CSE 589
Applied Algorithms
Spring 1999

Course Introduction

Depth First Search

Instructors

- Instructor
  - Richard Ladner
  - ladner@cs.washington.edu
  - 206 543-9347
- TA
  - Saurabh Sinha
  - (saurabh@cs.washington.edu)

Resources

- 589 Course Web Page
  - http://www.cs.washington.edu/education/courses/589/CurrentQtr/
- Papers and Sections from Books
- Recommended Algorithms Book
  - Introduction to Algorithms by Cormen, Leiserson, and Rivest

Engagement by Students

- Weekly Assignments
  - Algorithm design and evaluation
- Project with a written report
  - Evaluate several alternative approaches to algorithmically solve a problem
  - Must include readings from literature
  - May include an implementation study
  - May be done in small teams

Final Exam and Grading

- Thursday, June 10th, 6:30 - 8:20 pm
- Percentages
  - Weekly Assignments 30%
  - Project 30%
  - Final 40%

Some Topics

- Network spanning tree (warm up)
- Cache conscious sorting
- Data Compression
- Computational Biology
- Computational Geometry

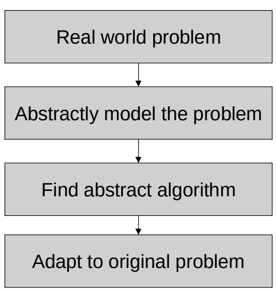
Along the Way

- Analysis of algorithms
- Data structures
- NP-completeness
- Dynamic programming
- Greedy algorithms
- Clustering algorithms
- Branch-and-bound algorithms
- Approximation algorithms
- Classics of algorithms

What We'll Do Today

- Applied Algorithms - By example
- Broadcasting in a network
- Depth First Search
- Breadth First Search
- Minimum Spanning Tree

Applied Algorithm Scenario



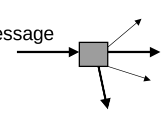
Modeling

- What kind of algorithm is needed
  – Sorting or Searching
  – Graph Problem
  – Linear Programming
  – Dynamic Programming
  – Clustering
  – Algebra
- Can I find an algorithm or do I have to invent one

Broadcasting in a Network

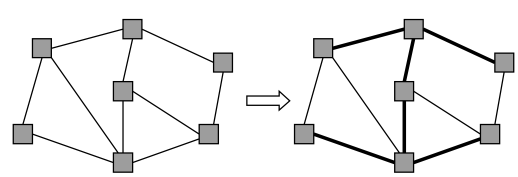
- Network of Routers
  – Organize the routers to efficiently broadcast messages to each other

Incoming message



- Duplicate and send to some neighbors.
- Eventually all routers get the message

Spanning Tree in a Graph

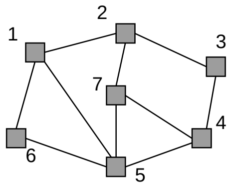


Vertex = router
Edge = link between routers

Spanning tree
- Connects all the vertices
- No cycles

Undirected Graph

- G = (V,E)
  - V is a set of vertices (or nodes)
  - E is a set of unordered pairs of vertices



V = {1,2,3,4,5,6,7}
E = {{1,2},{1,6},{1,5},{2,7},{2,3},
{3,4},{4,7},{4,5},{5,6}}

2 and 3 are adjacent
2 is incident to edge {2,3}

Spanning Tree Problem

- Input: An undirected graph G = (V,E). G is connected.
- Output: T contained in E such that
  - (V,T) is a connected graph
  - (V,T) has no cycles

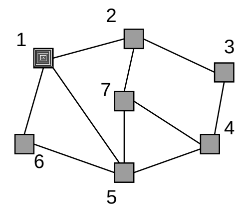
Depth First Search Algorithm

- Recursive marking algorithm
- Initially every vertex is unmarked

```
DFS(i: vertex)
  mark i;
  for each j adjacent to i do
    if j is unmarked then DFS(j)
end{DFS}
```

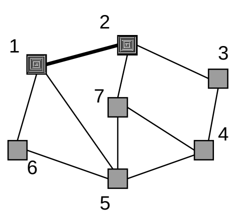

Example of Depth First Search

DFS(1)



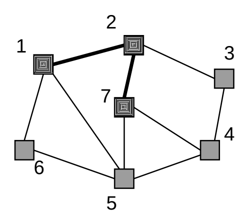
Example Step 2

DFS(1)
DFS(2)

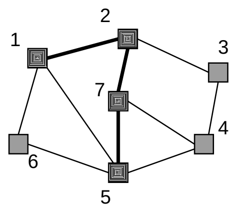


Example Step 3

DFS(1)
DFS(2)
DFS(7)

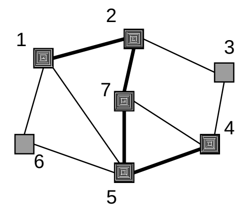


Example Step 4



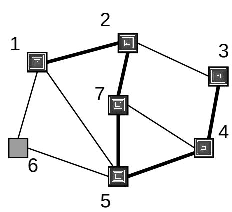
DFS(1)
DFS(2)
DFS(7)
DFS(5)

Example Step 5



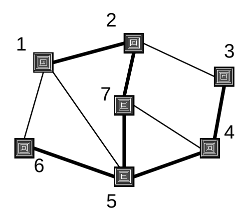
DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)

Example Step 6



DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
DFS(3)

Example Step 7



DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
DFS(3)
DFS(6)

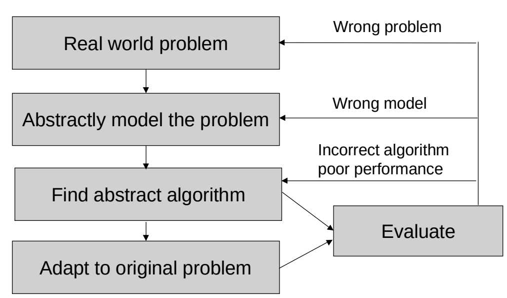
Note that the edges traversed in the depth first search form a spanning tree.

Spanning Tree Algorithm

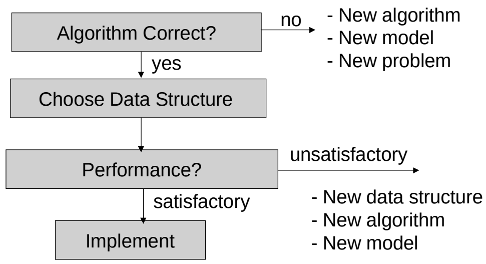
```
ST(i: vertex)
   mark i;
   for each j adjacent to i do
      if j is unmarked then
         Add {i,j} to T;
         ST(j);
end{ST}
```

```
Main
T := empty set;
ST(1);
end{Main}
```


Applied Algorithm Scenario

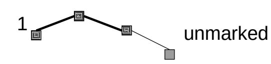


Evaluation Step Expanded



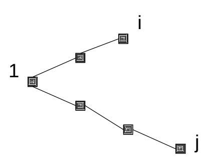
Correctness of ST Algorithm

- There are no cycles in T
  - This is an invariant of the algorithm.
  - Each edge added to T goes from a vertex in T to a vertex not in T.
- If G is connected then eventually every vertex is marked.

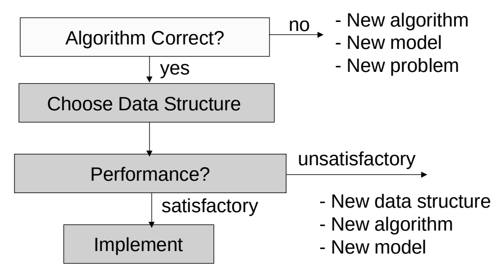


Correctness (cont.)

- If G is connected then so is (V,T)



Data Structure Step



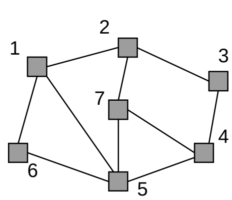
Edge List and Adjacency Lists

- List of edges

1	5	1	2	2	3	5	7	5	5
2	1	6	7	3	4	6	4	7	4

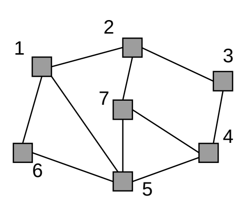
- Adjacency lists

1 → 2 → 5 → 6
2 → 3 → 1 → 7
3 → 2 → 4
4 → 3 → 7 → 5
5 → 6 → 1 → 7 → 4
6 → 1 → 5
7 → 4 → 5 → 2



Adjacency Matrix

	1	2	3	4	5	6	7
1	0	1	0	0	1	1	0
2	1	0	1	0	0	0	1
3	0	1	0	1	0	0	0
4	0	0	1	0	1	0	1
5	1	0	0	1	0	1	1
6	1	0	0	0	1	0	0
7	0	1	0	1	1	0	0



Data Structure Choice

- Edge list
  - Simple but does not support depth first search
- Adjacency lists
  - Good for sparse graphs
  - Supports depth first search
- Adjacency matrix
  - Good for dense graphs
  - Supports depth first search

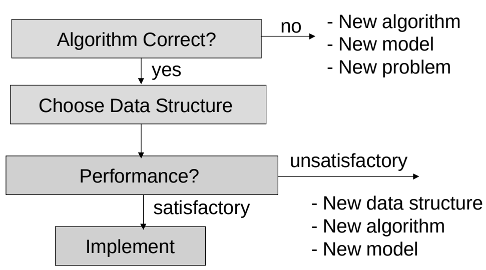
Spanning Tree with Adjacency Lists

```
ST(i: vertex)
   M[i] := 1;
   v := G[i];
   while  not(v = null)
      j := v.vertex;
      if M[j] = 0 then
         Add {i,j} to T;
         ST(j);
      v := v.next;
end{ST}
```

```
Main
   G is array of adjacency lists;
   M[i] := 0 for all i;
   T is empty;
   Spanning_Tree(1);
end{Main}
```

M is the marking array
Node of linked list
vertex next

Performance Step



Performance of ST Algorithm

- n vertices and m edges
- Connected graph
- Storage complexity O(m)
- Time complexity O(m)

Other Uses of Depth First Search

- Popularized by Hopcroft and Tarjan 1973
- Connected components
- Biconnected components
- Strongly connected components in directed graphs
- topological sorting of a acyclic directed graphs