Spanning Tree Algorithms

Department of Computer Science
University of Maryland, College Park
**Spanning Tree**

- Set of edges connecting all nodes in graph
  - need $N-1$ edges for $N$ nodes
  - no cycles, can be thought of as a tree
- Can build tree during traversal

(a) Graph G

(b) Spanning tree T of graph G
Spanning Tree Construction

- **Recursive algorithm**

  Known = { start }
  explore ( start );

  void explore (Node X) {
    for each successor Y of X
      if (Y is not in Known)
        Parent[Y] = X
        Add Y to Known
        explore(Y)
  }
Spanning Tree Construction

- Iterative algorithm

Known = { start }
Discovered = { start }
while ( Discovered ≠ ∅ ) {
    take node X out of Discovered
    for each successor Y of X
        if (Y is not in Known)
            Parent[Y] = X
            Add Y to Discovered
            Add Y to Known
}
Breadth & Depth First Spanning Trees

Breadth-first

Depth-first
Depth-First Spanning Tree Example

1. Start at node A.
2. Visit node B, then node C.
3. Visit node D, then node E.
4. Visit node F, then node G.
5. Visit node F again, then node E.
6. Visit node D, then node E.
7. Visit node F again, then node E.
8. Visit node G.
9. Visit node C.
10. Visit node B.
11. Visit node C.
12. Visit node G.
13. Visit node C.
14. Visit node D.
15. Visit node E.
16. Visit node F.
17. Visit node G.
Breadth-First Spanning Tree Example
Spanning Tree Construction

- Many spanning trees possible
  - Different breadth-first traversals
    - Nodes same distance visited in different order
  - Different depth-first traversals
    - Neighbors of node visited in different order
- Different traversals yield different spanning trees
Minimum Spanning Tree (MST)

- Spanning tree with minimum total edge weight

(a) Graph G

(b) A spanning tree of cost $C = 43$

(c) A minimum spanning tree of cost $C = 28$
Minimum Spanning Tree (MST)

• Possible to have multiple MSTs
  • Different spanning trees with same weight

• Example applications
  • Minimize length of telephone lines for neighborhood
  • Minimize distance of airplane routes serving cities
Algorithms for Finding MST

• Three well known algorithms
  1. Borůvka’s algorithm [1926]
     • For constructing efficient electricity network
     • Rediscovered by Sollin in 1960s
  2. Prim’s algorithm [1957]
     • First discovered by Vojtěch Jarník in 1930
     • Similar to Djikstra’s algorithm
  3. Kruskal’s algorithm [1956]
     • By Prof. Clyde Kruskal’s uncle
Algorithms for Finding MST

1. **Borůvka’s algorithm**
   - Add vertices to MST in parallel

2. **Prim’s algorithm**
   - Add vertices to MST
     - One at a time
     - Closest vertex first

3. **Kruskal’s algorithm**
   - Add edges to MST
     - One at a time
     - Lightest edge first
Shortest Path – Dijkstra’s Algorithm

S = ∅
P[ ] = none for all nodes
C[start] = 0, C[ ] = ∞ for all other nodes

while ( not all nodes in S )

  find node K not in S with smallest C[K]
  add K to S

  for each node J not in S adjacent to K
    if ( C[K] + cost of (K,J) < C[J] )
      C[J] = C[K] + cost of (K,J)
P[J] = K

Optimal solution computed with greedy algorithm
MST – Prim’s Algorithm

S = ∅
P[ ] = none for all nodes
C[start] = 0, C[ ] = ∞ for all other nodes

while ( not all nodes in S )
    find node K not in S with smallest C[K]
    add K to S
    for each node J not in S adjacent to K
        if ( /* C[K] + */ cost of (K,J) < C[J] )
            C[J] = /* C[K] + */ cost of (K,J)
            P[J] = K

Keeps track of vertex w/ minimal distance to current tree
Optimal solution computed with greedy algorithm
MST – Kruskal’s Algorithm

sort edges by weight (from least to most)
tree = ∅

for each edge (X,Y) in order
  if it does not create a cycle
    add (X,Y) to tree
  stop when tree has N–1 edges

Keeps track of
  • lightest edge remaining
  • whether adding edge to MST creates cycle

Optimal solution computed with greedy algorithm
MST – Kruskal’s Algorithm Example
MST – Kruskal’s Algorithm

• When does adding (X,Y) to tree create cycle?
• Two approaches to finding cycles
  1. Traversal
  2. Connected sub-graph
MST – Kruskal’s Algorithm

• **Traversal approach**
  - Traverse tree starting at X
  - If we can reach Y, adding (X,Y) would create cycle

• **Example**
  - Question
    - Add (X,Y) to MST?
  - Answer
    - No, since can already reach Y from X by traversing MST
MST – Kruskal’s Algorithm

• Connected sub-graph approach
  • Maintain set of nodes for each connected subgraph
  • Initialize one connected sub-graph for each node
  • If X, Y in same set, adding (X,Y) would create cycle
  • Otherwise
    1. Add edge (X,Y) to spanning tree
    2. Merge sets containing X, Y

• To test set membership
  • Use Union-Find algorithm
MST – Connected Subgraph Example

Original graph

- A: 5
- B: 9
- C: 13
- D: 15

Ordered set of edges

- <A, B>: 5
- <A, C>: 9
- <B, C>: 13
- <C, D>: 15
- <B, D>: 17

Sets

- {A} {B} {C} {D}

Edge being considered for addition

- <A, B>: Include, since it connects two nodes in distinct sets
- <A, C>: Include, since it connects two nodes in distinct sets
MST – Connected Subgraph Example

Original graph

Ordered set of edges

\(<A, B> 5\)
\(<A, C> 9\)
\(<B, C> 13\)
\(<C, D> 15\)
\(<B, D> 17\)

MST

Sets

3. \(\{A, B, C\} \{D\}\)

Edge being considered for addition

\(<B, C>\) Reject, since it connects nodes in the same set and would create a cycle

4. \(\{A, B, C\} \{D\}\)

\(<C, D>\) Include, since it connects two nodes in distinct sets

Finished
Union-Find Algorithm

- **Union-Find**
  - Algorithm & data structure
  - Very efficient for testing membership in disjoint sets
- **Problem description**
  - Start with n nodes, each in different subgraph
  - Support two operations
    - Find $\rightarrow$ are nodes $x$ & $y$ in same subgraph?
    - Union $\rightarrow$ merge subgraphs containing $x$ & $y$
Union-Find Algorithm

- **Basic approach**
  - Each node has a parent pointer
  - Find $\rightarrow$ follow parent pointer(s) to root of tree
  - Union $\rightarrow$ point root of 1\textsuperscript{st} tree to root of 2\textsuperscript{nd} tree

- **Example**
  - Union(a, b); union(c, d); union(a, c)
**Union-Find Algorithm**

- **Path compression**
  - Speeds up future `Find()` operations
    1. Follow parent pointer(s) to root of tree
    2. Update all nodes along path to point to root

- **Example**
  - `Find(d)`

---

So how fast is Union-Find?
Ackermann’s Function

• Function

```c
int A(x, y) {
    if (x == 0)
        return y + 1;
    if (y == 0)
        return A(x - 1, 1);
    return A(x - 1, A(x, y - 1));
}
```

• $A(\ )$ grows fast

- $A(2,2) = 7$
- $A(3,3) = 61$
- $A(4,2) = 2^{65536} - 3$
- $A(4,3) = 2^{2^{65536}} - 3$
- $A(4,4) = 2^{2^{2^{65536}}} - 3$
Inverse Ackermann’s Function

• Definition
  • $\alpha(n)$ is the inverse Ackermann’s function
  • $\alpha(n) = \text{the smallest } k \text{ such that } A(k,k) \geq n$

• Fun fact
  • $\alpha(\text{number of atoms in universe}) = 4$

• Union-find
  • A sequence of $n$ operations requires $O(n \alpha(n))$ time
  • Practically speaking, indistinguishable from $O(n)$