Trees, II

李昂生

Discrete Mathematics
U CAS
July, 2018

Outline

1. Definitions
2. Counting the trees
3. Spanning trees in a graph
4. Optimisation of spanning trees

General view

- Spanning trees
- Trees in graphs
- Optimisation of spanning trees

Spanning Tree of a Graph

Definition 1

Let G be a simple graph. A *spanning tree* of G is a subgraph of G that is a tree containing all the vertices of G.

Theorem 2

A simple graph is connected if and only if it has a spanning tree.

Proof.

Breaking a cycle of G by deleting exactly one edge one by one, until there is no cycle in the resulting graph. □

Internet Protocol

To send data from a source computer to multiple receiving computers by following the routes of a spanning tree of the computer network. This ensures that a dataset can be quickly sent to all the receivers.

Depth-First Search for Finding a Spanning Tree

Let $G = (V, E)$ be a connected graph. We construct a spanning tree T of G by adding one new vertex and one new edge each time.
Let $T = (V_T, E_T)$ be the tree constructed.
The algorithm proceeds as follows:

1. Let v_1 be a vertex and add v_1 to V_T, and set $E_T = \emptyset$. Suppose that $v_1, v_2, \cdots, v_i$ are all added to V_T with the ordering as they are listed above.
2. Let j be the largest k such that there is a vertex $u \in V \setminus V_T$ with which there is an edge between u and v_k. Then.
    - let v_{i+1} be such an u,
    - add v_{i+1} to V_T, and
    - enumerate edge (v_{i+1}, v_j) into E_T.

Proof

Clearly, if G is connected, then T is a spanning tree of G.

The time complexity:

$$O(m),$$

where m is the number of edges in G.

Breadth-First Search for Finding a Spanning Tree

Let $G = (V, E)$ be a connected graph. We construct a spanning tree T of G as follows:

1. Let v_1 be a vertex, add v_1 to V_T, set $E_T = \emptyset$.
2. If $V_T = V$, then terminate. Otherwise, then go on to the next stage.
3. For every $x \in V_T$ and every $y \in V \setminus V_T$, if there is an edge xy in E, then
   - enumerate y into V_T, and
   - enumerate edge xy into E_T.

Proof

Clearly, T is a spanning tree of G. The time complexity of the algorithm is

$$O(m),$$

where m is the number of edges in G.

Depth-First Search in Directed Graphs

Let G be a directed graph.
The depth-first search algorithm will find a spanning forest for G.

Application: Web spiders.

Enumeration of Trees

There are $2^{\binom{n}{2}}$ many simple graphs with set $[n] = \{1, 2, \cdots, n\}$.
Question: How many of these are trees?

Prüfer Code

Let $S = [n]$. We use T_n to denote the set of trees with vertex set S.
We will show that there is a one-to-one map between T_n and S^{n-2}.
Therefore,

$$|T_n| = n^{n-2},$$

this is the Cayley's formula.

The Algorithm for Finding Prüfer Code

Input: A tree T with vertex set $S \subseteq \mathbb{N}$.
Output: A list $f(T) \in S^{n-2}$.
The Prüfer code is found by the following steps:

1. Set $H = T$.
2. Defining a_i.
    - Let b be the least leaf in H,
    - Let a_i be the neighbour of b in H
    - Delete vertex b and edge ba_i from H
3. Go back to step 2.

The Prüfer code of T is:

$$f(T) = (a_1, a_2, \cdots, a_{n-2}).$$

The Algorithm for Finding the Tree for a Prüfer Code

Suppose that $S \subseteq \mathbb{N}$ is fixed. We can reconstruct the tree T from the Prüfer code $a = (a_1, a_2, \cdots, a_{n-2})$.
Idea: To retrieve all the edges.
At each step, we create one edge and mark one vertex.

1. Let $X = S$ and $Y = \{a_1, a_2, \cdots, a_{n-2}\}$.
2. For $1 \leq i \leq n-2$, in increasing order, consider a_i.
   - let x be the least $z \in X \setminus Y$,
   - create an edge xa_i,
   - extract x from X,
   - extract a_i from Y.
3. Finally, for the last two vertex a, b in X, create an edge ab. This builds a tree T such that the Prüfer code of T is $a = (a_1, a_2, \cdots, a_{n-2})$.

Cayley's Formula

Theorem 3

(Cayley, 1889) For a set $S \subseteq \mathbb{N}$ of size n, there are n^{n-2} trees with vertex set S.

For $n = 1$. Trivial.
Assume $n \geq 2$.
We show that the algorithm defines a bijection from the trees to S^{n-2}.
We prove it by induction on n.
For $n = 2$.
There is one tree with two vertices. The Prüfer code is empty, which is the element in S^0.
For $n > 2$.
By the algorithm, for every tree T, $f(T) \in S^{n-2}$.

Proof

Precisely,

- Every nonleaf vertex in T is in $f(T)$
- Every leaf vertex in T is not in $f(T)$
- $f(T)$ is the set of all the nonleaf vertices of T
- The first leaf deleted is the least element of S not in $f(T)$, and it is the neighbor of a_1, the first element of $f(T)$.

Proof - continued

Given $a \in S^{n-2}$, we show that there is a unique T such that $f(T) = a$.
We have shown that for every T, if $f(T) = a$, then for

$$x = \min\{y \in S \setminus \{a_1, \cdots, a_{n-2}\}\},$$

x is the least leaf of T and xa_1 is an edge of T.
Let $S' = S - x$ and $T' = T - xa_1$, then

$$f(T') = (a_2, \cdots, a_{n-2}).$$

By the inductive hypothesis, there is a unique T' such that $f(T') = (a_2, \cdots, a_{n-2})$.
Since T is formed by adding vertex x and edge xa_1 in T'. T is unique.
Therefore, there is a unique T such that $f(T) = a$.

Trees with Specified Degree Distribution

Theorem 4

Given positive integers $d_1, d_2, \cdots, d_n$ such that $\sum_{i=1}^{n} d_i = 2(n-1)$, there are exactly

$$\frac{(n-2)!}{\prod_{i=1}^{n}(d_i-1)}$$

trees with vertex set $[n]$ such that vertex i has degree d_i.
By the definition of $f(T)$, for any i, if i appears in $f(T)$, then i is not a leaf of T, and i appears for $d_i - 1$ times.
Therefore, the number of the trees is the number of the lists of length $n-2$ such that for every i, there are $d_i - 1$ copies if i in the list.
For every i and j with $j \le d_i - 1$, we use (i, j) to denote the jth copy of i. This ensues that there are $(n-2)!$ lists.
For each i, the $d_i - 1$ copies of i are indistinguishable for trees.

n

Matrix Tree Theorem

Let $G = (V, E)$ be a graph. We use $\tau(G)$ to denote the number of spanning trees of G.

Theorem 5

(Matrix Tree Theorem) Given a loopless graph G with vertex set $v_1, v_2, \cdots, v_n$, let a_{ij} be the number of edges with endpoints v_i and v_j.
Define $Q = (q_{ij})$ as follows

$$q_{ij} = \begin{cases} -a_{ij}, & \text{if } i \neq j \\ d(v_i), & \text{otherwise.} \end{cases} \tag{1}$$

Let Q^* be the matrix obtained from Q by deleting row s and column t. Then

$$\tau(G) = (-1)^{s+t} \det(Q^*). \tag{2}$$

Proof

Consider $s = t$; the general statement follows from a result in linear algebra.
Step 1. If D is an orientation of G, and M is the incident matrix of D, then $Q = MM^T$ with edges $e_1, e_2, \cdots, e_m$, the entries of M are m_{ij},

$$m_{ij} = \begin{cases} 1, & \text{if } v_i \text{ is the tail of } e_j \\ -1, & \text{if } v_i \text{ is the head of } e_j \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Entry ij in MM^T is the dot product of rows i and j of M.
When $i \neq j$, the product counts -1 for every edge of G joining the two vertices; when $i = j$, it counts 1 for every incident edge and yields the degree.

Proof - continued

Step 2. If B is an $(n-1) \times (n-1)$ submatrix of M, then

$$\det(B) = \begin{cases} \pm 1, & \text{if the } n-1 \text{ edges form a spanning tree of } G \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Case 1. The corresponding $n-1$ edges form a spanning tree of G.
We prove by induction on n.
For $n = 1$, by convention a 0×0 matrix has determinant 1.
For $n > 1$, let T be a spanning tree whose edges are the columns of B. Since T has at least two leaves, and only one row is deleted, B has a rwo corresponding to a leaf x of T. This row has only one nonzero entry in B.

Proof - continued

When computing the determinant by expanding alone this row, the only submatrix B' with nonzero weight in the expansion corresponds to the spanning subtree of $G - x$ obtained by deleting x and its incident edge from T. Since B' is an $(n-2) \times (n-2)$ submatrix of the incident matrix for an orientation of $G - x$, the inductive hypothesis yields $\det B' = \pm 1$. Since the nonzero entry in row x is ± 1, we obtain the same result for B.

Case 2. The $n - 1$ edges corresponding to columns of B do not form a spanning tree.

In this case, the edges contain a cycle. We form a linear combination of the columns with coefficient 0 if the corresponding edge is not in C, $+1$ if it is followed forward by C, and -1 if it is followed backward by C. The result is total weight 0 at each vertex, so the columns are linearly dependent, which yields $\det B = 0$.

Binet-Cauchy Formula

For $n \leq m$, A be an $n \times m$ matrix, B be an $m \times n$ matrix and $C = AB$.
Then

$$\det C = \sum_{S \subseteq [m], |S| = n} \det A_S \det B_S, \tag{5}$$

where A_S is the $n \times n$ submatrix of A consisting of the columns indexed by A, B_S is the $n \times n$ submatrix of B consisting of the n rows of B indexed by S.

Proof - continued

Step 3 Computation of $\det Q^*$.
Let M^* be the result of deleting row t of M. Then $Q^* = M^*(M^*)^T$. If $m < n - 1$, then the determinant is 0 and there are no spanning subtrees, so we assume the $m \geq n - 1$. When $m \geq p$, A is $p \times m$, and B is $m \times p$, and

$$\det AB = \sum_S \det A_S \det B_S,$$

where the sum runs over all p-sets S in $[m]$, A_S is the submatrix of A consisting of the columns indexed by S, and B_S is the submatrix of B consisting of the rows indexed by S.

Proof - continued

When we apply the formula to $Q^* = M^*(M^*)^T$, the submatrix A_S is an an $(n-1) \times (n-1$ matrix of M, and $B_S = A_S^T$. Hence the summation counts $1 = (\pm 1)^2$ for each set of $n-1$ edges corresponding to a spanning tree, and 0 for all other sets of $n-1$ edges.

Minimum Spanning Trees

Definition 6

A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the least possible sum of weights of its edges.

We introduce two algorithms to find the minimum spanning trees.

Kruskal's Algorithm

Input: A weighted connected graph.
Idea: Maintain an acyclic spanning subgraph H, enlarging it by adding the edges with the least possible weights to form a spanning tree.
Suppose that

$$e_1, e_2, \cdots, e_m$$

are all the edges with nondecreasing weights.
Initialisation Set $V_H = V$ and $E_H = \emptyset$.
Iteration:

1. If H is connected, then terminate with output $H = (V_H, E_H)$.
2. Otherwise. Let e be the edge with the least weight such that
   - $e \in E \setminus E_H$,
   - e joins two connected components of the current H.

   Then,
   enumerate e into E_H, and go on to the next iteration.

Correctness of the Algorithm

Theorem 7

(Kruskal, 1956) In a connected graph G, the Kruskal's algorithm outputs a minimum spanning tree of G

By the construction of H, H is acyclic and connected, so is a spanning tree of G.

Proof

Let T be the tree H constructed by the algorithm, and T^* be a minimum spanning tree of G.
If $T \neq T^*$, then there are edges that are in T, but not in T^*. Let e be the first such edge.
Assume that e is enumerated into T at iteration k.
Then $T^* + e$ creates a unique cycle C. Since T has no cycle, there is an edge $e' \in C$ such that $e' \notin E_T$. However, $e' \in E_{T^*}$.
Consider

$$T^* + e - e'.$$

Since $e' \notin E_T$, at the iteration at which e is enumerated into E_T, both e and e' are available. So $w(e) \leq w(e')$.
Therefore $T^* + e - e'$ is a spanning tree with weight at most the same as that of T^*.

Proof - continued

Then $T_1 = T^* + e - e'$ is a spanning tree of G such that for each $j \leq k + 1$, T and T_1 share the edges enumerated by the algorithm at iterations j.
Repeating the same argument, there is a minimum spanning tree T_N of G that is equal to T.

The time complexity is:

$$O(m \log m).$$

Prim's Algorithm

Input: $G = (V, E)$ a connected weighted graph.
Idea Each time, adding a new vertex through the least weight edge.
Initialisation: Set $X = \{v\}$, $E_T = \emptyset$, v is a fixed vertex in V
Iteration If $X = V$, then terminate and output $T = (X, E_T)$.
Otherwise, then let y be the vertex satisfying:

- $y \in V \setminus X$,
- there is an edge (y, x) for some $x \in X$, denoted by $yx = e$,
- for any $x \in X$ and any $z \in V \setminus X$, if $xz \in E$, then $w(xz) \geq w(e)$.

Then:
enumerate y into X and $e = yx$ into E_T for the x with $yx = e$.

Proof

Let $G = (V, E)$ be a connected weighted graph. Suppose that

$$e_1, e_2, \cdots, e_{n-1}$$

are the edges chosen by the algorithm with the order as they are listed.
For each k, $1 \le k \le n-1$, let T_k be the tree consisting of edges $e_1, e_2, \cdots, e_k$.
Let T^* be a minimum spanning tree of G that contains edges $e_1, e_2, \cdots, e_k$.
Assume $e_{k+1} \notin T^*$.
Consider $T_1 = T^* + e_{k+1}$.
T_1 contains a unique cycle C with e_{k+1} as an edge.
Since T_{k+1} is a tree, there is an edge $e \in C$ such that is not in T_{k+1} and such that e shares an endpoint with one of the edges $e_1, e_2, \cdots, e_k$.

Proof - continued

By the choice of e_{k+1},

$$w(e_{k+1}) \leq w(e).$$

Let $T_2 = T^* + e_{k+1} - e$.
Then T_2 is a minimum spanning tree of G which shares edges $e_1, e_2, \cdots, e_{k+1}$ with T.
Continuing the procedure, we are able to find a minimum spanning tree of G that shares all the edges $e_1, e_2, \cdots e_{n-1}$ with T.

The challenge

Open Question Is there an algorithm that finds a path from any vertex to any other vertex in a network $G = (V, E)$ in time $\mathrm{poly}(D, \log n)$, where D is the diameter of G?

Note, in a small world, $D = O(\log n)$, or $\mathrm{poly}(\log n)$.

- Stanley Migram's experiment shows yes
- Kleinberg's results show no

New ideas

In Migram's experiments, people use the global clues of his/her acquaintances to forward a letter.
Question: How to extract global clues of a graph?

Fingerprint of a graph - 1

Definition 8

Let $G = (V, E)$ be a connected graph with diameter $D = O(\log n)$ of $\mathrm{poly}(\log n)$. A *fingerprint* T of G is a priority tree of the codewords of the vertices of G that is constructed by a random BFS procedure:

(1) Let v be a vertex of G chosen randomly and uniformly. Let λ be the root node of T. We say that λ is the *codeword* of v, written $c(v) = \lambda$ and v is the *marker* of λ, written $m(\lambda) = v$.

Fingerprint of a graph - 2

- We say that a vertex $u \in V$ is *marked*, if the codeword $c(u)$ of u is defined.
- We say that a tree node $\alpha \in T$ is *met*, if for the marker v of α, all the neighbor vertices of u in G have been marked.
- We say that a tree node α *requires attention* if α has not been met.

Fingerprint of a graph - 3

(2) If there is a tree node that requires attention, then let α be the tree node with $|\alpha|$ least and then with $<_{\mathrm{L}}$-least that requires attention. Let v be the marker of α.
Suppose that $u_0, u_1, \cdots, u_{k-1}$ are all the neighbors of v in G that have not marked yet, with ordering listed randomly and uniformly.

(3) The immediate successors of α are $0 <_{\mathrm{L}} 1 <_{\mathrm{L}} \cdots <_{\mathrm{L}} (k-1)$. For every j from 0 to $k-1$, define the *codeword* of u_j to be $\alpha\hat{}\langle j\rangle$, and the *marker* of $\alpha\hat{}\langle j\rangle$ to be u_j.

Therefore, a fingerprint T is a tree of the codewords of the vertices of G.

The complexity of the construction of a fingerprint T

- Time

$$O(m)$$

- Space

$$\tilde{O}(n),$$

where n and m are the number of vertices and edges of G, respectively.

The height of T

The height of T if $h(T)$, satisfying:

$$\frac{D}{2} \leq h(T) \leq D.$$

T-distance

Given two vertices $x, y \in V$, let α and β be the codewords of x and y in T, respectively.
Let γ be the longest common initial segment of α and β. Then

$$d^T(x, y) = |\alpha| + |\beta| - 2 \cdot |\gamma|. \tag{6}$$

The time complexity of the computation of d^T is

$$O(D).$$

Shortest paths in T

For any two vertices x and y in V, let α and β be the codewords of x and y in T, respectively.
If either $\alpha \subseteq \beta$ or $\beta \subseteq \alpha$, then

$$d_G(x, y) = d^T(x, y), \tag{7}$$

where $d_G(x, y)$ is the length of the shortest path from x to y in G.

Algorithmic small world phenomenon

Theorem 9

(Huang, Lewis-Pye, Li, Li, Pan, Yao) Let $G = (V, E)$ be a graph with diameter D, and T be a fingerprint of G. Then

$$\Pr_{x,y \in_{\mathrm{R}} V}[d^T(x,y) \leq 3 \cdot d_G(x,y)] \geq \frac{2}{3}. \tag{8}$$

Proof

Let r be the marker of the root node λ of T. Then r is vertex randomly and uniformly chosen from V. For randomly chosen $x, y \in V$,

- if $d_G(x, r) \leq d_G(x, y)$, then $d_G(r, y) \leq 2 \cdot d_G(x, y)$,
- if $d_G(r, y) \leq d_G(x, y)$, then $d_G(x, r) \leq 2 \cdot d_G(x, y)$,
- in either case,

$$d^T(x, y) \leq d^T(x, r) + d^T(r, y) \leq 3 \cdot d_G(x, y),$$

which holds with probability at least $\frac{2}{3}$.

Interactive algorithm

It works as follows:
Human navigation For x, y, x forwards the letter to her neighbour z such that $d^T(z, y)$ is the least among $d^T(w, y)$ among all neighbours w of x.

- The algorithm interacts with T as an *auxiliary data structure*
- The auxiliary data structure is constructed in time $O(m)$ and space $\tilde{O}(n)$.
- The time complexity of the algorithm on input pair x, y is

  $$O(D \cdot \Delta),$$

  where Δ is the maximum degree of G.
  It is $\mathrm{poly}(\log n)$, if both D and Δ are $\mathrm{poly}(\log n)$.
- the algorithm is an *interactive algorithm* finding short paths in time $\mathrm{poly}(\log n)$.

Random spanning space

For N fingerprints $T_1, T_2, \cdots, T_N$ (for $N = O(\log n)$), for every vertex v, define the codeword of v by

$$c(v) = (\alpha_1, \alpha_2, \cdots, \alpha_N), \tag{9}$$

where α_j is the codeword of v in fingerprint T_j.
A *random spanning space* is the set of the codewords of all the vertices defined as above.
Using random spanning space as auxiliary data structure to develop interactive algorithms to find short or shortest paths in small worlds.

New direction

- to develop a tree method for graph algorithms
- to develop a theory of interactive algorithms

谢谢！