CS5800 – ALGORITHMS

MODULE 7. GREEDY ALGORITHMS - II

Lesson 3: Prim & Kruskal

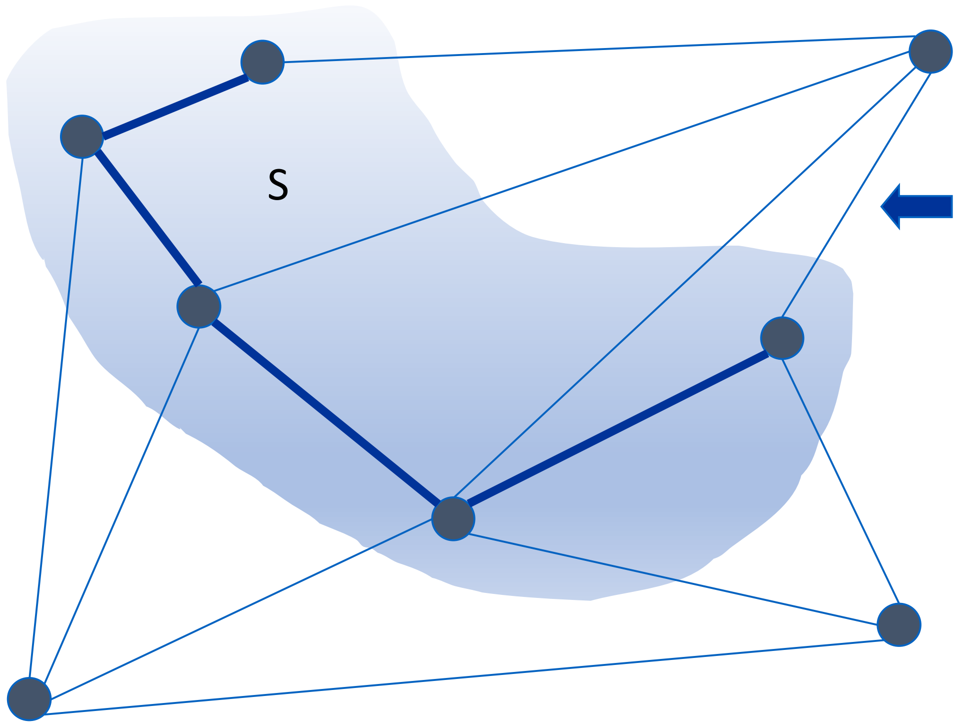
Ravi Sundaram

Topics

- Prim’s algorithm
  - Proof of correctness
  - Implementation & Complexity
- Kruskal’s algorithm
  - Proof of correctness
  - Implementation & Complexity
- Reverse-delete algorithm
- Summary

Prim's Algorithm: Proof of Correctness

- Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]
  - Initialize S = any node.
  - Apply cut property to S.
  - Add min cost edge in cut-set corresponding to S to T, and add one new explored node u to S.



Prim's Algorithm: Implementation

```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q
   Initialize set of explored nodes S ← ϕ

   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and ($c_e$ < a[v]))
             decrease priority a[v] to $c_e$
}
```

- 1 x buildheap
- n x deletemin
- m x changekey

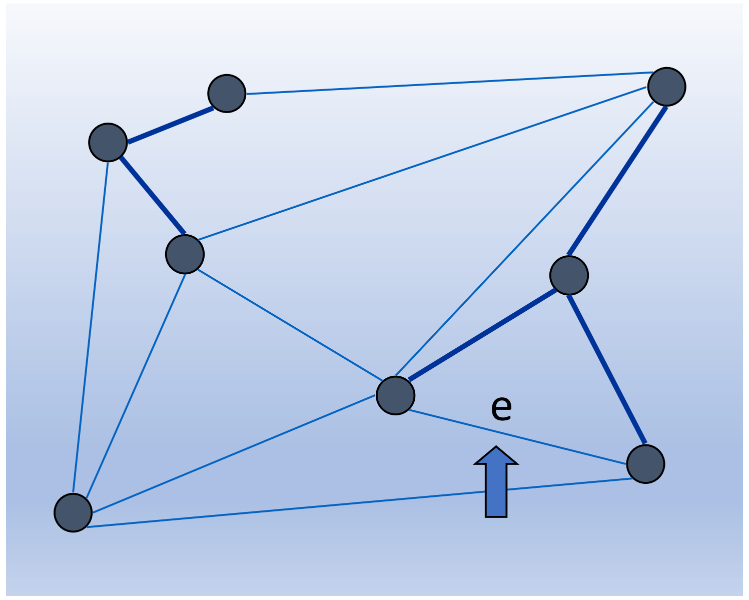
Prim's Algorithm: Complexity

- Implementation. Use a priority queue a la Dijkstra.
  - Maintain set of explored nodes S.
  - For each unexplored node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.

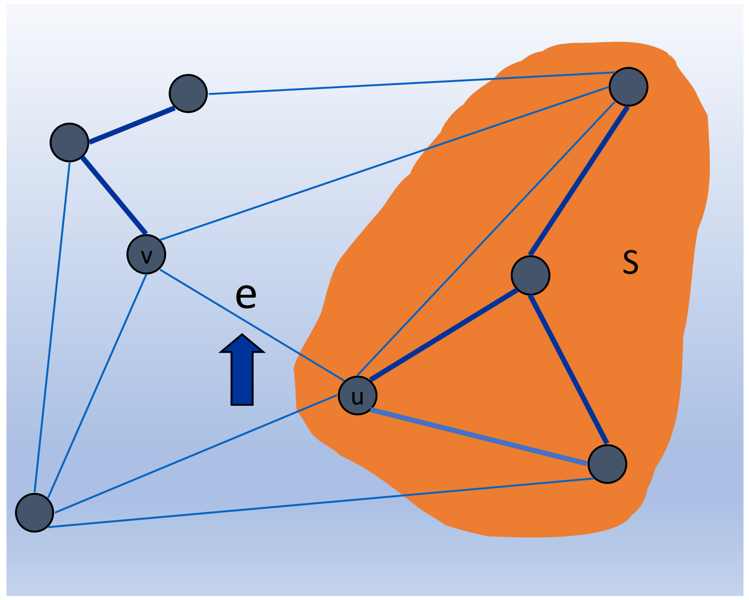
- O(m log n) with a binary heap.

Kruskal's Algorithm: Proof of Correctness

- Kruskal's algorithm. [Kruskal, 1956]
  - Consider edges in ascending order of weight.
  - Case 1: If adding e to T creates a cycle, discard e according to cycle property
  - Case 2: Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.



Case 1



Case 2

Kruskal's Algorithm: Implementation:

```
Kruskal(G, c) {
   Sort edges by weights so that $c_1 \le c_2 \le \dots \le$
$c_m$.
   T ← $\phi$

   foreach ($u \in V$) make a set containing
singleton u
                                  are u and v in different connected components?
   for i = 1 to m
      (u,v) = $e_i$
      if (u and v are in different sets) {
         T ← T $\cup$ {$e_i$}
         merge the sets containing u and v
      }                           merge two components
   return T
}
```

- Sort m edges
- n x makeset
- m x find & unions

Kruskal's Algorithm: Complexity

- Implementation. Use the union-find data structure.
  - Build set T of edges in the MST.
  - Maintain set for each connected component.

- $O(m \log n)$ for sorting and $O(m\, \alpha\underbrace{(m, n)}_{\text{essentially a constant}})$ for union-find.

$m \leq n^2 \Rightarrow \log m$ is $O(\log n)$

Reverse-delete Algorithm: Correctness & Complexity

- Reverse-delete algorithm. [Kruskal, 1956]
  - Consider edges in descending order of weight.
  - If deleting e does not disconnect G then discard e according to cycle property
  - Else retain the edge according to cut property
  - At end you will be left with the MST

  - Implementation requires checking connectivity
  - Best is $O(m \log n (\log\log n)^3)$. [Thorup, 2000]

Distinct edge weights – removing assumption

- To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.
- Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs. e.g., if all edge costs are integers, perturbing cost of edge e_i by i / n^2
- Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

```
boolean less(i, j) {
   if      (cost(e_i) < cost(e_j)) return
true
   else if (cost(e_i) > cost(e_j)) return
false
   else if (i < j)                  return
true
   else                             return
false
}
```

Summary

- Three basic greedy approaches
- Prim’s: grow a tree from a node a la Dijkstra. Implementation uses binary heap. Proof uses cut/cycle property.
- Kruskal’s: add edges in increasing order of weight unless addition creates a cycle. Implementation requires sorting and uses Union-Find. Proof uses cut/cycle property.
- Reverse-delete: remove edges in decreasing order so long as the graph doesn’t disconnect. Proof uses cut/cycle property.
- Main Takeaway: Several greedy approaches work for finding the MST. The proofs are simplified by assuming the edges have distinct weights, and this assumption is easily relaxed.