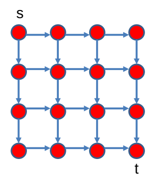
CSE 417
Algorithms and Complexity

Autumn 2020
Lecture 14
MST + Recurrences

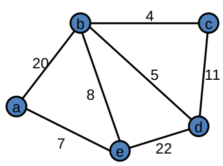
Announcements

- Homework
  - Assignment will include a sample midterm
  - Programming
    - Shortest Path and Bottleneck Paths on Grid Graphs with random edge lengths
    - What is the expected length of an s-t path?
    - What is the expected bottleneck length of an s-t path



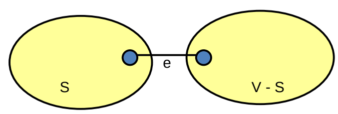
Greedy Algorithms for Minimum Spanning Tree

- [Prim] Extend a tree by including the cheapest out going edge
- [Kruskal] Add the cheapest edge that joins disjoint components



Edge inclusion lemma

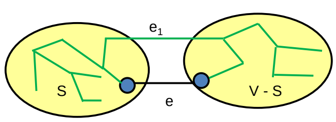
- Let S be a subset of V, and suppose e = (u, v) is the minimum cost edge of E, with u in S and v in V-S
- e is in every minimum spanning tree of G
  - Or equivalently, if e is not in T, then T is not a minimum spanning tree



e is the minimum cost edge between S and V-S

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



- $T_1 = T - \{e_1\} + \{e\}$ 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

- Show that when an edge is added to the MST by Prim or Kruskal, the edge is the minimum cost edge between S and V-S for some set S.

Prim's Algorithm

```
S = { };    T = { };
while S != V
        choose the minimum cost edge
        e = (u,v), with u in S, and v in V-S
        add e to T
        add v to S
```

Prove Prim's algorithm computes an MST

- Show an edge e is in the MST when it is added to T

Kruskal's Algorithm

Let $C = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$; T = { }

while |C| > 1

  Let e = (u, v) with u in C_i and v in C_j be the minimum cost edge joining distinct sets in C

  Replace C_i and C_j by $C_i \cup C_j$

  Add e to T

Prove Kruskal's algorithm computes an MST

- Show an edge e is in the MST when it is added to T

Divide and Conquer

- Recurrences, Sections 5.1 and 5.2
- Algorithms
  - Fast Matrix Multiplication
  - Counting Inversions (5.3)
  - Closest Pair (5.4)
  - Multiplication (5.5)

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

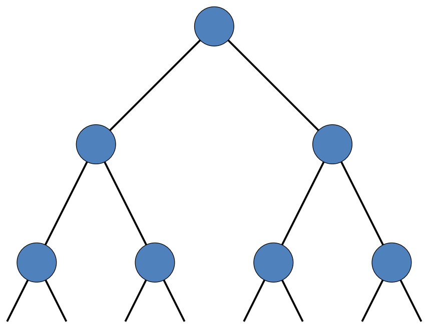
- Cost of Merge
- Cost of Mergesort

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

Recurrence Analysis

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

Unrolling the recurrence



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

Substitution

Prove $T(n) <= n (\log_2 n + 1)$ for $n >= 1$

Induction:
Base Case:

Induction Hypothesis:

A better mergesort (?)

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

What is the recurrence?

Unroll recurrence for $T(n) = 3T(n/3) + n$

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

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

Where does this recurrence arise?

Solving the recurrence exactly

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

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

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

Recurrences

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