CSE 421
Algorithms

Richard Anderson

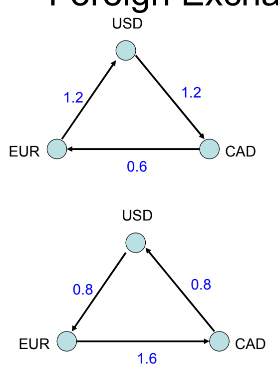
Lecture 11

Minimum Spanning Trees

Announcements

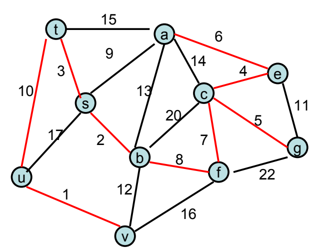
- Monday – Class in EE1 003 (no tablets)

Foreign Exchange Arbitrage



	USD	EUR	CAD
USD	------	0.8	1.2
EUR	1.2	------	1.6
CAD	0.8	0.6	-----

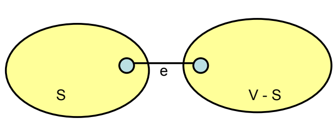
Minimum Spanning Tree



Temporary Assumption: Edge costs distinct

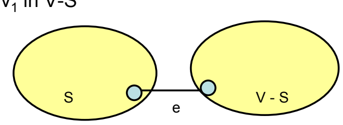
Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct
- Let S be a subset of V, and suppose e = (u, v) be the minimum cost edge of E, with u in S and v in V-S
- e is in every minimum spanning tree
  - Or equivalently, if e is not in T, then T is not a minimum spanning tree



Proof

- Suppose T is a spanning tree that does not contain e
- Add e to T, this creates a cycle
- The cycle must have some edge $e_1 = (u_1, v_1)$ with u_1 in S and v_1 in V-S



Why is e lower cost than e1?

- $T_1 = T – \{e_1\} + \{e\}$ is a spanning tree with lower cost
- Hence, T is not a minimum spanning tree

Optimality Proofs

- Prim's Algorithm computes a MST
- Kruskal's Algorithm computes a MST

Reverse-Delete Algorithm

- Lemma: The most expensive edge on a cycle is never in a minimum spanning tree

Dealing with the distinct cost assumption

- Force the edge weights to be distinct
  - Add small quantities to the weights
  - Give a tie breaking rule for equal weight edges

MST Fun Facts

- The minimum spanning tree is determined only by the order of the edges – not by their magnitude
- Finding a maximum spanning tree is just like finding a minimum spanning tree

Divide and Conquer

```
Array Mergesort(Array a){
        n = a.Length;
        if (n <= 1)
                return a;
        b = Mergesort(a[0..n/2]);
        c = Mergesort(a[n/2+1 .. n-1]);
        return Merge(b, c);
```

Algorithm Analysis

- Cost of Merge
- Cost of Mergesort

$T(n) = 2T(n/2) + cn; T(1) = c;$

Recurrence Analysis

- Solution methods
  - Unrolling recurrence
  - Guess and verify
  - Plugging in to a "Master Theorem"

A better mergesort (?)

- Divide into 3 subarrays and recursively sort
- Apply 3-way merge

$T(n) = aT(n/b) + f(n)$

$T(n) = T(n/2) + cn$

$T(n) = 4T(n/2) + cn$

$T(n) = 2T(n/2) + n^2$

$T(n) = 2T(n/2) + n^{1/2}$

Recurrences

- Three basic behaviors
  - Dominated by initial case
  - Dominated by base case
  - All cases equal – we care about the depth