5. Predictive text compression methods

Change of viewpoint:

- Emphasis on *modelling* instead of *coding.*

Main alternatives for text modelling and compression:

1. *Predictive* methods:
   - One symbol at a time
   - Context-based probabilities for entropy coding
2. *Dictionary* methods:
   - Several symbols (= substrings) at a time
   - Usually not context-based coding

Purpose of a predictive model

- Supply probabilities for message symbols.
- A good model makes good *’predictions*’ of symbols to follow.
- A good model assigns a *high probability* to the symbol that will actually occur.
- A high probability will not ’waste’ code space e.g. in arithmetic coding.
- A model can be *static* (off-line coding in two phases) or *dynamic* (adaptive, one-phase coding)

(1) Finite-context models

- A few (k) preceding symbols ('k-gram') determine the *context* for the next symbol.
- Number k is called the *order* of the model.
- Special agreement:
  $k = -1$ means that each symbol has probability $1/q$
- A distribution of symbols is built (maintained) for each context.
- In principle, increasing k will improve the model.
- Problem with large k:
  Reliable statistics cannot be collected;
  the (k+1)-grams occur too seldom.

Illustration of a finite-context model

Sample text:

"... compression saves resources ..."

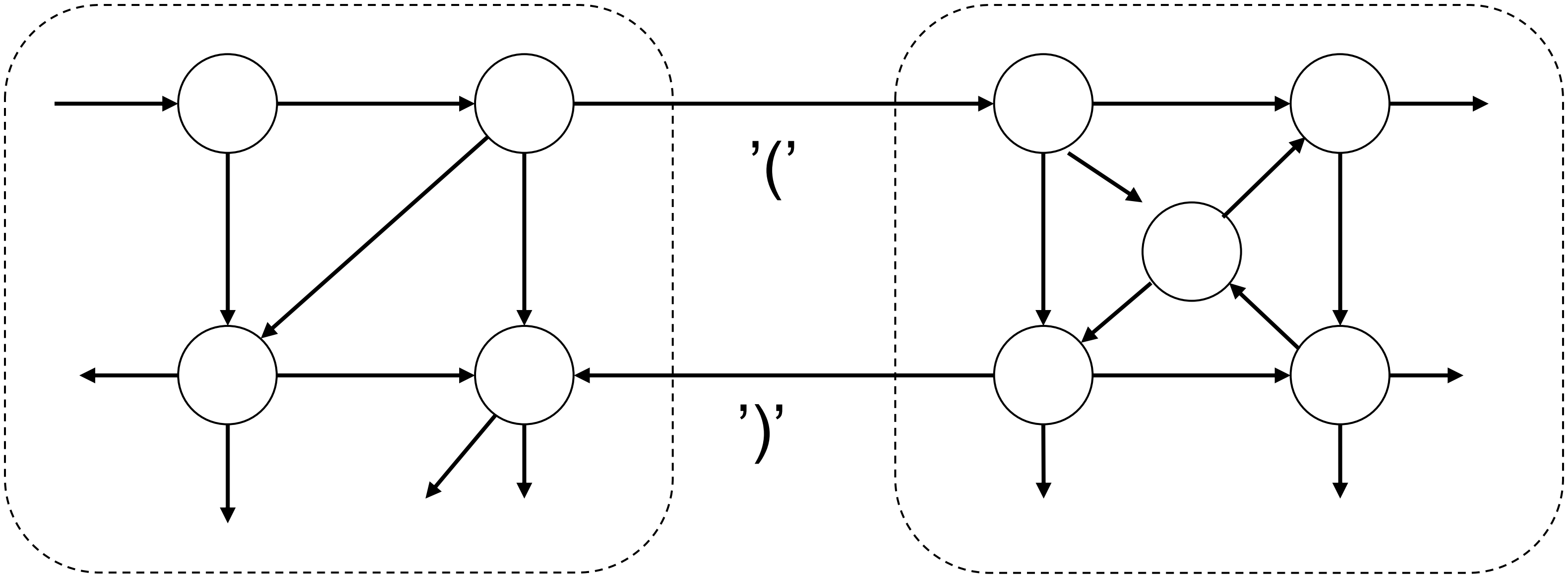
Context	Successor	Prob
…	…	…
com	e	0.2
com	m	0.3
com	p	0.5
…	…	…
omp	a	0.4
omp	o	0.3
omp	r	0.3
…	…	…

(2) Finite-state models

- May capture non-contiguous dependencies between symbols; have a limited *memory.*
- Are also able to capture regular blocks (alignments)
- Markov model
- Finite-state machine: states, transitions, trans.probabilities
- Compression: Traversal in the machine, directed by source symbols matching with transition labels.
- Encoding based on the distribution of transitions leaving the current state.
- Finite-state models are in principle stronger than finite-context models; the former can simulate the latter.
- Automatic generation of the machine is difficult.
- Problem: the machine tends to be very large.

Finite-state model: The memory property

Modelling of matching parentheses:
“ …(a+b)(c-d) + (a-c)(b+d)…”



States with low probability for ’)’

States with higher probability for ’)’

(3) Grammar models

- More general than finite-state models.
- Can capture arbitrarily deep nestings of structures.
- The machine needs a *stack*.
- Model description: *context-free grammar* with probabilities for the production rules.
- Automatic learning of the grammar is not feasible on the basis of the source message only.
- Natural language has a vague grammar, and not very deep nested structures.
- Note: *XML* is a good candidate for compressing using a grammar model (implementations exist).

Sketch of a grammar model

- Production rules for a fictitious programming language, complemented with probabilities :

<program> := <statement>[0.1] |
<program> <statement> [0.9]

<statement> := <control statement> [0.3] |
<assignment statement> [0.5] |
<input/output statement> [0.2]

<assignment statement> := <variable> '=' <expression> [1.0]

<expression> = <variable> [0.4] |
<arithmetic expression> [0.6]

……

5.1. Predictive coding based on fixed-length contexts

Requirements:

- Context (= prediction block) length is fixed = k
- Approximations for successor distributions
- Default predictions for unseen contexts
- Default coding of unseen successors

Data structure:

- Trie vs. hash table
- Context is the argument of the hash function H
- Successor information stored in the home address
- Collisions are rare, and can be ignored; successors of collided contexts are mixed
- Hash table more compact than trie: contexts not stored

Three fast fixed-context approaches of increasing complexity

1. Single-symbol prediction & coding of success/failure

2. Multiple-symbol prediction of probability order & universal coding of order numbers

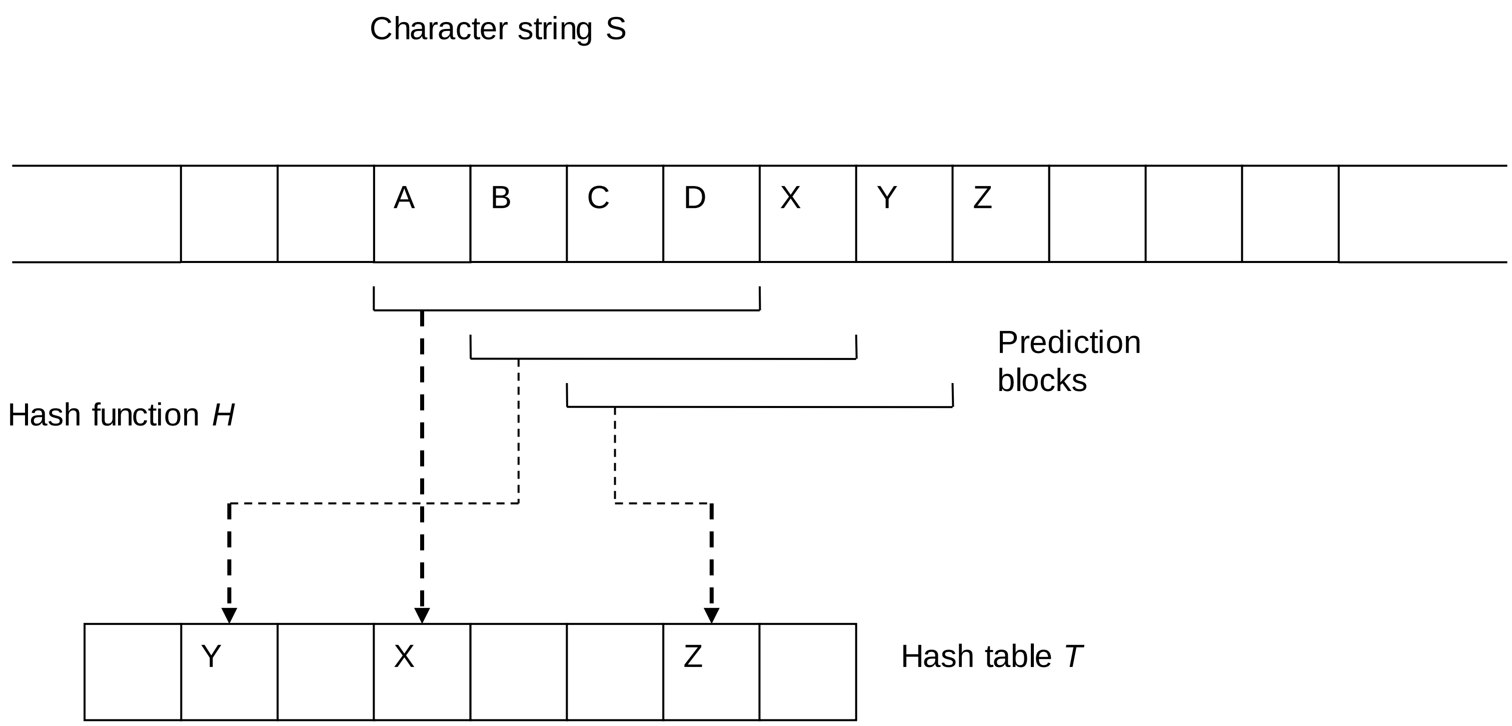
3. Multiple-symbol prediction of probabilities & arithmetic coding

A. Prediction based on the latest successor

Algorithm 5.1. Predictive success/failure encoding using fixed-length contexts.
Input: Message $X = x_1x_2 \ldots x_n$, context length k, hashtable size m, default symbol d
Output: Encoded message, consisting of bits and symbols.

begin
    for i := 0 **to** m–1 **do** $T[i]$:= d
    Send symbols $x_1, x_2, \ldots, x_k$ as such to the decoder
    for i := k+1 **to** n **do**
    begin
        addr := $H(x_i{-}k \ldots x_i{-}1)$
        pred := T[*addr*]
        if *pred* = x_i
            then Send bit 1 /* Prediction succeeded */
            else begin
                Send bit 0 and symbol x_i /* Prediction failed */
                T[*addr*] := *pred*
            end
    end
end

Prediction based on the latest successor: data structure

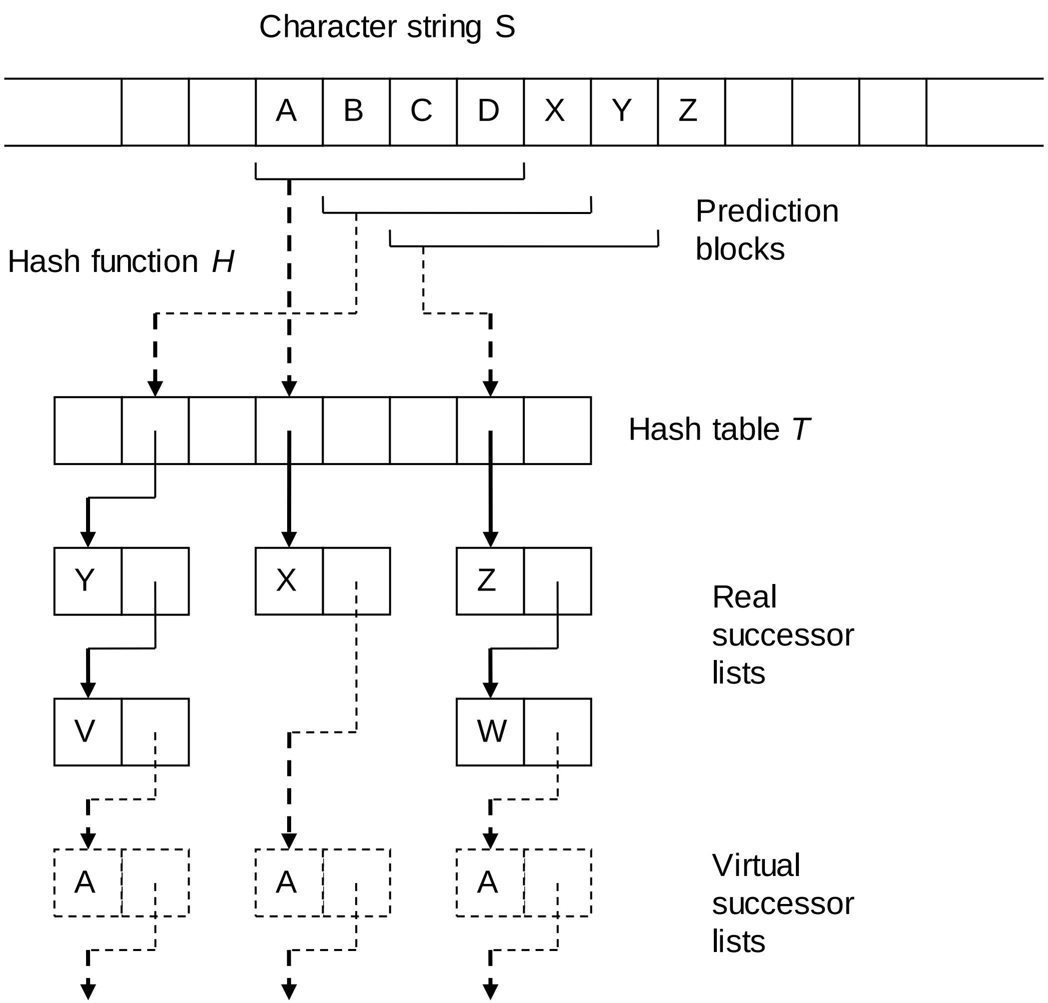


B. Prediction of successor order numbers

Algorithm 5.2. Prediction of symbol order numbers using fixed-length contexts.
Input: Message $X = x_1x_2 \ldots x_n$, context length k, hash table size m.
Output: Encoded message, consisting of the first k symbols and γ-coded integers.

begin
  for i := 0 **to** m–1 **do** $T[i]$:= NIL
  Send symbols $x_1, x_2, \ldots, x_k$ as such to the decoder
  for i := k+1 **to** n **do**
  begin
    $addr$:= $H(x_{i-k} \ldots x_{i-1})$
    if x_i is in list $T[addr]$
    then begin
      r := order number of x_i in $T[addr]$
      Send $\gamma(r)$ to the decoder
      Move x_i to the front of list $T[addr]$
      end
    else begin
      r := order number of x_i in alphabet S, ignoring symbols in list $T[addr]$
      Send $\gamma(r)$ to the decoder
      Create a node for x_i and add it to the front of list $T[addr]$
      end
  end
end

Prediction of successor order numbers: the data structure

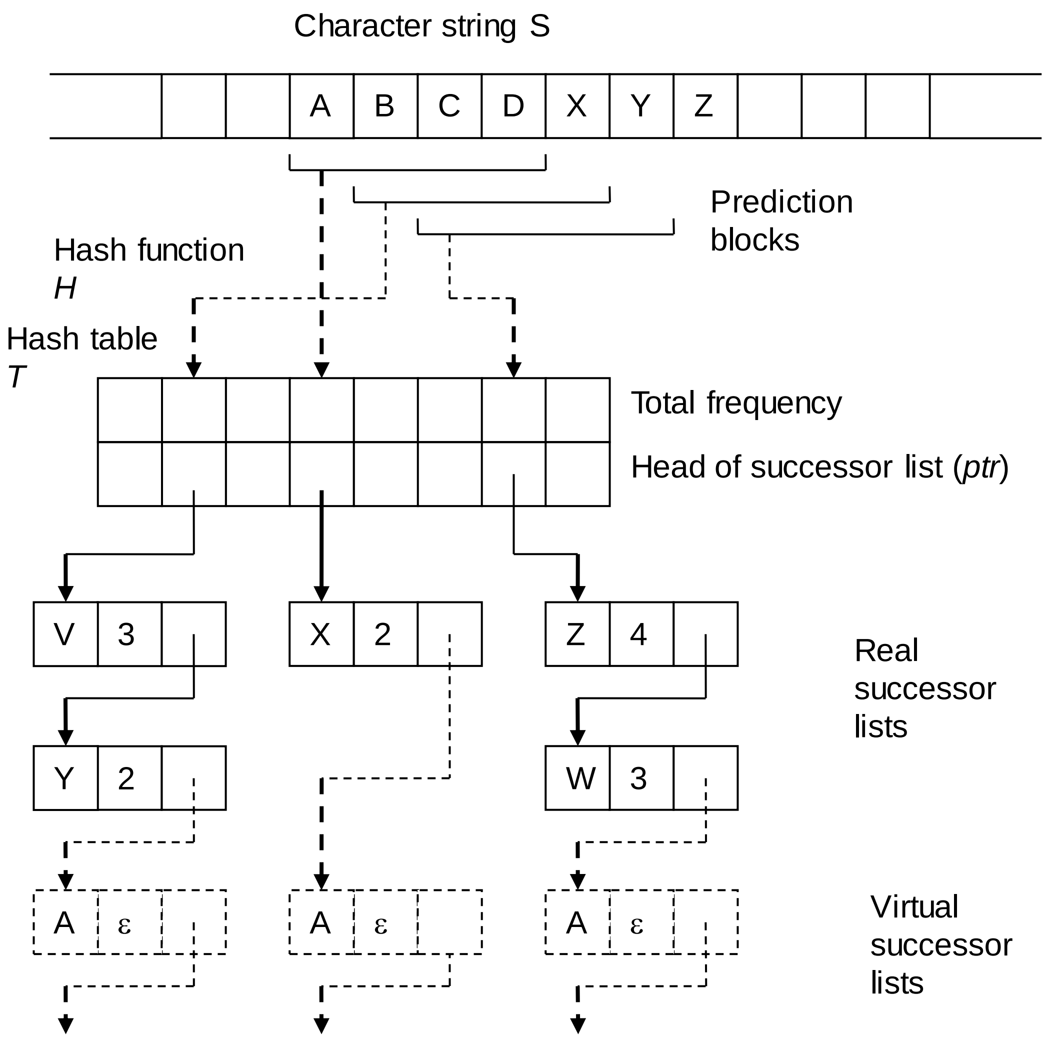


C. Statistics-based prediction of successors

Algorithm 5.3. Statistics-based coding of successors using fixed-length contexts.
Input: Message $X = x_1x_2 \ldots x_n$, context length k, alphabet size q, hash table size m.
Output: Encoded message, consisting of the first k symbols and an arithmetic code.

begin
    for i := 0 **to** m–1 **do**
    begin $T[i].head$:= NIL; $T[i].total$:= $\varepsilon \cdot q$;
    Send symbols $x_1, x_2, \ldots, x_k$ as such to the decoder
    Initialize arithmetic coder
    for i := k+1 **to** n **do**
    begin
        $addr := H(x_{i-k} \ldots x_{i-1})$
        if x_i is in list $T[addr].head$ (node N)
        then F := sum of frequencies of symbols in list $T[addr].head$ before ***N.***
        else   **begin**
                F := sum of frequencies of real symbols in list L headed by $T[addr].head.$
                $F := F + \varepsilon \cdot$(order number of x_i in the alphabet, ignoring symbols in list L)
                Add a node N for x_i into list L, with $N.freq = \varepsilon$.
                end
        Apply arithmetic coding to the cumulative probability interval
                $[F \,/\, T[i].total),\ (F+N.freq) \,/\, T[i].total)$
        $T[i].total := T[i].total + 1$
        N.freq := $N.freq + 1$
    end /* of for i := … */
    Finalize arithmetic coding
end

Statistics-based prediction of successors: Data structure



5.2. Dynamic-context predictive compression (Ross Williams, 1988)

Idea:

- Predict on the basis of the *longest* context that has occurred before.
- Context lengths grow during adaptive compression.

Problems:

- How to store observed contexts?
- How long contexts should we store?
- When is a context considered reliable for prediction?
- How to solve failures in prediction?

Dynamic-context predictive compression (cont.)

Data structure:

- Trie, where paths represent *backward* contexts
- Nodes store frequencies of context successors
- Growth of the trie is controlled

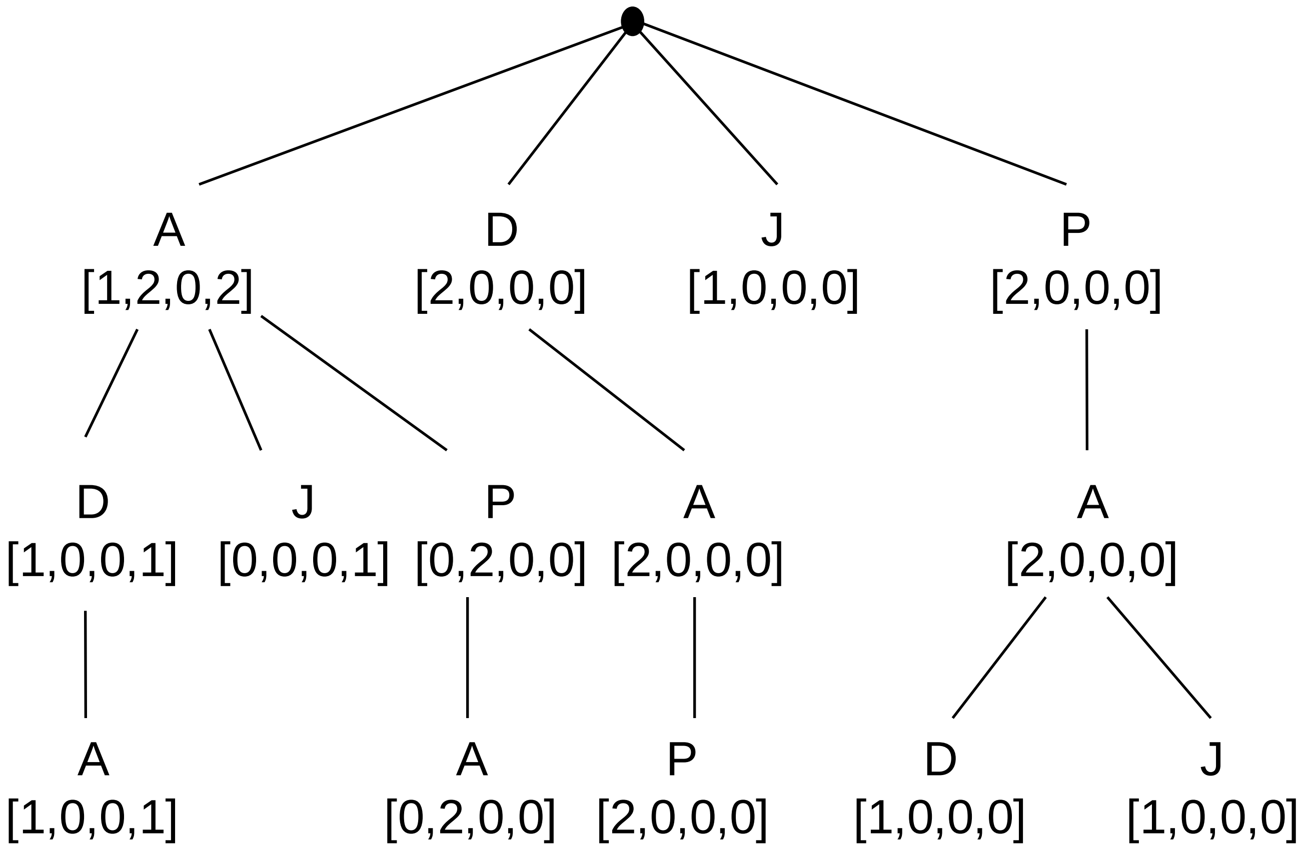
Parameters:

- Extensibility threshold ($et \in [2, \infty)$)
- Maximum depth (m)
- Maximum number of nodes (z)
- Credibility threshold ($ct \in [1, \infty)$)

Zero frequency problem:

- Probability of a symbol with x occurrences out of y: $\xi(x, y) = \dfrac{qx + 1}{q(y + 1)}$

Dynamic-context predictive compression: Trie for “JAPADAPADAA ...”



Using the previous trie

- Assumed continuation: “JAPADAPADAA | DA …”
- Parameters: q=4, ct=1
- Successor ‘D’:
  - Longest downward path in the trie: A[1,2,0,2] which is credible
  - Successor prob’s: P(‘A’)=5/24, P(‘D’)=P(‘P’)=9/24, P(‘J’)=1/24
  - Inf(‘D’) = $-\log_2(9/24) \approx 1.415$ bits
  - Node update: A[1,2,0,2]→A[1,3,0,2]
  - Insert new node: A-A[0,1,0,0]
- Successor ‘A’:
  - Longest credible path: D-A[2,0,0,0]
  - Probability of successor ‘A’ = 9/12, Inf(‘A’) = $-\log_2(3/4) \approx 0.415$ bits
  - Node updates: D[2,0,0,0]→D[3,0,0,0], D-A[2,0,0,0] → D-A[3,0,0,0], Insert new node D-A-A[1,0,0,0]

Dynamic-context predictive compression: The algorithm

Algorithm 5.4. Dynamic-context predictive compression.
Input: Message $X = x_1 x_2 \ldots x_n$, parameters *et*, *m*, *z*, and *ct*.
Output: Encoded message.

begin
  Create(*root*); *nodes* := 1;
  Initialize arithmetic coder
  for *i* := 1 **to** *q* **do** *root.freq*[*i*] := 0
  for *i* := 1 **to** *n* **do**
  begin
    current := *root*; *depth* := 0
    next := *current.child*[x_{i-1}] /* Assume a fictitious symbol *x*0 */
    while *depth* < *m* **and** *next* $\neq$ NIL **cand** *next.freq* $\geq$ *ct* **do**
    begin
      current := *next*
      depth := *depth* + 1
      next := *current.child*[$x_{i-depth-1}$]
    end
    arith_encode(ξ(*current.cumfreq*[x_{i-1}], *current.freqsum*),
      ξ(*current.cumfreq*[x_i], *current.freqsum*))

Dynamic-context predictive compression: The algorithm (cont.)

{Start to update the trie }

next := *root*; *depth* := 0

while *next* $\neq$ NIL **do**

begin

  current := *next*

  current.freq[x_i] := *current.freq*[x_i] + 1

  depth := *depth* + 1

  next := *current*.*child*[$x_{i-depth}$]

end

/* Continues … */

Dynamic-context predictive compression: The algorithm (cont.)

```
        /* Study the possibility of extending the trie */
        if depth < m and nodes < z and current.freqsum ≥ et
        then begin
              new(newnode)
              for j := 1 to q do
              begin
                 newnode.freq[j] := 0
                 newcode.child[j]
              end
              current.child[x_{i-depth}] := newnode
              newnode.freq[x_i] := 1
              nodes := nodes + 1
        end
    end
    Finalize arithmetic coder
end
```

Test results

Text type	Source size	Bits per symbol
English text (Latex)	39 836	3.164
Dictionary	201 039	4.081
Pascal program	20 933	2.212

- The results are rather good, but not the best possible.
- Reason: only the longest credible contexts are used; if prediction fails, the shorter contexts could succeed.