

MA/CSSE 473

Day 36

Disjoint Sets

**Kruskal and Prim
Algorithms**



MA/CSSE 473 Day 36

- HW 14 due now
 - document Includes a preview of a tough problem from HW 15
- HW 15 due Tuesday
- Don't forget the Boyer-Moore demonstration animation, due Nov 20
- **Student Questions**
- Kruskal's algorithm
- Disjoint Sets



Recap: The Greedy Rule

- Whenever a choice is to be made, pick the one that seems best for the moment, without taking future choices into consideration.
- For example, a greedy Scrabble player will simply maximize the score for each turn, never saving any “good” letters for possible better plays later.
 - Doesn’t necessarily optimize score for entire game.



MST lemma

Let G be a weighted connected graph with a MST T ;
let G' be any subgraph of T , and let C be any connected component
of G' .

If we add to C an edge $e=(v,w)$ that has minimum-weight among all
edges that have one vertex in C and the other vertex not in C ,

then G has an MST that contains the union of G' and e .

[WLOG v is the vertex of e that is in C , and w is not in C]

**Proof: Done last
time.**



Recall Kruskal's algorithm

- To find a MST:
- Start with a graph containing all of G 's n vertices and none of its edges.
- for $i = 1$ to $n - 1$:
 - Among all of G 's edges that can be added without creating a cycle, add one that has minimal weight.

Does this algorithm produce an MST for G ?



Does Kruskal produce a MST?

- Claim: After every step of Kruskal's algorithm, we have a set of edges that is part of an MST
- Base case ...
- Induction step:
 - Induction Assumption: before adding an edge we have a subgraph of an MST
 - We must show that after adding the next edge we have a subgraph of an MST
 - Suppose that the most recently added edge is $e = (v, w)$.
 - Let C be the component (of the “before adding e ” MST subgraph) that contains v
 - Note that there must be such a component and that it is unique.
 - Are all of the conditions of MST lemma met?
 - Thus the new graph is a subgraph of an MST of G



Does Prim produce an MST?

- Proof similar to Kruskal.
- It's done in the textbook



Data Structures for Kruskal

- A sorted list of edges
- Disjoint subsets of vertices, representing the connected components at each stage.
 - Start with n subsets, each containing one vertex.
 - End with one subset containing all vertices.
- Disjoint Set ADT has 3 operations:
 - `makeset(i)`: creates a singleton set containing i .
 - `findset(i)`: returns a "canonical" member of its subset.
 - I.e., if i and j are elements of the same subset,
`findset(i) == findset(j)`
 - `union(i, j)`: merges the subsets containing i and j into a single set.



Kruskal Algorithm

Assume vertices are numbered 1...n

Sort edgelist by weight

```
for i = 1..n: makeset(i)
```

```
i, count, result = 1, 0, []
```

```
while count < n-1:
```

```
    if findset(edgelist[i].v) !=
```

```
        findset(edgelist[i].w):
```

```
        result += [edgelist[i]]
```

```
        count += 1
```

```
        union(edgelist[i].v, edgelist[i].w)
```

```
    i += 1
```

```
return result
```

What can we say about efficiency of this algorithm (in terms of n and m)?



Disjoint set ADT

- A collection of numbers (taken from the set $\{1, \dots, n\}$) are partitioned into disjoint sets, each containing a (marked) representative element
- Operations:
 - **makeSet(i)**: Create a singleton set containing the number i
 - **findSet(i)**: Find the unique representative for the set that contains i . Note that i and j are in the same set iff **$\text{findSet}(i) == \text{findSet}(j)$**
 - **union(i, j)**: Combine the sets containing i and j into a single set (before the union, i and j must be in different sets)



Example of operations

- makeset (1)
- makeset (2)
- makeset (3)
- makeset (4)
- makeset (5)
- makeset (6)
- union(4, 6)
- union (1,3)
- union(4, 5)
- findset(2)
- findset(5)

What are the sets after these operations?



Set Representation

- Each disjoint set is a tree, with the "marked" element as its root
- Efficient representation of the trees:
 - an array called *parent*
 - $\text{parent}[i]$ contains the index i 's parent.
 - If i is the root, $\text{parent}[i]=i$
- Show the parent array for the previous example
- The parent array contains all of the info that we need for our first representation



Using this representation

```
def makeset1(i):
```

- `makeset(i):` `parent[i] = i`

- `findset(i):`

```
def findset1(i):
```

```
    while i != parent[i]:
```

```
        i = parent[i]
```

```
    return i
```

- `mergetrees(i,j):`

- assume that i and j are the marked elements from different sets.

```
def mergetrees1(i, j):
```

```
    parent[i] = j
```

- `union(i,j)`

- assume that i and j are elements from different sets

```
def union1(i, j):
```

```
    mergetrees1(findset1(i), findset1(j))
```

