1. IN AN ADJACENCY MATRIX FOR A GRAPH G, WE NUMBER THE VERTICES OF G AS $1 \ldots n$, AND FORM AN $n \times n$ MATRIX WITH A 1 IN THE (i,j) & (j,i) ENTRIES IF VERTICES i & j ARE ADJACENT, AND 0 OTHERWISE.

[FOR A DIGRAPH, WE PUT A 1 IN THE (i,j) ENTRY ALONE IF THERE IS AN EDGE FROM VERTEX i TO VERTEX j.]

(a) A NODE-WEIGHT FOR VERTEX i CAN BE STORED ALONG THE DIAGONAL IN THE (i,i) ENTRY, AND EDGE-WEIGHTS CAN BE USED INSTEAD OF 1's IN THE REST OF THE MATRIX.

(b) ADJACENCY MATRICES ARE GOOD FOR MATRICES THAT ARE SMALL OR DENSE (I.E., HAVING A LOT OF EDGES PER VERTEX), AS THE n^2 ENTRIES WILL BE AN EFFICIENT WAY TO STORE THE GRAPH.

2. TO REPRESENT A GRAPH G VIA ADJACENCY LISTS, WE NUMBER (OR LETTER, ETC.) ITS VERTICES AND FOR EACH VERTEX STORE A LIST OF ADJACENT VERTICES.

[FOR A DIGRAPH, WE COULD ALSO RECORD WHETHER EACH ADJACENT VERTEX IS A HEAD OR A TAIL, OR INSTEAD JUST RECORD THE VERTICES ADJACENT BY OUT-EDGES.]

(a) ADJACENCY LISTS ARE BETTER FOR SPARSE GRAPHS (IN WHICH EACH EDGE CONNECTS TO RELATIVELY FEW NEIGHBORS), AS WE ONLY STORE THE ADJACENT VERTICES.

3. (a)

	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	1	0
3	1	1	0	1	0	0
4	1	0	1	0	1	1
5	0	1	0	1	0	1
6	1	0	0	1	1	0

ADJACENCY MATRIX

1: 2, 3, 4, 6
2: 1, 3, 5
3: 1, 2, 4
4: 1, 3, 5, 6
5: 2, 4, 6
6: 1, 4, 5

ADJACENCY LISTS

(b)

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
2	1	0	1	1	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0
4	0	1	1	0	0	0	0	0	0
5	1	0	0	0	0	1	0	0	0
6	0	0	0	0	1	0	1	0	1
7	0	0	0	0	0	1	0	1	0
8	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	1	0	0	0

ADJACENCY MATRIX

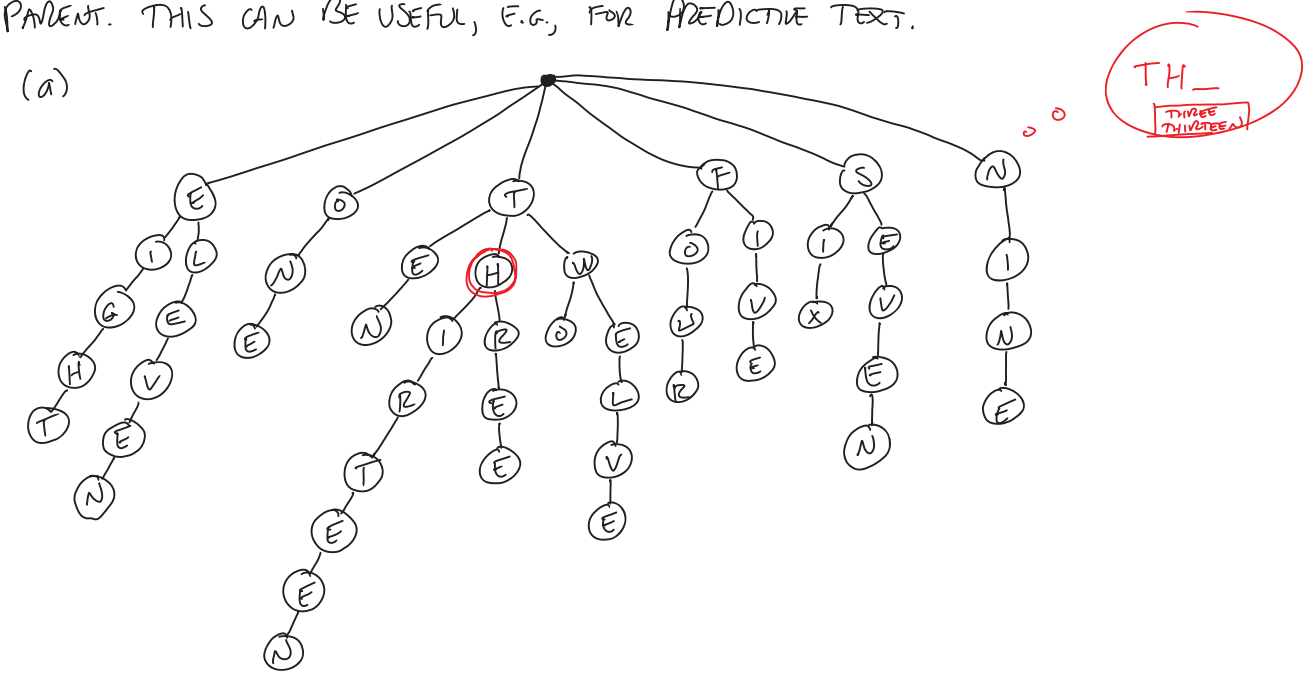
1: 2, 5
2: 1, 3, 4
3: 2, 4
4: 2, 3
5: 1, 6
6: 5, 7, 9
7: 6, 8
8: 7
9: 6

ADJACENCY LIST

3. A TRIE (OR PREFIX TREE) IS A ROOTED TREE WITH AN UNLABELED ROOT VERTEX AND EVERY VERTEX HAVING A CHILD FOR EACH OF SOME SET OF LETTERS — EACH FROM THE ROOT TO A LEAF VERTEX SPELLS OUT SOME WORD OR OTHER TEXT. (THIS CAN BE DONE FOR ANY SORT OF SEQUENCE OF TOKENS, AS WELL, NOT ONLY LETTERS.)

WALKING ALONG A TRIE MODELS WELL THE PROCESS OF ENTERING TEXT — ADDING A CHARACTER MOVES TO A CHILD VERTEX, AND "BACKSPACING" MOVES UP TO THE PARENT. THIS CAN BE USEFUL, E.G., FOR PREDICTIVE TEXT.

(a)

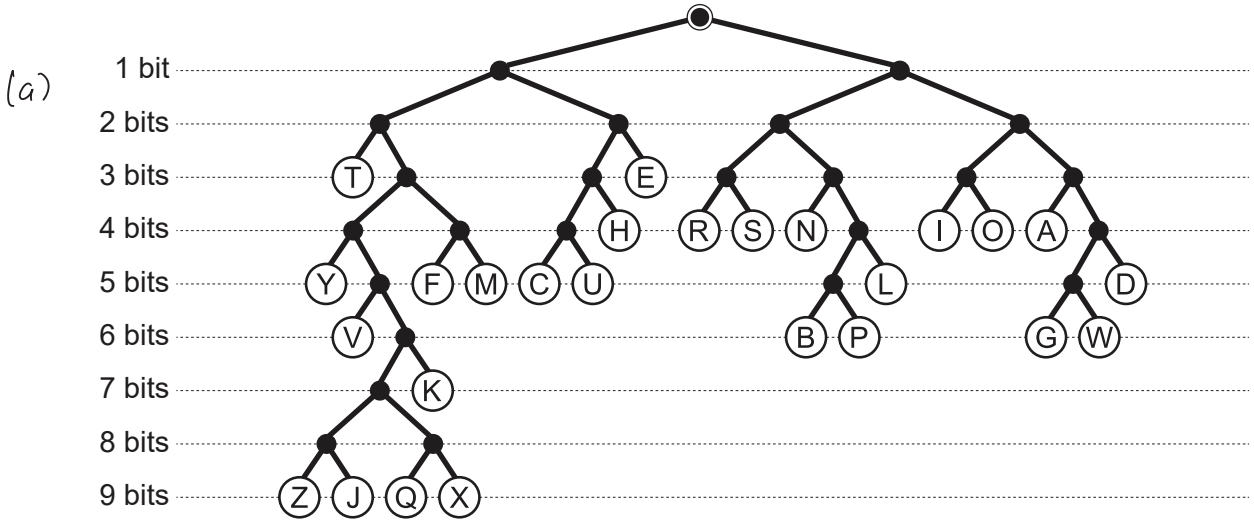


5. GIVEN A SET OF TOKENS (E.G., LETTERS) WITH RELATIVE FREQUENCY DATA FOR EACH ONE, HUFFMAN ENCODING GIVES THE PROVABLY BEST WAY OF ENCODING, IN BINARY, SEQUENCES OF TOKENS FOLLOWING THOSE RELATIVE FREQUENCIES:

- LIST OFF ALL TOKENS, SMALL-TO-LARGE IN FREQUENCY.
- ITERATIVELY TAKE THE TWO WITH THE LEAST FREQUENCY AND JOIN THEM BY A PARENT NODE WITH THE SUM OF THE FREQUENCIES (REMOVE THEM FROM THE LIST AND INSERT THE NEW NODE).

BECAUSE THE RESULTING CODE IS PREFIX-FREE (THE TREE TELLS US WHEN A CODE WORD IS FINISHED), WE ALWAYS KNOW WHERE TO SPLIT THE RESULTING STRING OF BITS.

(a)



(b) 5 BITS PER LETTER × 63 LETTERS = 315 BITS

HUFFMAN: 258 BITS

(c) THIS ENCODING WILL NOT BE OPTIMAL FOR ALL SETS OF WORDS! E.G., FOR AIRPORT CODES, THERE ARE LOTS OF LETTERS LIKE X (9 BITS!). THE TREE, AND THUS THE ENCODING, ARE TUNED TO ONE PARTICULAR DISTRIBUTION OF LETTERS.

6. IN A MARKOV CHAIN, THERE ARE A FINITE SET OF STATES, EACH WITH SOME SET OF PROBABILITIES FOR TRANSITION TO ANOTHER (OR THE SAME) STATE, SUCH THAT FOR EACH STATE, THE TRANSITION PROBABILITIES ARE NONNEGATIVE AND SUM TO 1.

(a) A DIGRAPH IS THE NATURAL OBJECT TO USE TO REPRESENT A MARKOV CHAIN:
- A VERTEX FOR EACH STATE, AND
- A DIRECTED EDGE FOR EACH TRANSITION, WEIGHTED BY THE TRANSITION PROBABILITY.

AT EACH VERTEX, THE OUTGOING WEIGHTS MUST BE NONNEGATIVE AND SUM TO 1.

(b) WE CAN EXPLORE A MARKOV CHAIN COMPUTATIONALLY VIA MCMC, STARTING AT A VERTEX AND RANDOMLY WALKING ALONG THE GRAPH, FOLLOWING THE GIVEN TRANSITION PROBABILITIES — DOING THIS A LARGE NUMBER OF TIMES AND RECORDING THE RESULTS CAN SHED SOME LIGHT ON THE BEHAVIOR OF THE SYSTEM.

(c)