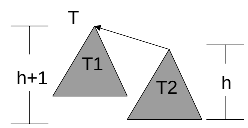
CSE 589
Applied Algorithms
Spring 1999

Prim's Algorithm for MST
Load Balance Spanning Tree
Hamiltonian Path

Performance of W-Union / PC-Find

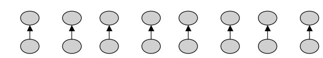
- The time complexity of PC-Find is O(log n).
- An up tree formed by W-Union of height h has at least 2^h nodes. Inductive Proof.



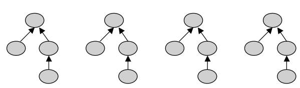
Weight(T2) $\geq 2^h$ (ind. hyp.)
Weight(T1) $\geq$ Weight(T2) $\geq 2^h$
Weight(T) $\geq 2^h + 2^h = 2^{h+1}$

Worst Case for PC-Find

n/2 Weighted Unions

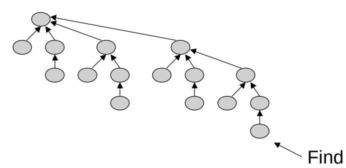


n/4 Weighted Unions



Example of Worst Cast (cont')

After n -1 = n/2 + n/4 + …+ 1 Weighted Unions



If there are $n = 2^k$ nodes then there are k pointers on the longest path to root.

Amortized Complexity

- For disjoint union / find with weighted union and path compression.
  - average time per operation is essentially a constant.
  - worst case time for a PC-Find is O(log n).
- An individual operation can be costly, but over time the average cost per operation is not.

Recall Kruskal

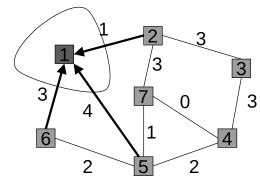
```
Sort the edges by increasing cost;
Initialize A to be empty;
for each edge {i,j} chosen in increasing order do
   u := PC-Find(i);
   v := PC-Find(j);
   if not(u = v) then
      add {i,j} to A;
      W-Union(u,v);
```

Evaluation of Kruskal

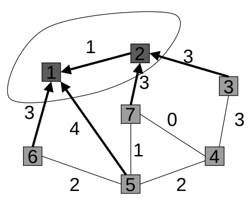
- Let G have n vertices and m edges.
- Sort the edges - O(m log m).
- Traverse the sorted edge list doing PC-Finds and W-Unions - O(m α(m,n)).
- Total time is O(m log m).

Prim's Algorithm

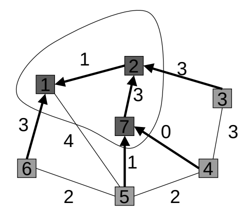
- We maintain a single tree.
- For each vertex not in the tree maintain the smallest edge to a vertex in the tree.



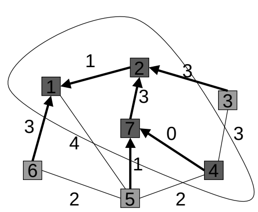
Prim's Algorithm 2



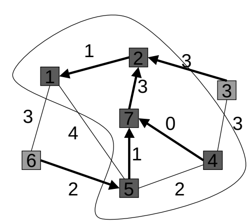
Prim's Algorithm 3



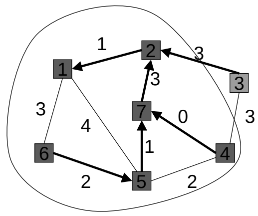
Prim's Algorithm 4



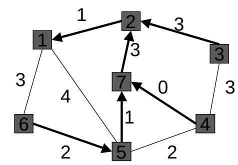
Prim's Algorithm 5



Prim’s Algorithm 6

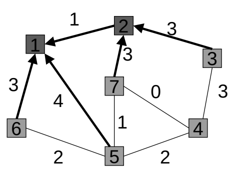


Prim’s Algorithm 7



Data Structures for Prim

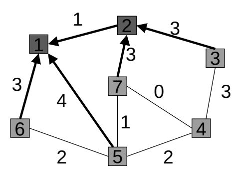
- Adjacency Lists - we need to look at all the edges from a newly added vertex.
- Array for the best edges in or to the tree.



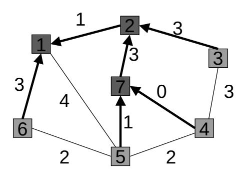
	1	2	3	4	5	6	7
to		1	2		1	1	2
cost		1	3		4	3	3

Data Structures for Prim

- Priority queue for all edges to the tree (blue edges).
  - Insert, delete-min, delete (e.g. binary heap).



	1	2	3	4	5	6	7
to		1	2		1	1	2
cost		1	3		4	3	3



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

Evaluation of Prim

- n vertices and m edges.
- Priority queue O(log n) per operation.
- O(m) priority queue operations.
  - An edge is visited when a vertex incident to it joins the tree.
- Time complexity is O(m log n).
- Storage complexity is O(m).

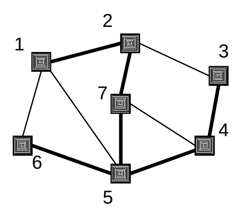
Kruskal vs Prim

- Kruskal
  - Simple
  - Good with sparse graphs - O(m log m)
- Prim
  - More complicated
  - Perhaps better with dense graphs - O(m log n)

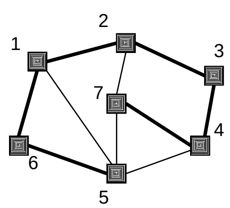
Load Balanced Spanning Tree (LBST)

- Input: An undirected graph G = (V,E) and number k.
- Output: Determine if there is a spanning tree (V,T) of G with the property that for each vertex v there are at most k edges in T incident to v. If there is such a spanning tree report it. We call such a tree a spanning tree of degree k.

Spanning Tree of Degree 3



Spanning Tree of Degree 2



Optimization Version of LBST

- Input: An undirected graph G = (V,E).
- Output: A number k and a spanning tree (V,T) of degree k. Furthermore, there is no spanning tree of degree < k.

Equivalence of two versions

- Reporting version can be easily reduced to the optimization version.
- Optimization version can be reduced to the reporting version by searching. Assume a function LBST(G,k) that returns a spanning tree of degree k if there is one, else returns null.

```
k := 2;
repeat
   T:= LBST(G,k);
   if T = null then k := k +1
until not(T = null)
```


LBST Decision Problem

- Input: An undirected graph G = (V,E) and number k.
- Output: Determine if G has a spanning tree of degree k.

- We expect a yes/no answer only without reporting a solution if the answer is yes.

Classes of Problems

- Decision Problem: just yes or no. Is there a solution or not.
- Reporting Problem: yes or no, and if yes then report a solution.
- Optimization Problem: find a best solution for some notion of best.

Hamiltonian Path Decision Problem

- Input: Undirected Graph G =(V,E).
- Output: Determine if there is a path in G that visits each node exactly once.

- Decision problem: Yes or No answer.
- This is a famous NP-complete problem.
- NP-complete problems do not appear to have polynomial time algorithms.
- NP-complete problems are hard to solve!

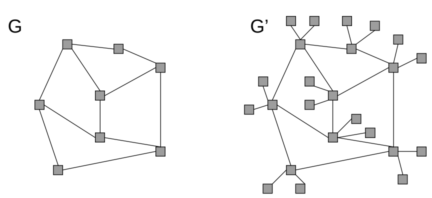
Hamiltonian Path Reducible to Spanning Tree of Degree 2

- If there an algorithm to quickly determine if a graph has a spanning tree of degree then there is an algorithm to quickly solve the Hamiltonian path problem.
  - A spanning tree of degree 2 is a Hamiltonian path!
  - These problems are essentially the same problem.

Hamiltonian Path is Reducible Spanning Tree of Degree k for any k

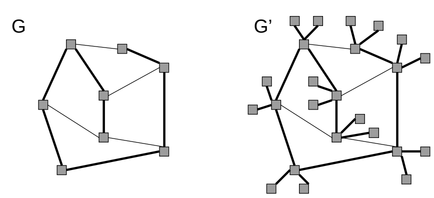
- Let G = (V,E) be an undirected graph. We can construct in polynomial time G’ = (V’,E’) with the property that G has a Hamiltonian path if and only if G’ has a spanning tree of degree k.
- Thus, if there is a polynomial time algorithm for the spanning tree problem then there is also also for the Hamiltonian path problem.
- But there is likely no such algorithm!

HP reducible to LBST of Degree 4



G has a Hamiltonian Path if and only if G’ has a spanning tree of degree 4.

HP reducible to LBST of Degree 4 (2)



G has a Hamiltonian Path if and only if G’ has spanning tree of degree 4.

HP Reducible to LBST of Degree 4 (3)

$$G = (V, E) \qquad V = \{u_1, u_2, ..., u_n\}$$

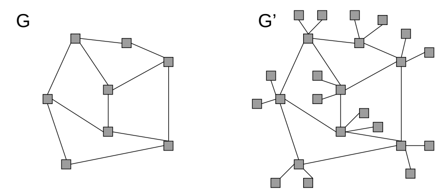
$$V' = V \cup \{v_{i,j} : 1 \le i \le n, 1 \le j \le k-2\}$$

$$E' = E \cup \{\{u_i, v_{i,j}\} : 1 \le i \le n, 1 \le j \le k-2\}$$

$$G' = (V', E')$$

G has a Hamiltonian Path if and only if G' a spanning tree of degree k.

HP reducible to LBST of Degree 4 (4)



Key fact: Any spanning tree in G' must contain all the new edges.