CS 491 CAP
Basic Graph Algorithm

Zhengkai Wu

University of Illinois at Urbana-Champaign

Oct 06, 2017

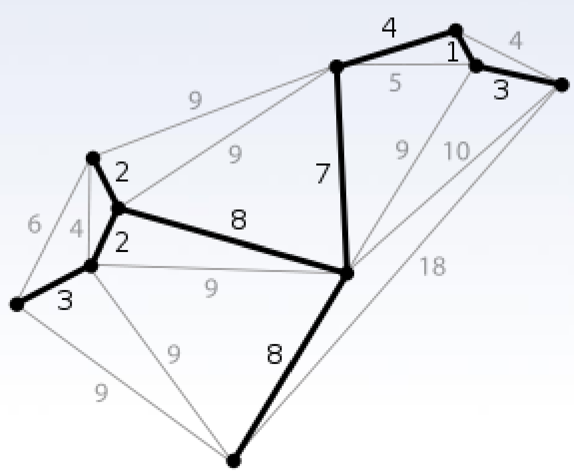
Today

- ◇ Minimum Spanning Tree (MST)
  - Kruskal's Algorithm
- ◇ Shortest Path
  - Single Source: Dijkstra Algorithm / Bellman-Ford Algorithm
  - All Sources: Folyd-Warshall Algorithm

illinois.edu

Minimum Spanning Tree

- ◇ Given a weighted, undirected graph: G = (V,E)
- ◇ Find a subset of E such that
  - Connecting all vertices
  - Without any cycles
  - Minimum possible total edge weight



Cycle Property

- For any cycle C in the graph, if the weight w of an edge e in C is larger than any other edges in C, the e cannot belong to any MST.

- Proof sketch:
- Consider there is such an edge e in an MST T1. e breaks T1 into two subgraphs. We can find an edge in C that connects these two subgraphs. Thus replacing e with that edge results in a tree with less total weight.

Cut Property

- Cut: $C=(S,T)$ while $S \cap T=\emptyset$ and $S \cup T=V$
- Cut-set: $\{(u,v) \mid u \in S \text{ and } v \in T\}$

- For any cut C, if e is the unique minimum weight edge in the cut-set of C, then e belongs to all MSTs.

- Proof Sketch:
- If e not in MST T1, adding e will form a cycle, replacing the other edge of the cycle in the cut-set will result in a better tree.

illinois.edu

Min-cost Edge Property

- If the minimum cost edge e of a graph is unique, then this edge is included in any MST.

- Proof Sketch:
- If e not in MST T1, adding e will form a cycle, replacing the any other edge in the cycle will result in a better tree.

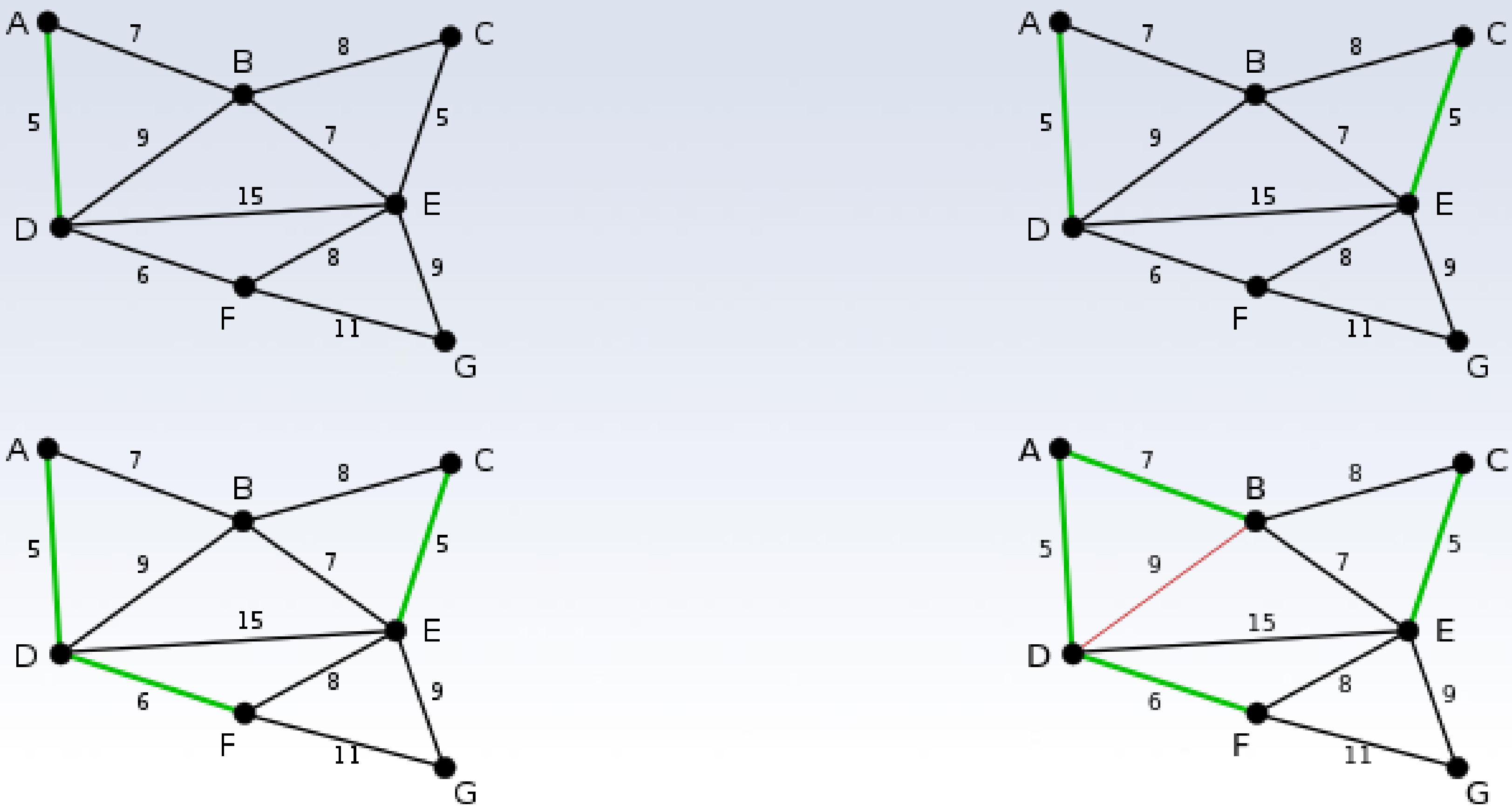
illinois.edu

Kruskal Algorithm

- Sort the edges in increasing order of weight
- Iterate through each edge e in order, until size of the MST = $|V| - 1$
  - If e connects two different connected components, then add e to the MST and merge the two connected components (using disjoint set data structure)
- Otherwise, ignore e and move on

illinois.edu

Example



Disjoint-set Data Structure

- Need to support three operations:
  - MAKE-SET(v): Initialization, generate a set with one element v.
  - FIND-SET(v): Find the representative of the set containing v.
  - UNION(u,v): Union the two sets containing u and v.
- Using a rooted tree to represent the set, storing the father in the array f[v].
  - If f[v] = v, then v is the representative.
  - MAKE-SET(v): let f[v] := v.
  - FIND-SET(v): Recursively do FIND-SET(f[v]) until f[v] = v.
  - UNION(u,v): let f[FIND-SET(v)] := FIND-SET(u).
- Too slow. Complexity can reach O(n^2) in n steps.

Disjoint-set Data Structure

- Two optimization
  - Union by rank: Union the tree with smaller depth to the larger one.
  - Path Compression: Let every f[v] to its root when doing a FIND-SET

```
function MakeSet(x)
  if x is not already present:
        add x to the disjoint-
set tree
        f[x] := x
        rank[x] := 0

function Find(x)
  if f[x] != x f[x]:= Find(f[x])
return f[x]
```

```
function Union(x, y)
  xRoot := Find(x)
  yRoot := Find(y)
  if xRoot == yRoot
  if rank[xRoot] < rank[yRoot]
        f[xRoot] := yRoot
  else if rank[xRoot] > rank[yRoot]
        f[yRoot] := xRoot
      else
        f[yRoot] := xRoot
        rank[xRoot] := rank[xRoot] + 1
```

Complexity

- By using the disjoint-set data structure.
- The time complexity of Kruskal algorithm become O(|E|log|E| (sorting time) + |E| (amortized complexity of disjoint-set))

illinois.edu

UVA 10842

- Given a graph, find a spanning tree with the minimum edge in the tree maximized.

Note that the maximum spanning tree is the tree we want.
Proof: Consider the procedure of Kruskal algorithm.

UVA 10600

- Given a graph. Print the value of MST and the second-minimum spanning tree. (n <= 200)

The second minimum spanning tree must be the MST replacing one edge.

Poj 2728

- N villages need to be connected, given their coordinates (xi,yi) and height hi.
- The distance is the euclidian distance between two villages and the cost is the difference of their height.
- Find the minimum ratio of the sum of distance divided by the sum of cost.

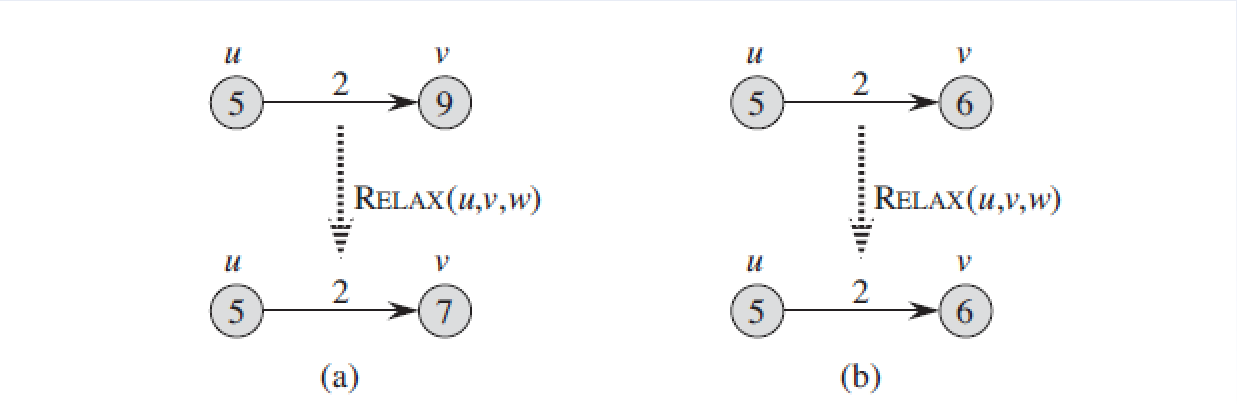
Binary search the answer, suppose the answer to be k. We need to determine whether k is satisfiable.
This is equivalent to finding a spanning tree in which sum(dist_i)/sum(cost_i) <= k.
Which is sum(dist_i-k*cost_i) <= 0. Which is equivalent to find the maximum spanning tree of edge weight dist_i-k*cost_i

Shortest Path

- Given a weighted directed/undirected graph: G = (V,E).
- Finding a path between node u and v that minimize the weight of the path.
- Different type of shortest path.
  - Allow negative weight?
  - If negative weight, is there a loop with negative weight?
  - Dense or Sparse graph?

Relaxation

- Relax Operation:
  - $D[v] \leftarrow \min(D[v], D[u] + w)$
- Update the value of the target node using the edge of weight w.



Dijkstra Algorithm

- ◇Single Source Shortest Path (SSSP) with no negative weight.
- ◇Time complexity:O(|E|log|V|) using priority queue.

Pseudocode:

- Maintain a set S that stores nodes for which the shortest path has already been determined
- Maintain a vector $D[v]$ to store the shortest distance estimate from s
- Initially, $S \leftarrow s$, $D[v] \leftarrow weight(s, v)$
  - If $e(s, v) \notin E$, $D[v] \leftarrow \infty$
- Repeat until $S = V$
  - Find $v \notin S$ with the smallest $D[v]$, and add it to S – Use priority queue for better performance
  - For each edge $v \rightarrow u$ with weight w,
    - Relax(u,v,w)

Bellman-Ford Algorithm

◊ SSSP allowing negative weight.
◊ Can detect negative loop.
◊ Time Complexity: O(|E|*|V|)

Pseudocode:
◊ Initialize $D[s] \leftarrow 0$ and $D[v] \leftarrow \infty$ for all $v \neq s$
◊ For $k = 1 \,..\, V - 1$:
- For each edge $u \rightarrow v$ of weight w:
  - Relax(u,v,w)

◊ For each edge $u \rightarrow v$ of weight w:
- If $D[v] > D[u] + w$:
  - The graph contains a negative weight cycle

illinois.edu

SPFA

- ◇A special optimization of Bellman-Ford Algorithm.
- ◇Use a queue to only updates the node that get affected by the relaxation
- ◇Have a good performance on a lot of graphs. But the worst can still be O(|V||E|)

Pseudocode:

◊ Initialize $D[s] \leftarrow 0$ and $D[v] \leftarrow \infty$ for all $v \neq s$ and queue q to contain only s.

◊ While not q.empty() do:

- ▪ u = q.pop();
- ▪ For each edge starting from u: $u \rightarrow v$ of weight w:
  - • Relax(u,v,w)
  - • if $D[v]$ is decreased and v is not in q: q.push(v)

Floyd-Warshall Algorithm

◇ Computes All Pairs Shortest Path (APSP)
◇ Time Complexity: O(|V|^3)

Pseudocode:

Initialize matrix D with weights from the given graph (∞ if there is no edge)

for k = 1 . . V:
    for i = 1 . . V:
        for j = 1 . . V:
            $D[i][j] \leftarrow \min(D[i][j], D[i][k] + D[k][j])$

illinois.edu

Floyd-Warshall Algorithm

- Why does it work? Dynamic Programming
  - $D[k][i][j]$: weight of shortest path from i to j using vertices numbered $\leq k$ as the intermediate nodes
- The recurrence relation then is
  - $D[k][i][j] = \min(D[k-1][i][j], D[k-1][i][k] + D[k-1][k][j])$
    - This holds because we have two possible choices: either use k as the intermediate node, or don't
- Turns out the first dimension is not necessary, can just overwrite
- Be careful that k must be the outmost loop variable.

POJ 3463

- Calculate the number of different shortest paths.

After calculate the shortest path d[i]. Do another dynamic programming. Count[i] += Count[j] (if d[i] = d[j] + w)

Poj 1734

- Find the minimal simple loop in the given weighted graph. (N<=100, M<=10000)

Using Floyd Algorithm, in the outmost loop, when we haven’t updating the d[i][j] using node k.
The current shortest path d[i][j] cannot go through k. So we can calculate the d[i][j]+g[j][k]+g[k][i] to be the candidate minimal loop.
Thus we can calculate all possible loop candidates with the Floyd Algorithm going to get the minimal one.