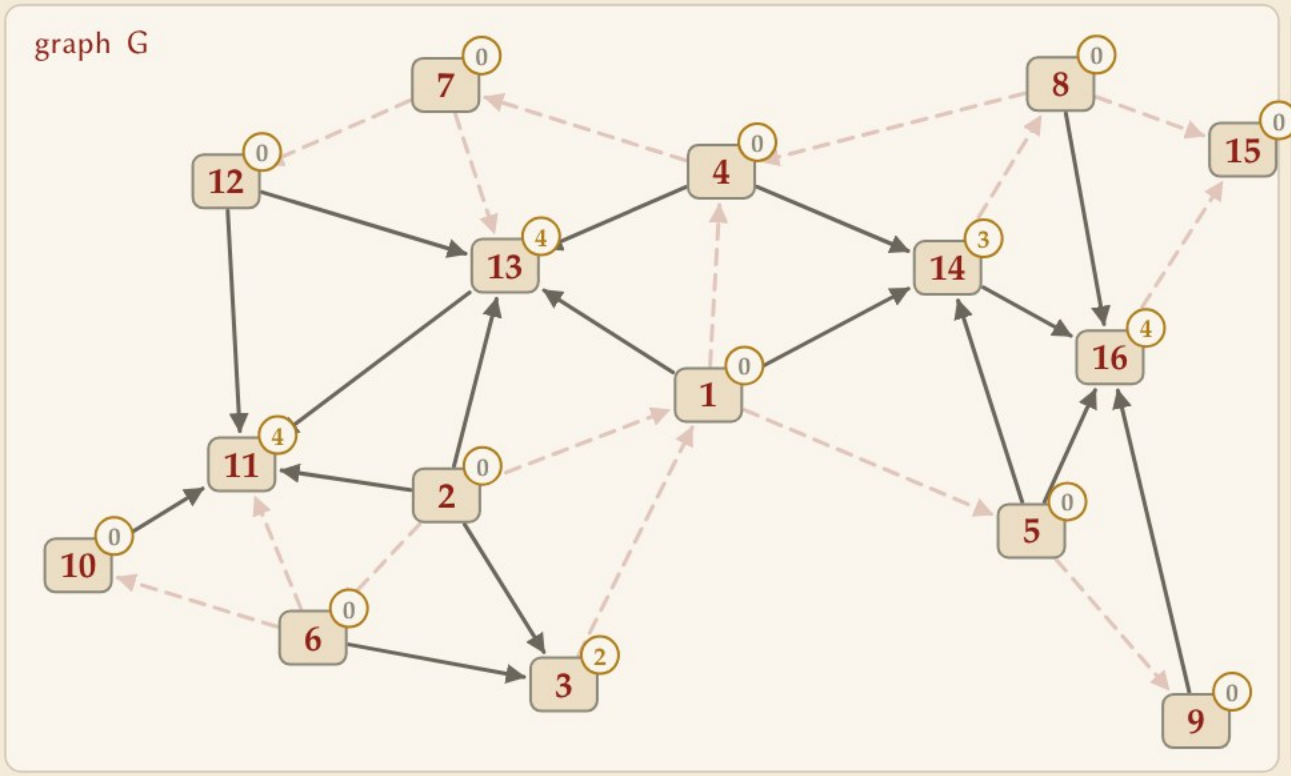
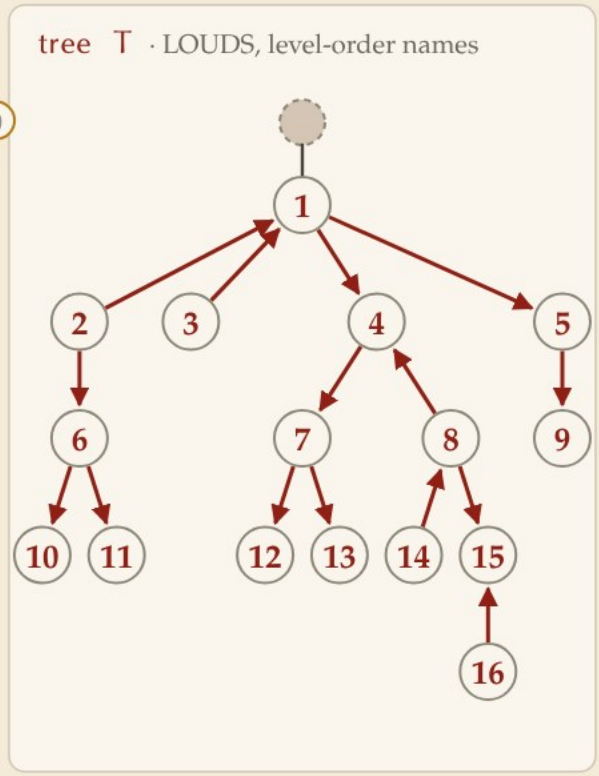
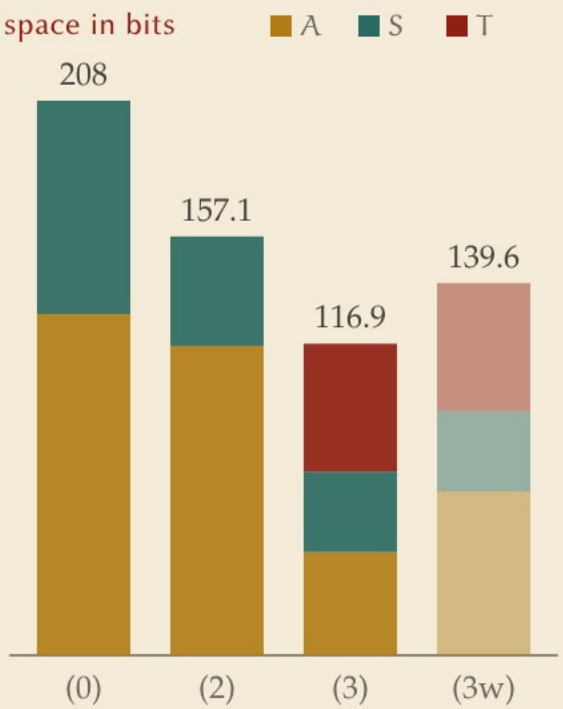
1 Tree extraction: the data structure





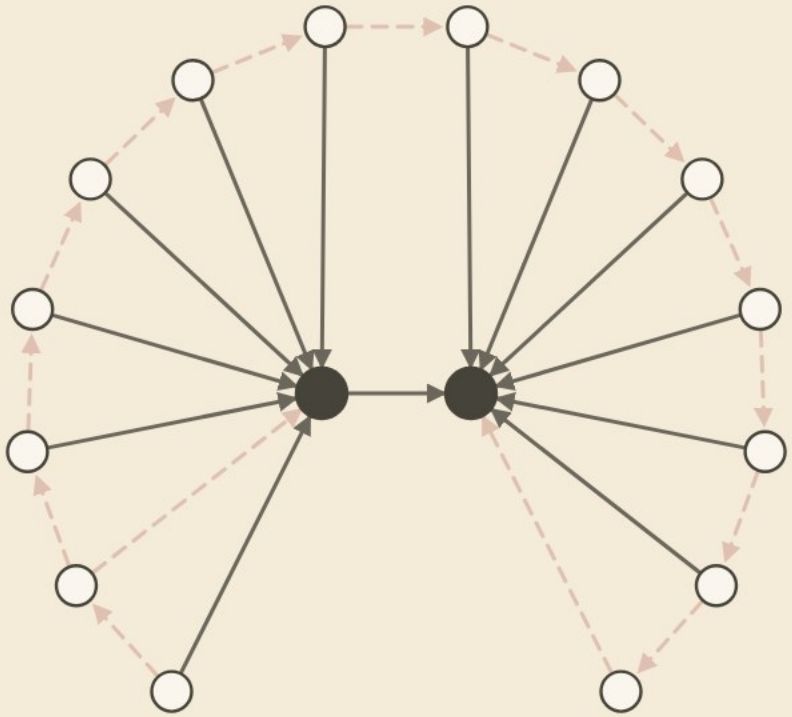


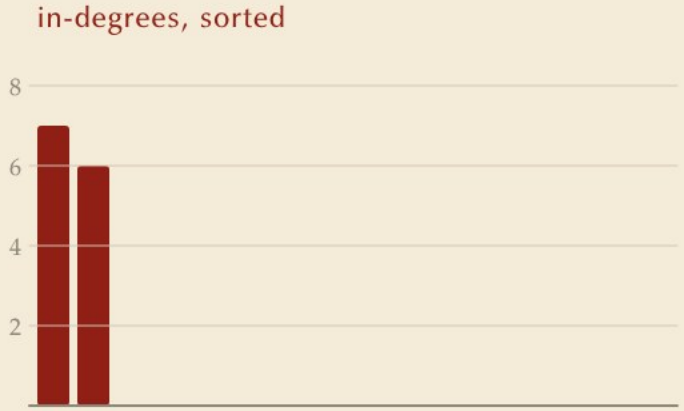
$A' =$ 13 14 | 3 | 11 13 | 13 14 | 14 16 | 3 | 16 | 16 | 11 | 11 13 | 11 | 16 — 38.7 bits

cell width = cost of the entry in bits

Tree extraction saves 40 bits here, $0.63\, n \lg n$: the label information we never needed. The worst greedy tree saves only 17 bits — **which tree** is the question of part 2.

2 Which tree? Minimum-entropy tree extraction





$H = \mathbf{12.9}$ bits (13 edges left)

greedy: Kruskal on edge weight $\lg d_v$

edges sorted by in-degree of their target

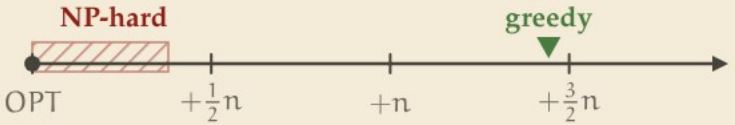
1	1	1	1	1	1	1
1	1	1	1	1	1	7
7	7	7	7	7	7	8
8	8	8	8	8	8	8

into the tree would close a cycle

NP-hard even to approximate within additive error $\mathrm{OPT} + \delta n$ for some $\delta > 0$.

greedy $H(G - T_{\mathrm{greedy}}) \leqslant \mathrm{OPT} + n/\ln 2$ in $O(n+m)$ time; undirected: error $(m+n-1)/\ln 2$.

Both bounds are linear in n: approximability settled up to the constant.



The greedy tree reaches 12.9 bits here, the optimum. In general it is within an additive $n/\ln 2$ — and that is best possible up to the constant.

3 Does it pay off? 18 benchmark graphs

(0) adjacency lists	(1) string + bitvector	(1u) undirected: edge once	(2) wavelet tree	(3) TREX, greedy tree	(4) + twin removal	Σ saving vs n lg n

Measured in units of $n \lg n$, tree extraction recovers almost all label information, consistently across graphs (median 0.95).

