Lecture15_Graphs & 16_MST

Saturday, October 17, 2020 5:00 PM

Representation of Graph
- Adjacency-matrix
- Adjacency-list

MST
- Greedy
- Prim
- Kruskal

Graph

Many problem in graph theory are solved with dynamic program and greedy.

Graph (review)

Definition. A directed graph (digraph) G = (V, E) is an ordered pair consisting of
- a set V of vertices (singular: vertex),
- a set $E \subseteq V \times V$ of edges.

In an undirected graph G = (V, E), the edge set E consists of unordered pairs of vertices.

In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| >= |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$. (Review CLRS, Appendix B.)

Cartesian product of two sets that used to creat order pairs.
A X A, A X B.
- any subset over Cartesian product gives you something we call relations
  Cartesian product: A X B
  Relation: $S \subseteq A \times B$
- each connection each ordered pair is an edge and then that gives us the graphs

Edge set is simply a sub set of the Cartesian product of the vertex sets among itself.

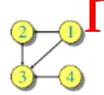
Graph representation:

Adjacency-matrix representation

Adjacency-matrix representation

The adjacency matrix of a graph G = (V, E), where V = {1, 2, …, n}, is the matrix A[1 . . n, 1 . . n] given by

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$



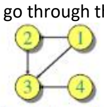
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ storage $\Rightarrow$ *dense* representation.

Adjacency-list representation

Adjacency-list representation

An adjacency list of a vertex $v \in V$ is the list Adj[v] of vertices adjacent to v.
- Need to go through the list to check if the member exist in the list.



$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
For digraphs, $|Adj[v]| = out\text{-}degree(v)$.

Handshaking Lemma: $\sum_{v \in V} degree(v) = 2|E|$ for undirected graphs $\Rightarrow$ adjacency lists use $\Theta(V + E)$ storage — a *sparse* representation.
- if you add up all the degrees for all the vertices then what you should be getting is precisely twice the size of the headset
- Running time: summation of degrees
- Prove: by observation. Be add one edge twice: one time for each direction.

➢ Adjacency-list saves space!

Party Hand shaking problem
For each handshake, two people will report that hand shake

Minimum spanning trees

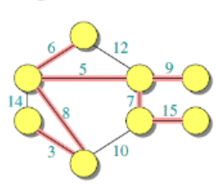
Minimum spanning trees

Input: A connected, undirected graph G = (V, E) with weight function w : E→**R**.
- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)

Output: A spanning tree T — a tree that connects all vertices — of minimum weight:

$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

Example of MST

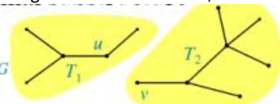


- The outputted subgraph mush be a tree

- The outputted subgraph must be a tree
- Why need n - 1 edges? If more than n - 1, we must have a cycle.

MST T:

(Other edges of G are not shown.)



Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2.

Theorem. The subtree T_1 is an MST of $G_1 = (V_1, E_1)$, the subgraph of G *induced* by the vertices of T_1:

V_1 = vertices of T_1,
$E_1 = \{ (x, y) \in E : x, y \in V_1 \}$.

Similarly for T_2.

Claim:

- T1 must be a MST in the graph G1
- T2 must be a MST in the graph G2
- G = G1 U G2? FALSE
- We are creating subproblem G1 & G2, but notice that there is some potential edges that connecting G1 and G2

Prove: copy-and-phase

Proof. Cut and paste:

$w(T) = w(u, v) + w(T_1) + w(T_2)$.

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. □

Do we also have overlapping subproblems?
- Yes.

Great, then dynamic programming may work!
- Yes, but MST exhibits another powerful property which leads to an even more efficient algorithm.

Greedy Algorithm

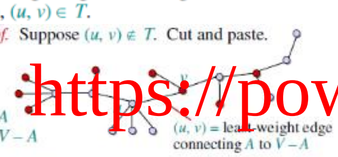
Greedy for MST

Greedy-choice property: A locally optimal choice is globally optimal.

Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.

Proof. Suppose $(u, v) \notin T$. Cut and paste.

T':



○ $\in A$
● $\in V - A$

Consider the unique simple path from u to v in T.

Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.

A lighter-weight spanning tree than T results. □

- Purple edges: A; red edges: V - A. Or vice versa
- Proof by contradiction
  1. Suppose the lest weight edge (that goes from purple to red) is **not** part of the MST
  2. Let T be a MST, and u, v has the smallest edge that not part of the MST (dash line). Suggesting these is must be an unique single path that goes from u to v.
  3. Contradiction

Note that we have assumption the "all weight of edges are distinct". To expand the theorem to a graph containing equally weighted edges, we change "this edge must be part of the optimal solution" to "it must be part of an optimal solution".

Prim's algorithm

Prim's Algorithm

IDEA: Maintain $V - A$ as a priority queue Q. Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in A.

Θ(v) total
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$

|V| times

degree(u) times

while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ▷ DECREASE-KEY
          $\pi[v] \leftarrow u$

update neighbor

At the end, $\{(v, \pi[v])\}$ forms the MST.

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

Time = $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$

- The extract min gives the least cost edge from red vertices to purple vertices
- If the edge weights are not distinct, it can be still unique, but not necessarily unique.

Analysis of Prim

Analysis of Prim

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case

Kruskal Algo

Kruskal's Algorithms

- Uses the disjoint-set data structure
- Running time = O(E lg V).
- Union-Find data structure

union – Find

– MAKE-SET(u) $\{\ldots, \{u\}, \ldots\}$

– FIND(u) $\{\ldots, \{u, v, x\}, \ldots\}$

– UNION(u, v) $\{\ldots, \{u, a\}, \{v, b\}, \ldots\}$

$\Downarrow$

$\{\ldots, \{u, a, v, b\}, \ldots\}$

Kruskal's ALG

MST = { }

For all v ∈ V MAKE-SET(v)

Sort all edges in increasing order of weight

For all (u, v) ∈ E

if FIND(u) ≠ FIND(v)

Add (u, v) to MST

UNION(u, v)

return MST

Recent Algo

Best to date:

- Karger, Klein, and Tarjan [1993].
- Randomized algorithm.
- O(V + E) expected time.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder