CSEP 521
Applied Algorithms

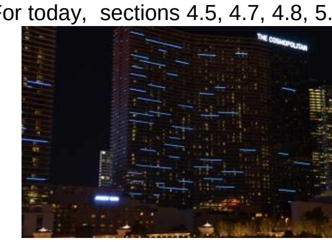
Richard Anderson

Winter 2013

Lecture 4

Announcements

- Reading
  - For today, sections 4.5, 4.7, 4.8, 5.1, 5.2



Interval Scheduling

Highlights from last lecture

- Greedy Algorithms
- Dijkstra's Algorithm

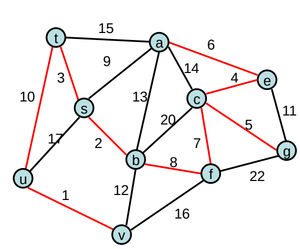
Today

- Minimum spanning trees
- Applications of Minimum Spanning trees
- Huffman codes
- Homework solutions
- Recurrences

Minimum Spanning Tree

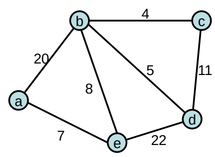
- Introduce Problem
- Demonstrate three different greedy algorithms
- Provide proofs that the algorithms work

Minimum Spanning Tree



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
- [ReverseDelete] Delete the most expensive edge that does not disconnect the graph

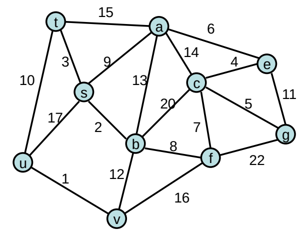


Greedy Algorithm 1 Prim's Algorithm

- Extend a tree by including the cheapest out going edge

Construct the MST with Prim's algorithm starting from vertex a

Label the edges in order of insertion

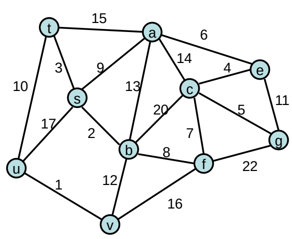


Greedy Algorithm 2 Kruskal's Algorithm

- Add the cheapest edge that joins disjoint components

Construct the MST with Kruskal's algorithm

Label the edges in order of insertion

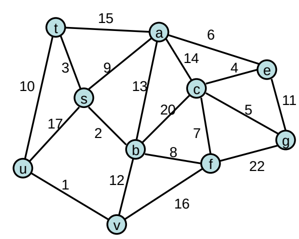


Greedy Algorithm 3 Reverse-Delete Algorithm

- Delete the most expensive edge that does not disconnect the graph

Construct the MST with the reverse-delete algorithm

Label the edges in order of removal

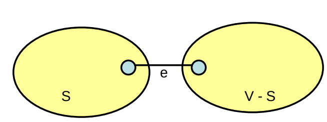


Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct

Edge inclusion lemma

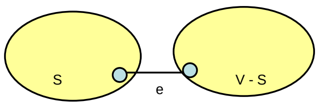
- Let S be a subset of V, and suppose e = (u, v) is the minimum cost edge of E, with u in S and v in V-S
- e is in every minimum spanning tree of G
  - Or equivalently, if e is not in T, then T is not a minimum spanning tree



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

Dijkstra's Algorithm for Minimum Spanning Trees

S = {}; d[s] = 0; d[v] = infinity for v != s

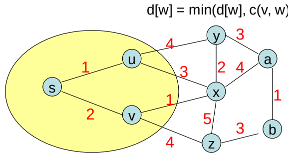
While S != V

Choose v in V-S with minimum d[v]

Add v to S

For each w in the neighborhood of v

d[w] = min(d[w], c(v, w))



Kruskal's Algorithm

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

Reverse-Delete Algorithm

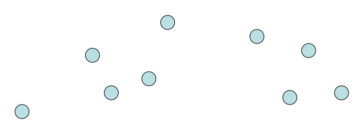
- Lemma: The most expensive edge on a cycle is never in a minimum spanning tree

Dealing with the assumption of no equal weight edges

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

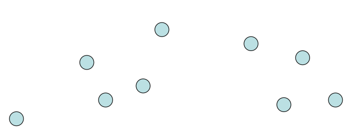
Application: Clustering

- Given a collection of points in an r-dimensional space, and an integer K, divide the points into K sets that are closest together

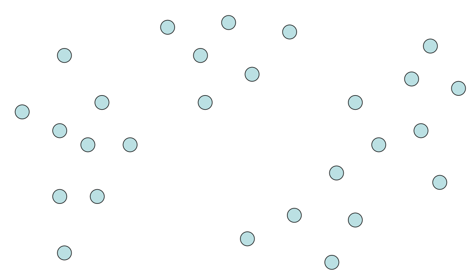


Distance clustering

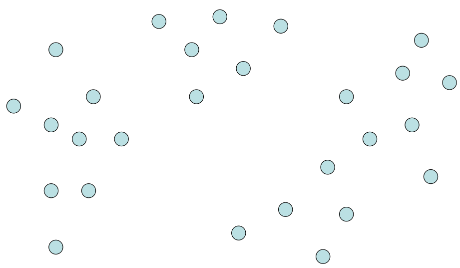
- Divide the data set into K subsets to maximize the distance between any pair of sets
  - dist (S_1, S_2) = min {dist(x, y) | x in S_1, y in S_2}



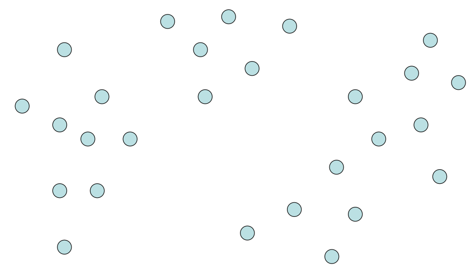
Divide into 2 clusters



Divide into 3 clusters



Divide into 4 clusters



Distance Clustering Algorithm

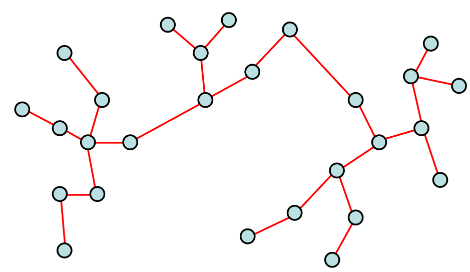
Let C = {{v_1}, {v_2},. . ., {v_n}}; T = { }

while |C| > K

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 U C_j

K-clustering



Huffman Codes

- Given a set of symbols of known frequency, encode in binary to minimize the average length of a message

S = {a, b, c, d}, f(a) = .4, f(b) = .3, f(c) = .2, f(d) = .1

Prefix codes

- A code is a prefix code, if there is no pair of code words X and Y, where X is a prefix of Y
- A prefix code can be decoded with a left to right scan
- A binary prefix code can be represented as a binary tree

Optimal prefix code

- Given a set of symbols with frequencies for the symbols, design a prefix code with minimum average length
- ABL(Code): Average Bits per Letter

Properties of optimal codes

- The tree for an optimal code is full
- If $f(x) \leq f(y)$ then $depth(x) \geq depth(y)$
- The two nodes of lowest frequency are at the same level
- There is an optimal code where the two lowest frequency words are siblings

Huffman Algorithm

- Pick the two lowest frequency items
- Replace with a new item with there combined frequencies
- Repeat until done

Correctness proof (sketch)

- Let y, z be the lowest frequency letters that are replaced by a letter w
- Let T be the tree constructed by the Huffman algorithm, and T' be the tree constructed by the Huffman algorithm when y, z are replaced by w
  - ABL(T') = ABL(T) – f(w)

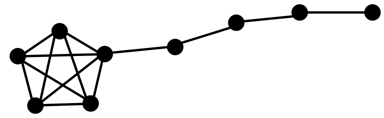
Correctness proof (sketch)

- Proof by induction
- Base case, n = 2
- Suppose Huffman algorithm is correct for n symbols
- Consider an n+1 symbol alphabet . . .

Homework problems

Exercise 8, Page 109

Prove that for any c, there is a graph G such that Diag(G) ≥ c APD(G)

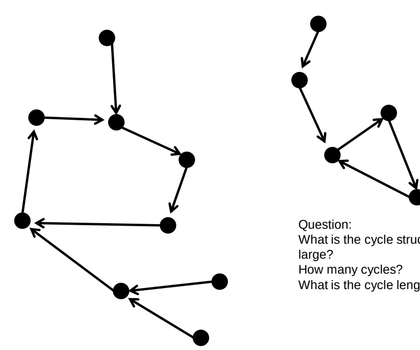


Exercise 12, Page 112

- Given info of the form P_i died before P_j born and P_i and P_j overlapped, determine if the data is internally consistent

Programming Problem

Random out degree one graph



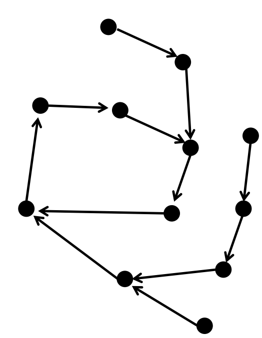
Question:
What is the cycle structure as N gets large?
How many cycles?
What is the cycle length?

Topological Sort Approach

- Run topological sort
  - Determine cycles
  - Order vertices on branches
- Label vertices on the cycles
- Label vertices on branches computing cycle weight

Pointer chasing algorithm

- Label vertices with the number of their cycle
- Pick a vertex, follow chain of pointers
  - Until a labeled vertex is reached
  - Until a new cycle is discovered
- Follow chain of vertices a second time to set labels



The code . . .

```
void MarkCycle(int v,
               CycleStructure cycles,
               bool[] mark,
               sbyte[] cycle) {
    if (mark[v] == true)
        return;

    int y = v;
    int x;
    do {
        x = y;
        y = next[x];
        mark[x] = true;
    }
    while (mark[y] == false);

    int cycleID;
    if (cycle[y] == -1) {
        cycleID = cycles.AddCycle();
        for (int a = y; a != x; a = next[a]) {
            cycle[a] = (sbyte) cycleID;
            cycles.AddCycleVertex(cycleID);
        }
        cycle[x] = (sbyte) cycleID;
        cycles.AddCycleVertex(cycleID);
    }
    else
        cycleID = cycle[y];

    for (int a = v; cycle[a] == -1; a = next[a]) {
        cycle[a] = (sbyte) cycleID;
        cycles.AddBranchVertex(cycleID);
    }
}
```

Results from Random Graphs

What is the length of the longest cycle?

How many cycles?

Recurrences

Divide and Conquer

- Recurrences, Sections 5.1 and 5.2
- Algorithms
  - Counting Inversions (5.3)
  - Closest Pair (5.4)
  - Multiplication (5.5)
  - FFT (5.6)

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

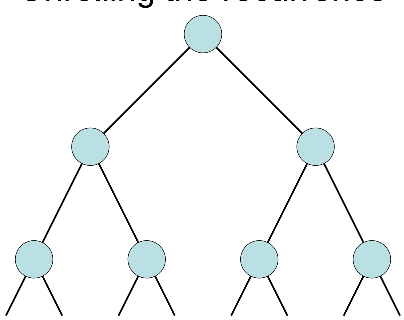
- Cost of Merge
- Cost of Mergesort

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

Recurrence Analysis

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

Unrolling the recurrence



Substitution

Prove $T(n) <= cn (\log_2 n + 1)$ 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 $T(n) = 3T(n/3) + dn$

Recurrence Examples

- $T(n) = 2\,T(n/2) + cn$
  - $O(n \log n)$
- $T(n) = T(n/2) + cn$
  - $O(n)$

- More useful facts:
  - $\log_k n = \log_2 n / \log_2 k$
  - $k^{\log n} = n^{\log k}$

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

Recursive Matrix Multiplication

Multiply 2 x 2 Matrices:

$$\begin{vmatrix} r & s \\ t & u \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} e & g \\ f & h \end{vmatrix}$$

r = ae + bf
s = ag + bh
t = ce + df
u = cg + dh

A N x N matrix can be viewed as a 2 x 2 matrix with entries that are (N/2) x (N/2) matrices.

The recursive matrix multiplication algorithm recursively multiplies the (N/2) x (N/2) matrices and combines them using the equations for multiplying 2 x 2 matrices

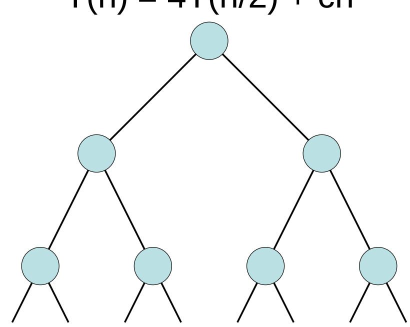
Recursive Matrix Multiplication

- How many recursive calls are made at each level?
- How much work in combining the results?
- What is the recurrence?

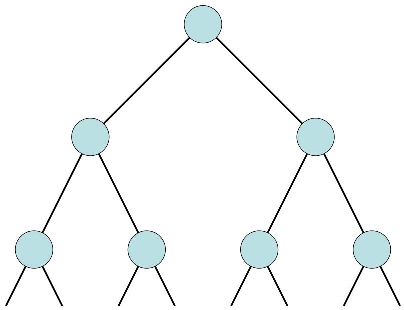
What is the run time for the recursive Matrix Multiplication Algorithm?

- Recurrence:

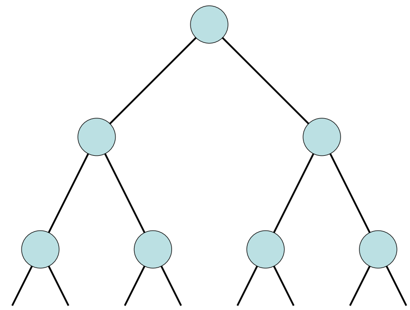
$T(n) = 4T(n/2) + cn$



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



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



Recurrences

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

What you really need to know about recurrences

- Work per level changes geometrically with the level
- Geometrically increasing ($x > 1$)
  - The bottom level wins
- Geometrically decreasing ($x < 1$)
  - The top level wins
- Balanced ($x = 1$)
  - Equal contribution

Classify the following recurrences (Increasing, Decreasing, Balanced)

- $T(n) = n + 5T(n/8)$
- $T(n) = n + 9T(n/8)$
- $T(n) = n^2 + 4T(n/2)$
- $T(n) = n^3 + 7T(n/2)$
- $T(n) = n^{1/2} + 3T(n/4)$

Strassen's Algorithm

Multiply 2 x 2 Matrices:

$$\left|\begin{matrix} r & s \\ t & u \end{matrix}\right| = \left|\begin{matrix} a & b \\ c & d \end{matrix}\right| \left|\begin{matrix} e & g \\ f & h \end{matrix}\right|$$

$r = p_1 + p_4 – p_5 + p_7$

$s = p_3 + p_5$

$t = p_2 + p_5$

$u = p_1 + p_3 – p_2 + p_7$

Where:

$p_1 = (b + d)(f + g)$

$p_2 = (c + d)e$

$p_3 = a(g – h)$

$p_4 = d(f – e)$

$p_5 = (a – b)h$

$p_6 = (c – d)(e + g)$

$p_7 = (b – d)(f + h)$

Recurrence for Strassen's Algorithms

- $T(n) = 7\ T(n/2) + cn^2$
- What is the runtime?