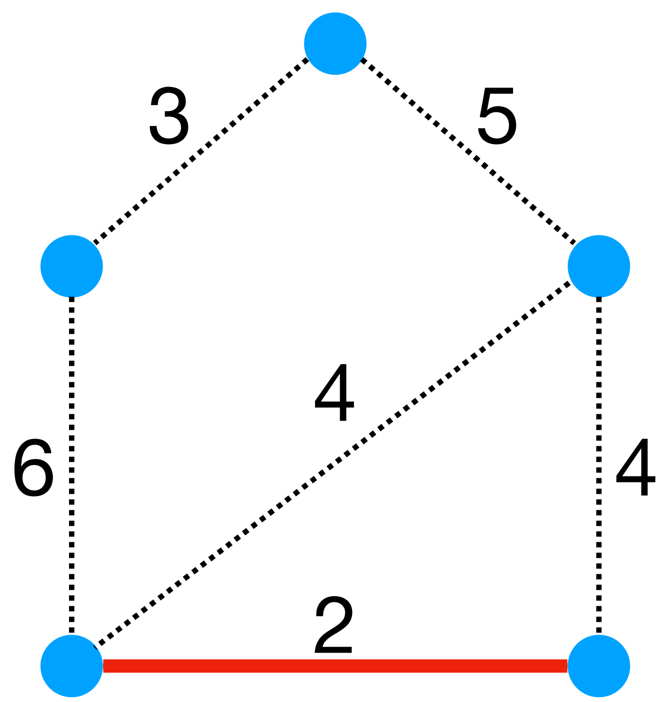
Add the smallest-weight edge that does not introduce a cycle.



Prim

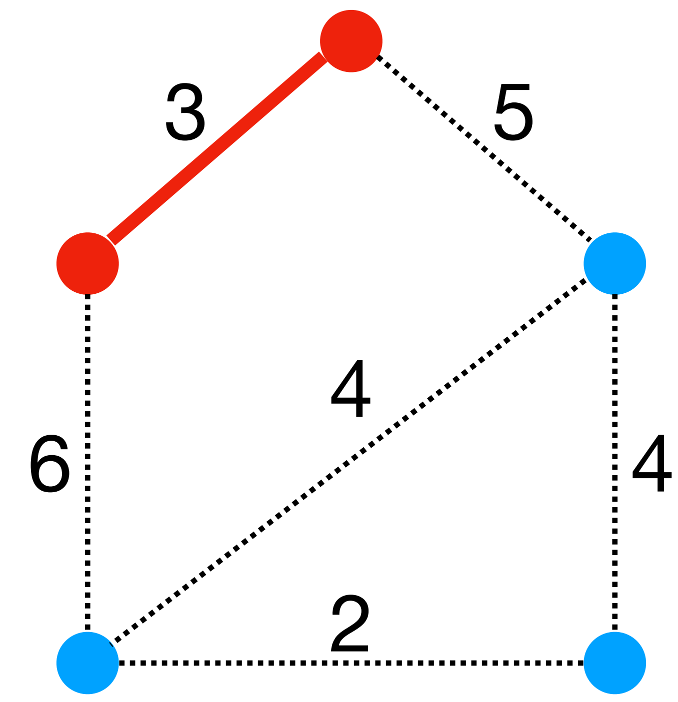
Add the smallest-weight edge with one endpoint in the current tree.

Kruskal and Prim



Kruskal

Add the smallest-weight edge that does not introduce a cycle.

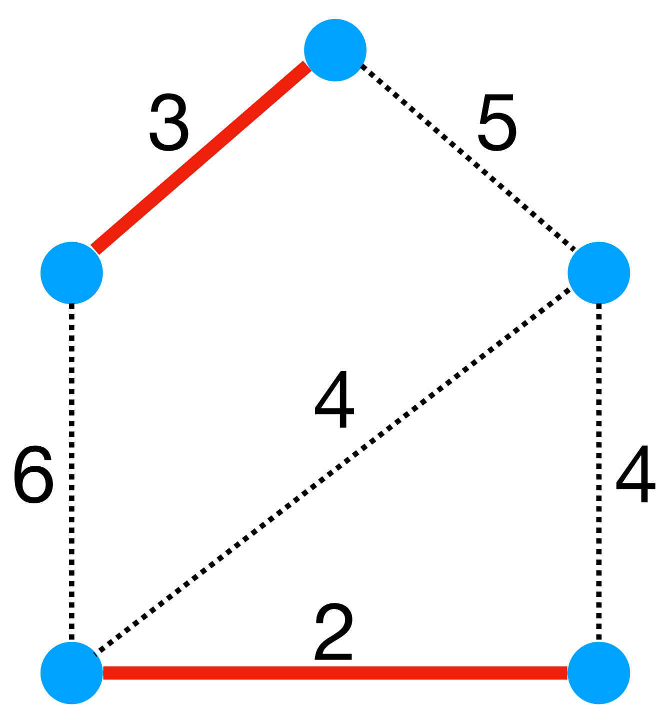


Prim

Add the smallest-weight edge with one endpoint in the current tree.

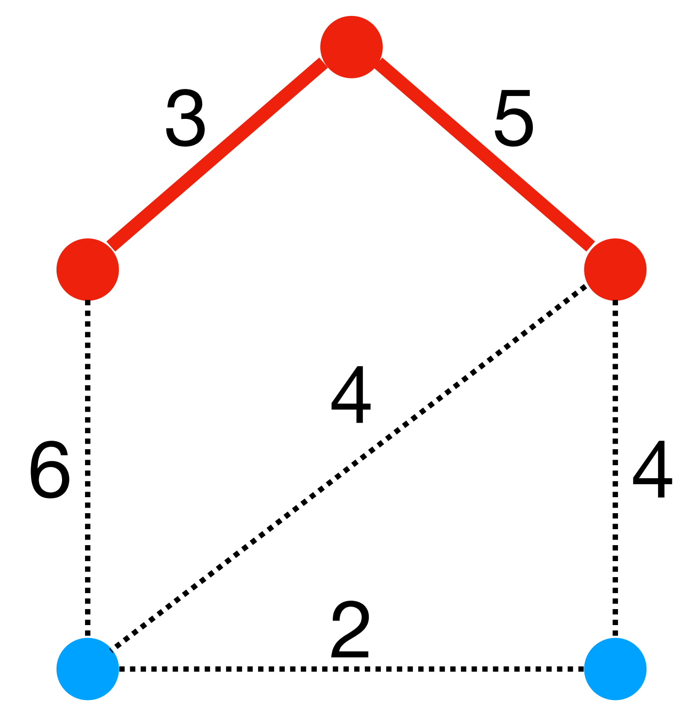
Kruskal and Prim

both algorithms have two choices at this point



Kruskal

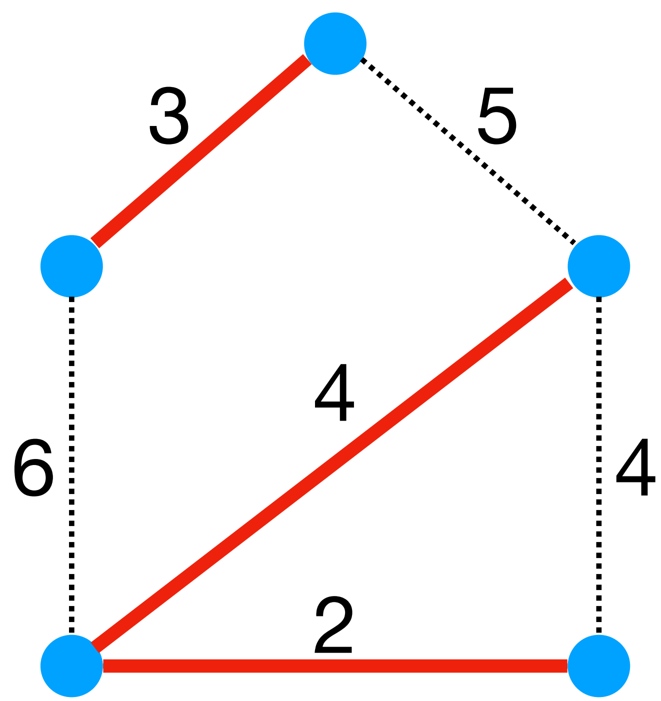
Add the smallest-weight edge that does not introduce a cycle.



Prim

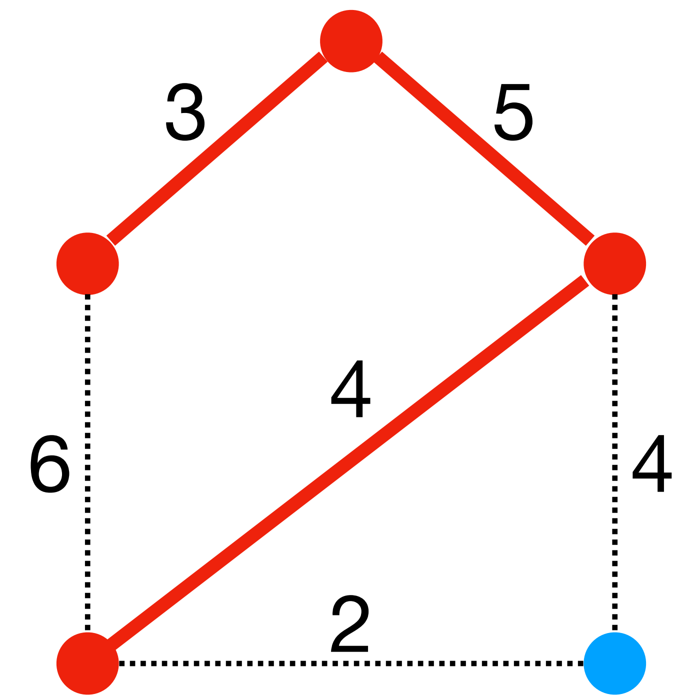
Add the smallest-weight edge with one endpoint in the current tree.

Kruskal and Prim



Kruskal

Add the smallest-weight edge that does not introduce a cycle.

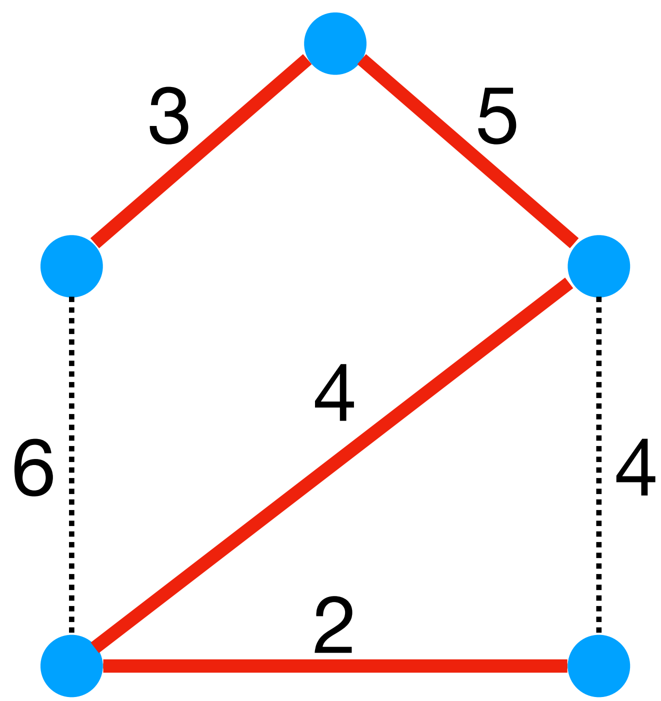


Prim

Add the smallest-weight edge with one endpoint in the current tree.

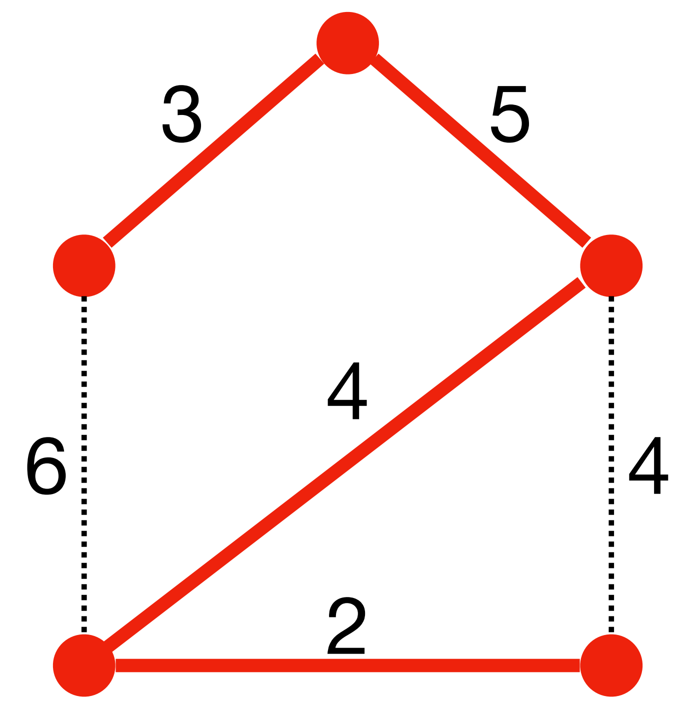
Kruskal and Prim

done!



Kruskal

Add the smallest-weight edge that does not introduce a cycle.



Prim

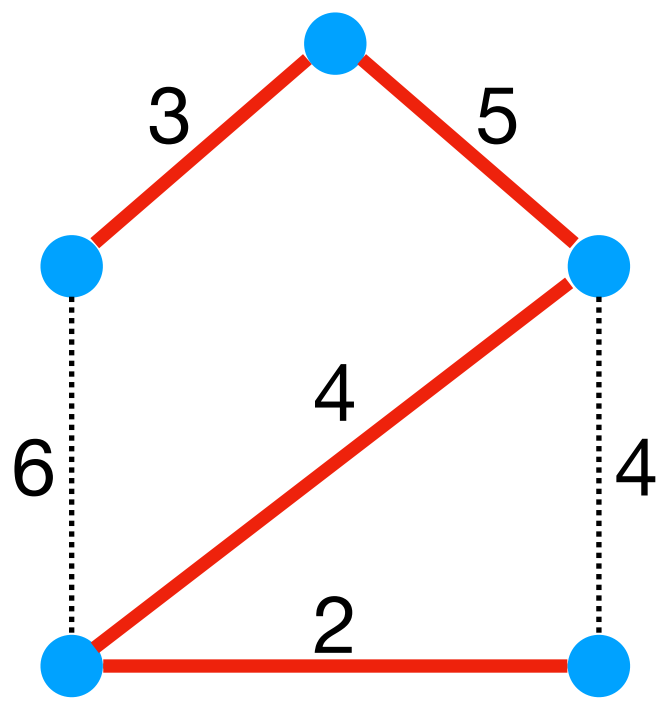
Add the smallest-weight edge with one endpoint in the current tree.

Proof that Kruskal and Prim produce optimal MSTs

- ~~On the final~~ **jk jk** probably covered in CSCI 405
- Structure of the proof:
  - Show that the algorithm maintains an invariant: the invariant is true after initialization, each iteration, and termination.
  - Show that the invariant and the termination condition (e.g., |V'| == |V|) implies that the tree is a MST.

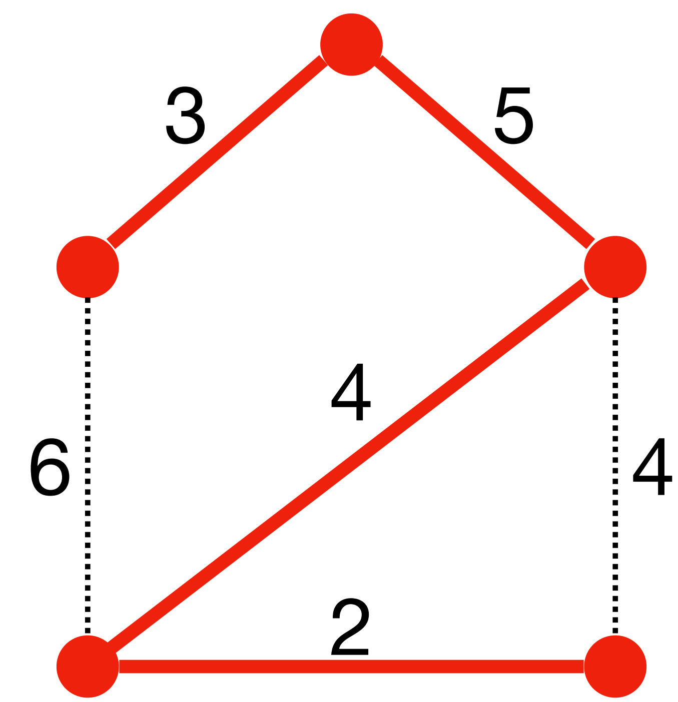
Kruskal and Prim

done!



Kruskal

Add the smallest-weight edge that does not introduce a cycle.



Prim

Add the smallest-weight edge with one endpoint in the current tree.

Kruskal and Prim: Implementation

Kruskal

Add the smallest-weight edge

`Heap<Edge,Double>`

that does not introduce a cycle.

Check with DFS? Sounds expensive.

Prim

Add the smallest-weight edge

`Heap<Edge,Double>`

with one endpoint in the current tree.

Maintain a Set of nodes currently in the tree.

Kruskal and Prim: Implementation

Kruskal	**Prim**
Add the smallest-weight edge	Add the smallest-weight edge

```
Heap<Edge,Double>
```

that does not introduce a cycle.	with one endpoint in the current tree.
Check with DFS? Sounds expensive.	Maintain a Set of nodes currently in the tree.

Prim: Implementation

```
Prim(s):
  V1 = {s}; # vertices in spanning tree
  E1 = {};  # edges in spanning tree

  #inv: (V1, E1) is a tree, V1 ⊆ V, E1 ⊆ E
  while (V1.size() < V.size()) {
    Pick an edge (u,v) with:
        min weight, u in V1,
        v not in V1;
    Add v to V1;
    Add edge (u, v) to E1
  }
```

maintain set of edges with

- u in V1
- v not in V1

Prim: Implementation

```
Prim(s):
  V1 = {s}; # vertices in spanning tree
  E1 = {};  # edges in spanning tree
  S = {Edges leaving s}
  #inv: (V1, E1) is a tree, V1 ⊆ V, E1 ⊆ E
  #     (u,v) is in S iff u is in V1 and v is not
  while (V1.size() < V.size()) {
    (u,v) = remove edge with min weight from S

    Add v to V1;
    Add edge (u, v) to E1
  }
```

Prim: Implementation

```
Prim(s):
  V1 = {s}; # vertices in spanning tree
  E1 = {};  # edges in spanning tree
  S = {Edges leaving s}
  #inv: (V1, E1) is a tree, V1 ⊆ V, E1 ⊆ E
  #     (u,v) is in S iff u is in V1 and v is not
  while (V1.size() < V.size()) {
    (u,v) = remove edge with min weight from S
    Add v to V1;
    Add edge (u, v) to E1
    for each edge (v,w) from v {
      add (v,w) to S if w is not in V1
    }
  }
```

maintain S invariant

Prim: Implementation

```
Prim(s):
  V1 = {s}; # vertices in spanning tree
  E1 = {};  # edges in spanning tree
  S = {Edges leaving s}
  #inv: (V1, E1) is a tree, V1 ⊆ V, E1 ⊆ E
  #     (u,v) is in S iff u is in V1 and v is not
  while (V1.size() < V.size()) {
    (u,v) = remove edge with min weight from S
    Add v to V1;
    Add edge (u, v) to E1
    for each edge (v,w) from v {
      add (v,w) to S if w is not in V1
    }
  }
```