Prim's Algorithm

Choose a vertex w ε V
F={w}
While V-F ≠ ϕ
  Let e be a minimum weight edge that emerges from F
  F=F+{e}

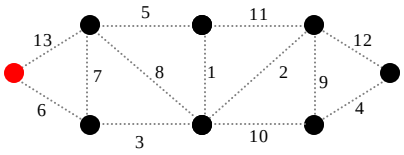
Prim's Algorithm

Choose a vertex w ε V
F={w}
While V-F ≠ ϕ
  Let e be a minimum weight edge that emerges from F
  F=F+{e}

e=(u,v) where u ε F and v ∉ F

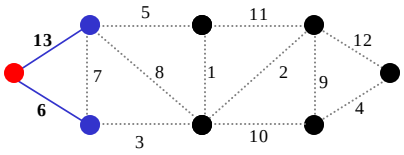
Prim's Algorithm - example

- Put some vertex in F:



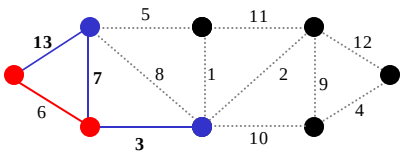
Prim's example cont.

Choose minimum weight edge that emerges from F.



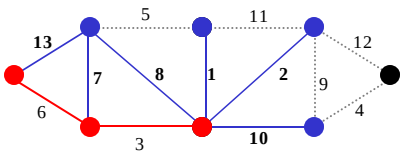
Prim's example cont.

Choose minimum weight edge that emerges from F.



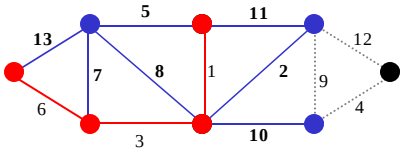
Prim's example cont.

Choose minimum weight edge that emerges from F.



Prim's example cont.

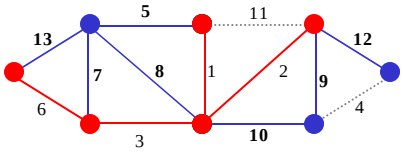
Choose minimum weight edge that emerges from F.



CS140 21-7

Prim's example cont.

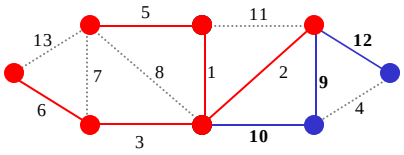
Choose minimum weight edge that emerges from F.



CS140 21-8

Prim's example cont.

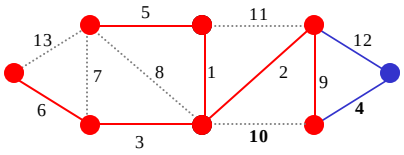
Choose minimum weight edge that emerges from F.



CS140 21-9

Prim's example cont.

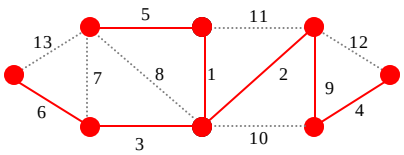
Choose minimum weight edge that emerges from F.



CS140 21-10

Prim's example cont.

Choose minimum weight edge that emerges from F.



CS140 21-11

Prim's Algorithm

- **Is it correct?**
- Is it efficient?

CS140 21-12

Claim

- At each point in the algorithm F is a subgraph of some MST of G.

Loop Invariant

Choose a vertex w ε V
F={w} Claim is True here.
While V-F≠ϕ
  Let e be a minimum weight edge that emerges from F
  F=F+{e}

Loop Invariant

If Claim is True here.

Choose a vertex w ε V
F={w}
While V-F≠ϕ
  Let e be a minimum weight edge with one but not both endpoints in F
  F=F+{e}

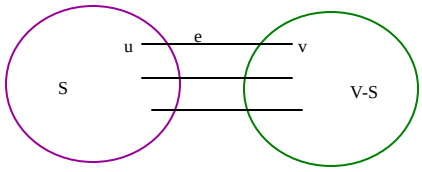
Then it is true here.

Proof of Loop Invariant

- Let T be a MST of G that contains F.
- Assume e=(u,v) is chosen to add to F.
- If e is an edge of T we are done so assume not.

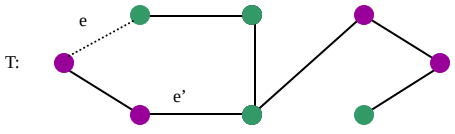
And again the same old story

- e is a minimum weight edge spanning the cut (S,V-S), where S is the set of vertices in F



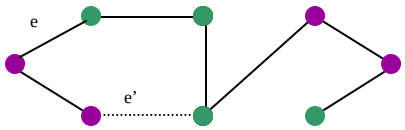
Yada yada …

- T+{e} contains exactly one cycle.
- Some other edge in the cycle also spans the cut (S,V-S)



Etc.

- T+{e}-{e'} is a MST of G that contains F+{e}.



And so …

- At each point in the algorithm F is a subgraph of some MST of G.
- But when the algorithm concludes, F is a spanning tree of G. Hence F is a MST of G.

Prim's Algorithm

- Is it correct? Yes!
- **Is it efficient?**

Prim's Algorithm
Running Time: O(n(?))

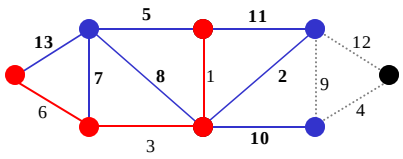
Choose a vertex w ε V
F={w}
While V-F ≠ ϕ

Let e be a minimum weight edge that emerges from F
F=F+{e}

How should we implement this?

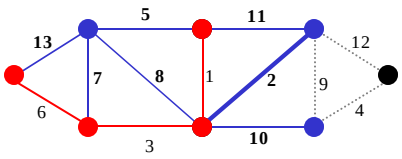
Consider naïve approach first…

- Keep list of emerging edges ordered by weight: 2,5,7,8,10,11,13



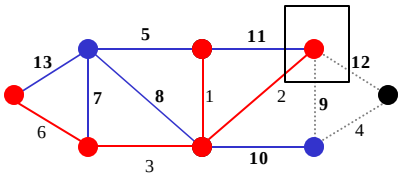
Choose edge to add to F

- List: 2,5,7,8,10,13
- Choose first edge in list to add to F.



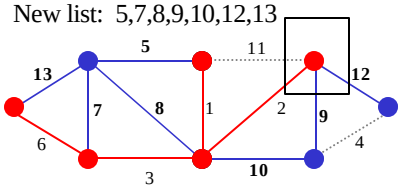
Add edge to F

- And of course its other endpoint



Update list

- Old list: 2,5,7,8,10,13
- Remove 2, 11 and Add 9,12
- New list: 5,7,8,9,10,12,13



Prim's Algorithm Running Time: O(nm)

Choose a vertex w ε V

F={w}

While V-F ≠ ϕ

Let e be a minimum weight edge that emerges from F

F=F+{e}

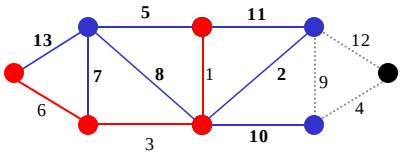
Naïve approach O(m)

Prim's Algorithm

- Is it correct? Yes!
- **Is it efficient?**
  - Remember Kruskal's algorithm is O(m lg(m).
  - How does O(nm) compare to O(m lg(m))?

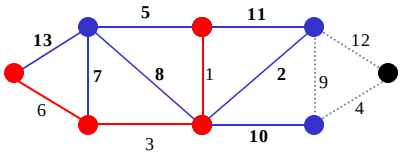
Can we improve on the naïve approach?

- Do we really need to keep track of all of the emerging edges?



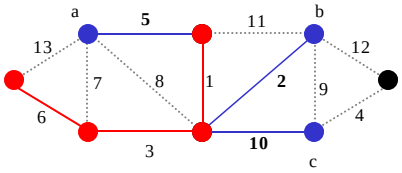
Fringe vertices

- v is a fringe vertex if it is in V-F and it is connected by an edge to a vertex in F



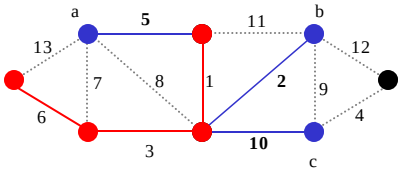
A less naïve approach …

- Keep list of fringe vertices and their minimum weight edge to F: [b,2], [a,5], [c,10]



And even better…

- Keep fringe vertices and their minimum weight edge to F in a HEAP!!!!



Prim's Algorithm
A Better Implementation

Choose a vertex x ε V

F={x}, H=ϕ

For each e=(u,x): Add record [u,e] to heap H keyed on w(e)

While H ≠ ϕ

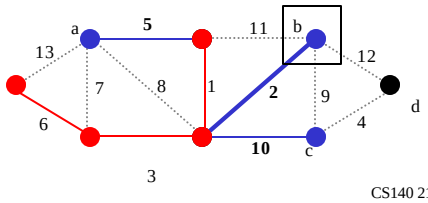
[u,e]=Find-min(H)

Add u and e to F

For each edge incident to u: Update heap

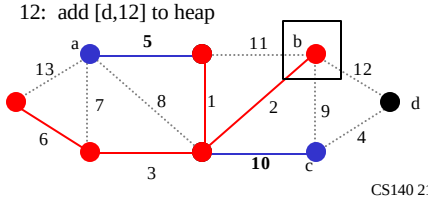
[b,2] at top of heap

- add b and the weight 2 edge to F



Update heap: [b,2],[a,5],[c,10]

- 2: remove [b,2] from heap
- 11: no action required
- 9: update [c,10] in heap
- 12: add [d,12] to heap



Are these standard operations?

- 2: remove [b,2] from heap **Extract-min**
- 11: no action required
- 9: update [c,10] in heap **?????**
- 12: add [d,12] to heap **Insert**

Decrease key

- Next homework assignment: Design decrease key algorithm for heaps that runs in time O(lg(n)).

Prim's Algorithm Running Time

Choose a vertex $x \in V$
$F=\{x\}$, $H=\phi$
For each e=(u,x): **Insert**
While $H \neq \phi$
  [u,e]=**Extract-min**(H)
  Add u and e to F
  For each edge incident to u: **Insert or Decrease-key** or do nothing

Prim's Algorithm Running Time

- Cost of heap operations:
  - Extract-min n O(lg(n)) each
  - Insert n O(lg(n)) each
  - Decrease-key m O(lg(n)) each
  - Do nothing m O(1) each
- Total time is O(m lg(n))

Prim's Algorithm Running Time

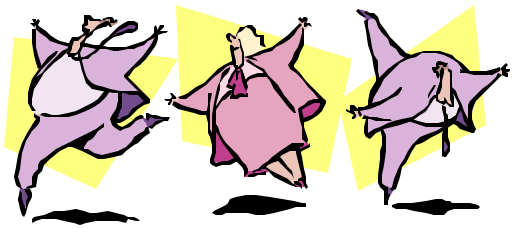
- Cost of heap operations:
  - Extract-min n O(lg(n)) each
  - Insert n O(lg(n)) each
  - Decrease-key m-n O(lg(n)) each
  - Do nothing m O(1) each
- Total time is O(m lg(n))
- Compared to Kruskal's O(m lg(m))…

They are asymptotically the same …

But Wait …Suppose we could decrease-key in time O(1)

- Cost of heap operations:
  - Extract-min n O(lg(n))
  - Insert n O(lg(n))
  - Decrease-key m ~~O(lg(n))~~ O(1)
  - Do nothing m O(1)
- Then total time is O(m + n lg(n))
- Compared to Kruskal's O(m lg(m))…

Bravo, bravo …



Do It With Fibonacci Heaps

Huh?

Don’t worry – it works!