CSE 589
Applied Algorithms
Spring 1999

Minimum Spanning Tree
Disjoint Union / Find

ST using Breadth First Search 1

- Uses a queue to order search

Breadth First Search 2

Queue = 2, 6, 5

Breadth First Search 3

Queue = 6, 5, 7, 3

Breadth First Search 4

Queue = 5, 7, 3

Breadth First Search 5

Queue = 7, 3, 4
Breadth First Search 6

Queue = 3, 4

Breadth First Search 7

Queue = 4

Breadth First Search 8

Queue =

Spanning Tree using Breadth First Search

Initialize T to be empty;
Initialize Q to be empty;
Enqueue(1, Q) and mark 1;
while Q is not empty do
  i := Dequeue(Q);
  for each j adjacent to i do
    if j is not marked then
      add {i, j} to T;
      Enqueue(j, Q) and mark j

Depth First vs Breadth First

• Depth First
  – Stack or recursion
  – Many applications
• Breadth First
  – Queue (recursion no help)
  – Can be used to find shortest paths from the start vertex

Best Spanning Tree

• Each edge has the probability that it won’t fail
• Find the spanning tree that is least likely to fail
Example of a Spanning Tree

Minimum Spanning Tree Problem

- Input: Undirected Graph $G = (V,E)$ and a cost function $C$ from $E$ to the reals. $C(e)$ is the cost of edge $e$.
- Output: A spanning tree $T$ with minimum total cost. That is: $T$ that minimizes $\sum_{e \in T} C(e)$

Reducing Best to Minimum

Let $P(e)$ be the probability that an edge doesn’t fail. Define:

$$C(e) = -\log_{10}(P(e))$$

Minimizing $\sum_{e \in T} C(e)$ is equivalent to maximizing $\prod_{e \in T} P(e)$ because $\prod_{e \in T} P(e) = 10^{\sum_{e \in T} C(e)}$

Example of Reduction

Minimum Spanning Tree

- Boruvka 1926
- Kruskal 1956
- Prim 1957 also by Jarnik 1930
- Karger, Klein, Tarjan 1995
  - Randomized linear time algorithm
  - Probably not practical, but very interesting

MST Optimality Principle

- $G = (V,E)$ with costs $C$. $G$ connected.
- Let $(V,A)$ be a subgraph of $G$ that is contained in a minimum spanning tree. Let $U$ be a set such that no edge in $A$ has one end in $U$ and one end in $V-U$. Let $C\{(u,v)\}$ minimal and $u$ in $U$ and $v$ in $V-U$. Let $A’$ be $A$ with $(u,v)$ added. Then $(V,A’)$ is contained in a minimum spanning tree.
Proof of Optimality Principle 1

$$C(u, v)$$ is minimal

Proof of Optimality Principle 2

$$C(u, v)$$ is minimal

$$C(u, v) \leq C(x, y)$$

Proof of Optimality Principle 3

$$T'$$ is also a minimum spanning tree

$$C(T') = C(T) + C(u, v) - C(x, y)$$

$$C(T') < C(T)$$

Kruskal’s Greedy Algorithm

Sort the edges by increasing cost;
Initialize A to be empty;
For each edge e chosen in increasing order do
if adding e does not form a cycle then
add e to A

Invariant: A is always contained in some minimum spanning tree

Example of Kruskal 1

Example of Kruskal 2
Example of Kruskal 7

Data Structures for Kruskal

- Sorted edge list
  \[\{(7,4), (2,1), (7,5), (5,6), (5,4), (1,6), (2,7), (2,3), (3,4), (1,5)\}\]

- Disjoint Union / Find
  - Union(a, b) - union the disjoint sets named by a and b
  - Find(a) returns the name of the set containing a

Example of DU/F 1

Example of DU/F 2

Example of DU/F 3
### Kruskal’s Algorithm with DU / F

Sort the edges by increasing cost; Initialize A to be empty; for each edge \((i,j)\) chosen in increasing order do
- \(u := \text{Find}(i)\);
- \(v := \text{Find}(j)\);
- if not\((u = v)\) then
  - add \((i,j)\) to \(A\);
  - \(\text{Union}(u,v)\);

### Up Tree for DU/F

**Initial state**

**Intermediate state**

**Final state**

### DU/F Operation

- **Find**(\(i\)) - follow pointer to root and return the root.
- **Union**(\(i,j\)) - assuming \(i\) and \(j\) roots, point \(i\) to \(j\).

**Union(1,7)**

### Weighted Union

- **Weighted Union**
  - Always point the smaller tree to the root of the larger tree

**W-Union(1,7)**

### Path Compression

- On a Find operation point all the nodes on the search path directly to the root.

**Find(6)**

### Elegant Array Implementation

**up**

**weight**
Up Tree Pseudo-Code

PC-Find(i : index)
  r := i;
  while not(up[r] = 0) do
    r := up[r];
    k := up[i];
    while not(k = r) do
      up[i] := r;
      i := k;
      k := up[k]
  return(r)
end{Find}

W-Union(i,j : index)
// i and j are roots
wi := weight[i];
wj := weight[j];
if wi < wj then
  up[i] := j;
  weight[i] := wi + wj;
else
  up[j] := i;
  weight[j] := wi +wj;
end{W-Union}

Disjoint Union / Find Notes

• Weighted union and path compression analyzed by Tarjan in 1975
  – Worst case time complexity for a W-Union
    is O(1) and for a PC-Find is O(log n).
  – Time complexity for m operations on n
    elements is O(m α(m,n)) where α is a very
    slow growing function α(m,n) ≤ 4 for all
    practical m and n. α is called inverse
    Ackermann’s function. Essentially
    constant time per operation!