Tightly Packed Tries

How to fit Large Models in Memory, and Make them Load Fast, Too

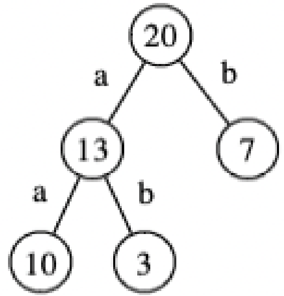
IN4120 - Sayna Ganjei – November 2025

What is a Trie?

- Each node stores a pointer to its child nodes
- On a 64-bit system, each pointer can consume 8 bytes, this can add upp, especially when there are many nodes
- This representation is flexible, but leads to hight memory usage, as each node includes pointers

total count	20
a	13
aa	10
ab	3
b	7

(a) Count table



(b) Trie representation

field	32-bit	64-bit
index entry: token ID	4	4
index entry: pointer	4	8
start of index (pointer)	4	8
overhead of index structure node value	x	y
total (in bytes)	$12 + x$	$20 + y$

(c) Memory footprint per node in an implementation using memory pointers

What is a Tightly Packed Tries (TPT) ?

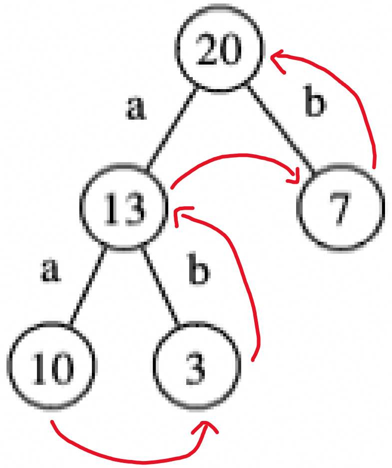
- TPT is a compact implementation of compressed trie structures with fast on-demand paging and short load times.
- TPT solves the problem of traditional tries by discarding pointers and replacing them with "relative offsets."

TSP: Using a Contiguous Byte Array Instead of Pointers

Each node includes:

- **Node value –** The value/cout associated with that node
- **Size of the index –** how many children the node has
- **Relative offset –** the distance between the paren tnode and its child node (in bytes)

Follows postorder traversal:
10 – 3 – 13 – 7 – 20
'aa' – 'ab' – 'a' – 'b' – root



(b) Trie representation

Node	Byte	Description	Value
ROOT	0	Offset of root node	13
'aa'	1	**Node value** - Only stores the node value since 'aa' has no children	10
	2	size of index (no children)	0
'ab'	3	**Node value** - Only stores the node value since 'aa' has no children	3
	4	size of index (no children)	0
'a'	5	**Node value**	13
	6	'a' has two child nodes, add 4 bytes/rows bellow	4
	7	Key for child **'aa'**	'a'
	8	Relative offset form 'a' to node 'aa' (5-4 = 1) (the distance to the **node value**)	4
	9	Key for child **'ab'**	'b'
	10	Relative offset form 'a' to node 'ab' (5-2 = 1) (the distance in bytes)	2
'b'	11	**Node value** - Only stores the node value since 'aa' has no children	7
	12	size of index (no children)	0
root	13	**Node value**	20
	14	'a' has two child nodes, add 4 bytes/rows bellow	
	15	Key for child 'a'	'a'
	16	Relative offset from root to 'a' (13-8 = 5) (the distance to the **node value**)	8
	17	Key for child 'b'	'b'
	18	Relative offset from root to 'b' (13-2 = 11) (the distance to the **node value**)	2

Compression Techniques (compressed index)

Trie compression by **variable-length coding**

- Variable length coding is a lossless compression technique where:
  - Frequent terms are encoded with shorter codes
  - Infrequent terms are given longer codes

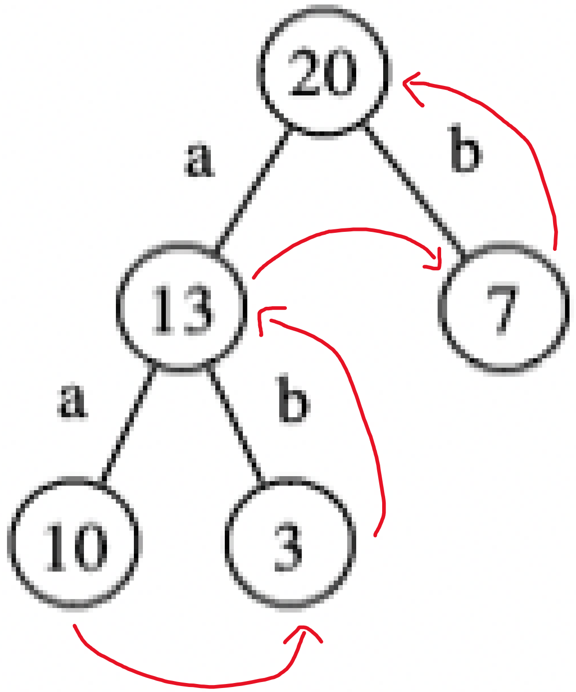
This approach leverages the fact that some terms appear more often than others. By assigning shorter codes to frequent occuring symbols, we can reduce the overall storage requirements.

«Stand alone» - node values, if they are integers, and the size of the index

«node indices » - the lists of child nodes and the respective arc labels

Memory Mapping

- TPTs use memory mapping to directly link a file to a memory region, enabling on-demand data loading for quick access and efficient memory use through OS paging.
- The paper uses the Boost Iostreams library in C++ for memory mapping.
- This approach lets the OS handle disk fallback, without having to design and code our own page management system.



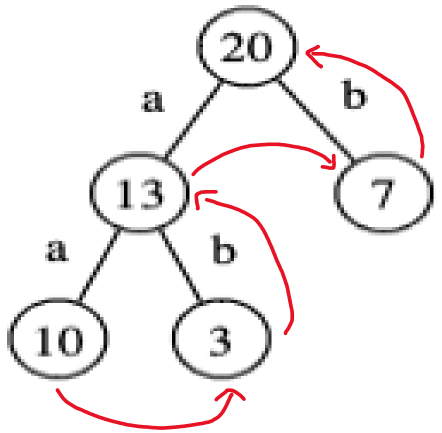
(b) Trie representation

Additional Tweeks to Further Tightly Pack the Tries

- Bit shifting:
  - Each node key value is shifted two bits to the left
- With the use of flags:
  - **Node Value Precence Flag:** Does it have an actual stored value, or does it use a default value?
  - **Terminal Node Flag:** Is it a terminal node (a leaf with no children), or does it have children?

Exsample ‘a’ :

Original Key (binary)	Shifted Key (Left 2 bits)	Flags (Vlaue Precence, Terminal)	Shifted Key with Flags (Binary)
0101	010100	(1, 0)	010110



(b) Trie representation

Encoding Node Values in Various NLP Applications

- **Count Tables –** Used to represent counts, such as how often sequences co-occur (e.g., bilingual phrases in machine translation).
  - Counts are stored as compressed integers for space efficiency.
  - To represent sequence co-occurrences, two sequences are concatenated with a special marker (an extra token) to separate them.
  - Use Case: Bilingual phrase pairs in SMT.

Encoding Node Values in Various NLP Applications

- **Back-off Language models** – A back-off language model estimates the probability of a word based on its context, handling cases where full context data isn't available.
  - Back-Off Mechanism: When probabilities for a longer context are unavailable, shorter contexts are used, scaled by back-off weights.

Encoding Node Values in Various NLP Applications

- **Phrase Tables for SMT** – use a bottom-up trie to store phrases compactly, and entropy encoding for scores to achive high compression

Experiments: N-gram Language Models

- Lower perplexity generally indicates a better predictive model
- Uses a 5-gram langugae model trained on the English Gigaword corpus
- Test text of 275,000 tokes (words)
- Language models for comparison:
  - SRILM
  - IRSTLM
  - Portage (pointer-based)
  - TPT
- Metrics:
  - Memory use
  - Runtime

Table 1: Memory use and runtimes of different LM implementations on a perplexity computation task.

		file/mem. size (GB)				1st run (times in sec.)					2nd run (times in sec.)				
		file	virt.	real	b/ng[1]	ttfr[2]	wall	usr	sys	cpu	ttfr	wall	usr	sys	cpu
full model loaded	**SRILM[3]**	5.2	16.3	15.3	42.2	940	1136	217	31	21%	846	1047	215	30	23%
	SRILM-C[4]	5.2	13.0	12.9	33.6	230	232	215	14	98%	227	229	213	14	98%
	IRST	5.1	5.5	5.4	14.2	614	615	545	13	90%	553	555	544	11	100%
	IRST-m[5]	5.1	5.5	1.6	14.2	548	744	545	8	74%	547	549	544	5	100%
	IRST-Q[6]	3.1	3.5	3.4	9.1	588	589	545	9	93%	551	553	544	8	100%
	IRST-Qm	3.1	3.5	1.4	9.1	548	674	546	7	81%	548	549	544	5	99%
	Portage	8.0	10.5	10.5	27.2	120	**122**	90	15	85%	110	112	90	14	92%
	TPT	**2.9**	**3.4**	**1.4**	**7.5**	**2**	127	**2**	2	**2%**	**1**	**2**	**1**	**1**	98%

Experiments: Statistical Machine Translation (SMT)

Table 3: Model load times and translation speed for batch translation with the *Portage* SMT system.

# of sentences per batch	Baseline			TPPT + Baseline LM			TPLM + Baseline PT			TPPT + TPLM		
	load time	w/s[1]	w/s[2]	load time	w/s[1]	w/s[2]	load time	w/s[1]	w/s[2]	load time[3]	w/s[1]	w/s[2]
47	210s	5.4	2.4	16s	5.0	4.6	178s	5.9	2.67	< 1s	5.5	5.5
10	187s	5.5	0.8	16s	5.1	3.6	170s	5.6	0.91	< 1s	5.6	5.6
1	—	—	—	15s	5.0	1.0	154s	5.5	0.12	< 1s	5.3	5.2

Baseline: *Portage*'s implementation as pointer structure with load-time filtering.
TP: Tightly packed; **PT:** phrase table; **LM:** language model
[1] words per second, excluding load time (pure translation time after model loading)
[2] words per second, including load time (bottom line translation speed)

Conclusion

TPT Efficiency:

- Significant reductions in memory usage and load times.
- Compact encoding and bit manipulation optimize storage.

Performance Gains:

- Faster translation speeds in SMT systems.
- High CPU efficiency, especially with cached data.

Ideal for NLP Applications:

- Suited for large-scale, memory-constrained, and real-time tasks.
- Balances compact storage with high processing efficiency.

Impact on Language Processing:

- Valuable advancement for language modeling and SMT.
- Improves scalability and responsiveness of NLP systems.

Sources

- Germann, Ulrich & Joanis, Eric & Larkin, Samuel. (2009). Tightly packed tries. 31-39. 10.3115/1621947.1621952.