lec8

Anders Johansson

23 november 2011

Contents

1 Trees and forests

1.1 Characterisation of trees

(Figure of trees)

The following are equivalent for a simple undirected graph T on n vertices

1. T is a tree;
2. T is connected and contains no cycles;
3. T is connected and has $n-1$ edges;
4. T is connected and every edge is a bridge;
5. any two vertices is connected by exactly one path;
6. T is acyclic but the addition of any new edge creates exactly one new cycle.

1. What are the corresponding statements for forests?
2. Prove, say, (iii) $\implies$ (iv).

2 Spanning trees and spanning forests

Given a graph G, a spanning subgraph $T \subset G$ which is a tree is called a *spanning tree*. A *spanning forest* is a subgraph $F \subset G$ which is the vertex disjoin union of spanning trees in each component of G.

1. The graph G is connected $\iff$ G has a spanning tree T
2. For a spanning tree $T \subset G$, every edge in T corresponds to a unique bond B_e of G. (Recall a bond is a minimal cut-set.) Every bond contain some edge of T.
3. Every edge e of G not in T corresponds to a unique cycle $C_e \subset G$ and every cycle contain an edge from $G - E(T)$.
4. If a set $W \subset E(G)$ is such that $W \cap E(T) \neq \emptyset$ for *every* spanning tree T, then W is a (not necessarily minimal) cut-set.
5. Let $\tau(G)$ denote the number of spanning trees in G. Show that

$$\tau(G) = \tau(G - e) + \tau(G/e).$$

The cycles $\{C_e : e \notin T\}$ and the bonds $\{B_e : e \in T\}$ constitute basises for the cycle space $Z(G)$ and the cut-space $Z^{\perp}(G)$, respectively.

3 Leafs and Prüfer codes

A *leaf* is a vertex v in a tree T of degree one. The handshake lemma readily gives that

Every tree has at least two leaves.

Deleting a leaf v from T gives a tree $T - v$.

A Pr´ufer code is a way to code a tree T “bottom up”. Assume $V(T) = \{1, \dots, n\}$ (or order the vertices using a labeling). Consider the following tree:

To construct the Pr´ufer code for a (labeled) tree T, we iterate the following procedure.

- Take the first leaf in order, say, i and write down its neighbour j.

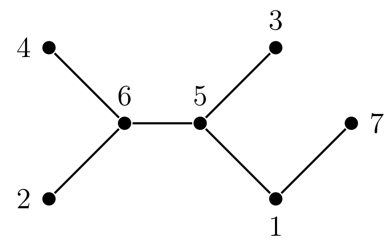


Figure 1: A simple tree on 7 vertices with code $(6, 5, 6, 5, 1)$.

- Delete the leaf i for T.

The code is thus a sequence $\mathbf{C} = (c_1, \ldots, c_{n-2}) \in V(T)^{n-2}$ of length $n - 2$ of vertices.

We can reconstruct the tree from its code $\mathbf{C}$ by first listing $\mathbf{L} = (1, 2, 3, \ldots, n)$ all vertices in the assumed order. Then iterate the following

- Let i be the first vertex in the list $\mathbf{L}$ not in the code. Add the edge ic where c is the first label in the code.
- Delete (Cross out) the first symbol c from $\mathbf{c}$ and delete i from $\mathbf{L}$.

As a final step, we add the edge between the two remaining vertices in $\mathbf{L}$.

Note that we have n^{n-2} Pr'ufer codes for trees $T \subset K_n$. Since we have established a bijection between the codes and trees, we conclude
Cayleys theorem: There are n^{n-2} (labeled) trees.

1. What trees have all symbols distinct in its code? Just one symbol appear?
2. What is the relation between the number of times a symbol appear and its degree?

4 The Matrix-tree theorem

Kirchhoff's matrix-tree theorem is a theorem about the number of spanning trees in a graph. It is a generalization of Cayley's formula which provides the number of spanning trees in a complete graph.

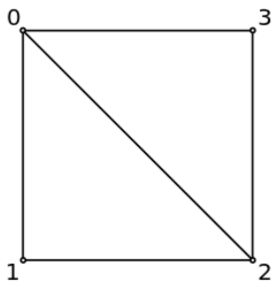


Figure 2: How many spanning trees does this kite graph have?

Kirchhoff's theorem uses cofactors of the *Laplacian matrix* $L = L(G)$ of a graph. Recall that $L = BB^{\mathsf{T}}$, where $B = B(G)$ is the incidence matrix relative some orientation, and that that is equal to $L = D - A$, where D is the diagonal matrix $\operatorname{diag}\deg(\cdot, G)$ and A is the *adjacency matrix*, $A_{ij} = 1$ if $i \operatorname{adj} j$ and zero otherwise. For example, for the "Kite graph", we have

$$L = [$$

For a given connected graph G with n labeled vertices, let $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$ be the non-zero eigenvalues of its Laplacian matrix. Then the matrisx-tree theorem states that the number of spanning trees of G is

$$\tau(G) = \frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}.$$

Equivalently the number of spanning trees is equal to the absolute value of *any* cofactor of the Laplacian matrix of G.

To obtain $\tau(G)$, we thus construct a $(n-1)\times(n-1)$-sub matrix of L by deleting any row and any column. For example,

$$L_{1,1} = [2 - 10 - 13 - 10 - 12].$$

Finally, we take the determinant to obtain $\tau G\tilde{}$, which in this case gives 8.

5 The depth-first and breadth first search

6 Shortest path problems

6.1 Combinatorial optimisation

Given a finite set $S \subset \{0,1\}^E$, sequences of 0 and 1 idexed by elements in some finite set E. A typical combinatorial optimisation problem is to find the minimum (or maximum) of some *objective function* $f(x)$, where $x \in S$.

Shortest path (SP) problem If S is the set of st-paths in a graph $G = (V, E)$ and $\ell : E \to R_+$ is a prescribed positive length. Minimise

$$\ell(x) = \sum_{e \in x} \ell(e) = \sum_{e \in E} \ell(e)x(e),$$

where in the last expression we consider a path x as a vector

$$x = (x_{e_1}, \ldots, x_{e_m}) \in \{0,1\}^E.$$

Minimum spanning tree (MST) problem Let $w : E \to R$ be a given weighting of the edges in a graph $G = (V, E)$. Let S be the set of spanning trees and minimise $w(T) = \sum_{e \in T} w(e)$.

The Huffman coding problem For

$$S = \{ \text{ complete binary trees with leaf weights } w(1), \ldots, w(m) > 0\}$$

minimise

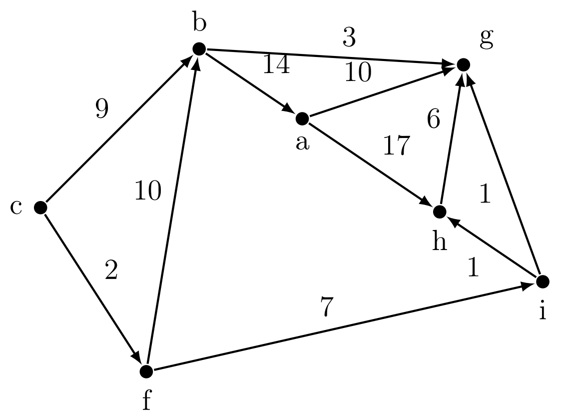
$$\sum_{\text{leafs } u} w(u)\ell(u).$$

The traveling salesman problem For $S = \{\text{Hamilton paths}\}$ minimise $\sum_{e \in H} w(e)$, where $w : E \to R$ is an edge weighting.

6.2 The directed shortest path problem.

Given a weighted digraph $G = (V, E, \ell)$, where the *positive* weighting $\ell : E \to R_+$ is called *length*, and a specified source $s \in V$ and sink $t \in V$.

1. What is the shortest directed path between c and a?



2. What is the shortest (undirected) oriented path between c and a?

The first problem is to find a *directed shortest path* from s to t. We can also try to find a maximal distance-minimising rooted (at s) directed tree in G — a distance tree —, i.e. a maximal sub-digraph T so that all branches out from s are shortest paths that minimise distance, i.e. for all u in $V(T)$ the path in T from s to u is a shortest path.

1. What is the distance-tree from c above?
2. Will this be an instance of the MST problem? NO.
3. Why positive lengths? What about negative length cycles.
4. Must $V(T)$ be spanning tree? NO. Describe the cut between $V(T)$ and $V \setminus V(T)$.

As will be explained later in the course, we solve these problem "dually". Instead of concetrating on the distance-tree, we try to construct a "dual" solution, namely the function $L : V \to R$, given by

$$L(v) = \text{dir-dist}_G(s, v),$$

where dir-dist(a, b) is the length of a shortest directed ab-path.

The function L is an example of a *value function.* Note: vertices can be thought of as "states" and $L(v)$ is the value for the problem if we are at state v — the shortest anti-directed path to the goal s.

We have, for $v \in V$ a kind of recursive formula for L

$$L(v) = \min \left\{ \ell(vu) + L(u) : u \in N_{-}(v) \right\}. \tag{*}$$

1. If $P : sx_1 \dots zy$ is a shortest sy-path, is $sx_1 \dots z$ a shortest sz-path? Yes

2. Show that (**??**) determines L uniquely, i.e. if $L(v)$ is any function satisfying (**??**) then $L(v) = \text{dir-dist}_G(s, v)$. (This is a special case of Bellmans optimality principle.)

6.3 The main loop in Dijkstra's algorithm

Algorithm 1 Main loop of value iteration.

procedure DIJKSTRA(G, ℓ) ▷ Digraph G and $\ell : E(G) \to R_{+}$
    For $v \in V$, let $L(v) \leftarrow \infty$ if $v \neq s$ and $L(s) \leftarrow 0$ and $P(v) \leftarrow \emptyset$.
    while $\exists x \, \exists y \in N_{-}(x) \quad L(x) > L(y) + \ell(yx)$ **do**
        $L(x) \leftarrow L(y) + \ell(yx)$
        $P(x) \leftarrow y$.
    end while
    return (L,P)
end procedure

1. How do we recreate the distance-tree from $P(v)$? What is the interpretation of $P(v)$? ($P(v)$ is the parent in the tree.)

2. Will this always *converge*? Yes — Value improvement. Will it *stop* in a finite time? Not necessarily.

3. When can we decide that a value $L(v)$ is safe, i.e. decidedly equal to the distance to s? Initially, we have the safe set $S = \{s\}$, but what about other times. If S is a set of "safe values" at a point in time and $L(v)$ minimises $L(x)$ for $x \notin S$. If for all x,

   $$L(x) = \min\{\ell(yx) + L(y) : y \in N_{-}(x)\}$$

   show that

   $$L(v) = L(u) + \ell(uv), \quad u \in S.$$

Algorithm 2 The safe version of Dijkstra's algorithm

procedure DIJKSTRA(G, ℓ)
    Initialise $L(v)$ and $P(v)$.
    $S \leftarrow \{s\}$.
    while $N_+(S) \not\subset S$ **do**
        for $x \in V \setminus S$ **do**
            for $y \in N_-(x)$ **do**
                if $L(x) > L(y) + \ell(yx)$ **then**
                    $P(x) \leftarrow y$.
                end if
            end for
        end for
        Let x minimise for $L(x)$, $x \notin S$ and let $S \leftarrow S \cup \{x\}$
    end while
    return (L, P)
end procedure

The consideration of safe values give the following refinement of Dijkstra's After each iteration of the loop in **??**, we have that

$$L(x) = \min\{\ell(yx) + L(y) : y \in N_-(x)\}.$$

Thus it is safe to extend S with one more element.

1. What does the condition $S \neq N_+(S)$ mean? How to change this if we only want a shortest st-path?
2. What about complexity? As it stands $O(|V| \times |E|)$.
3. Improvements? Do not scan all vertices in $V \setminus S$. Keep the set $N_+(S)$ in a heap ordered by L. ...