L6
Details: Training and Cracking

Sudhir Aggarwal and Shiva Houshmand
and Randy Flood
Florida State University
Department of Computer Science
E-Crime Investigative Technologies Lab
Tallahassee, Florida 32306

August 5-7, 2015

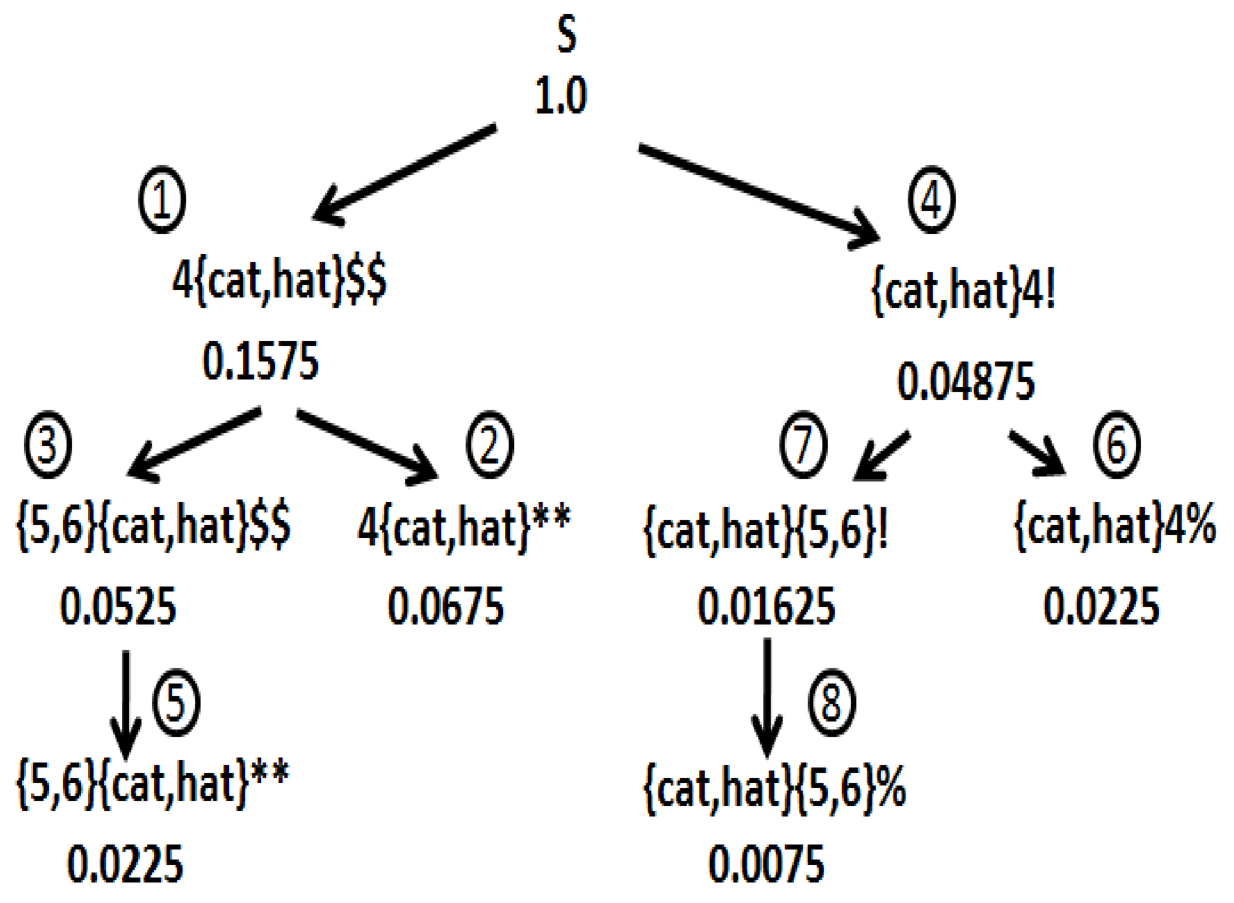
The Next Function

- Generates all possible different probability values of terminals for a given base structure without any duplication.
- A child node will never have probability higher than its parent.
- In order to generate terminals in probability order: A child node should never be popped from the priority queue before all its parents have been pushed into the queue.

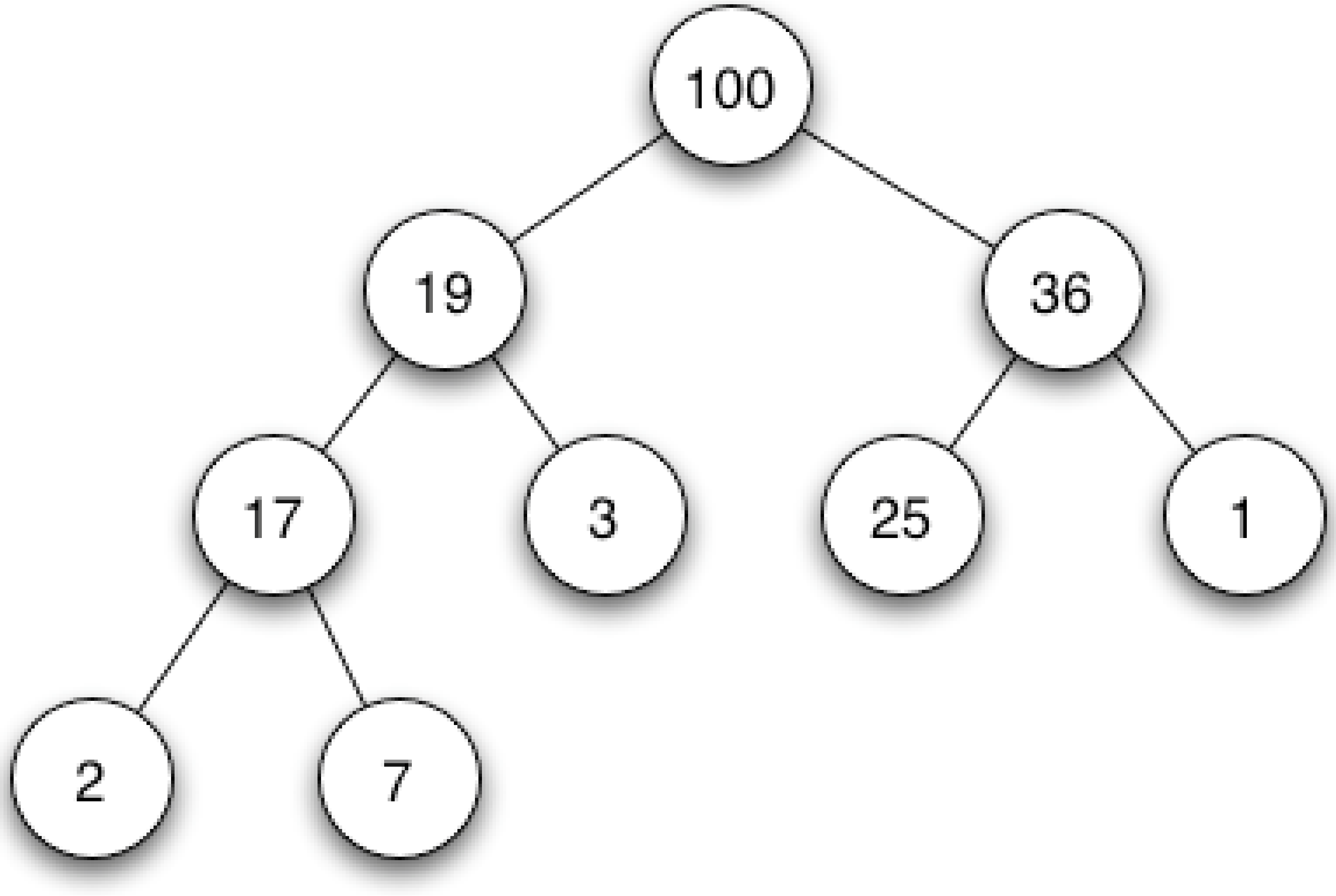
The Pivot Next Function

- We needed efficient next function algorithms to generate guesses in probabilistic order. Our first function was called a pivot function. Basically we limited which node would create children.

- Note that the structure to the right in not a priority queue!



Priority Queue max heap



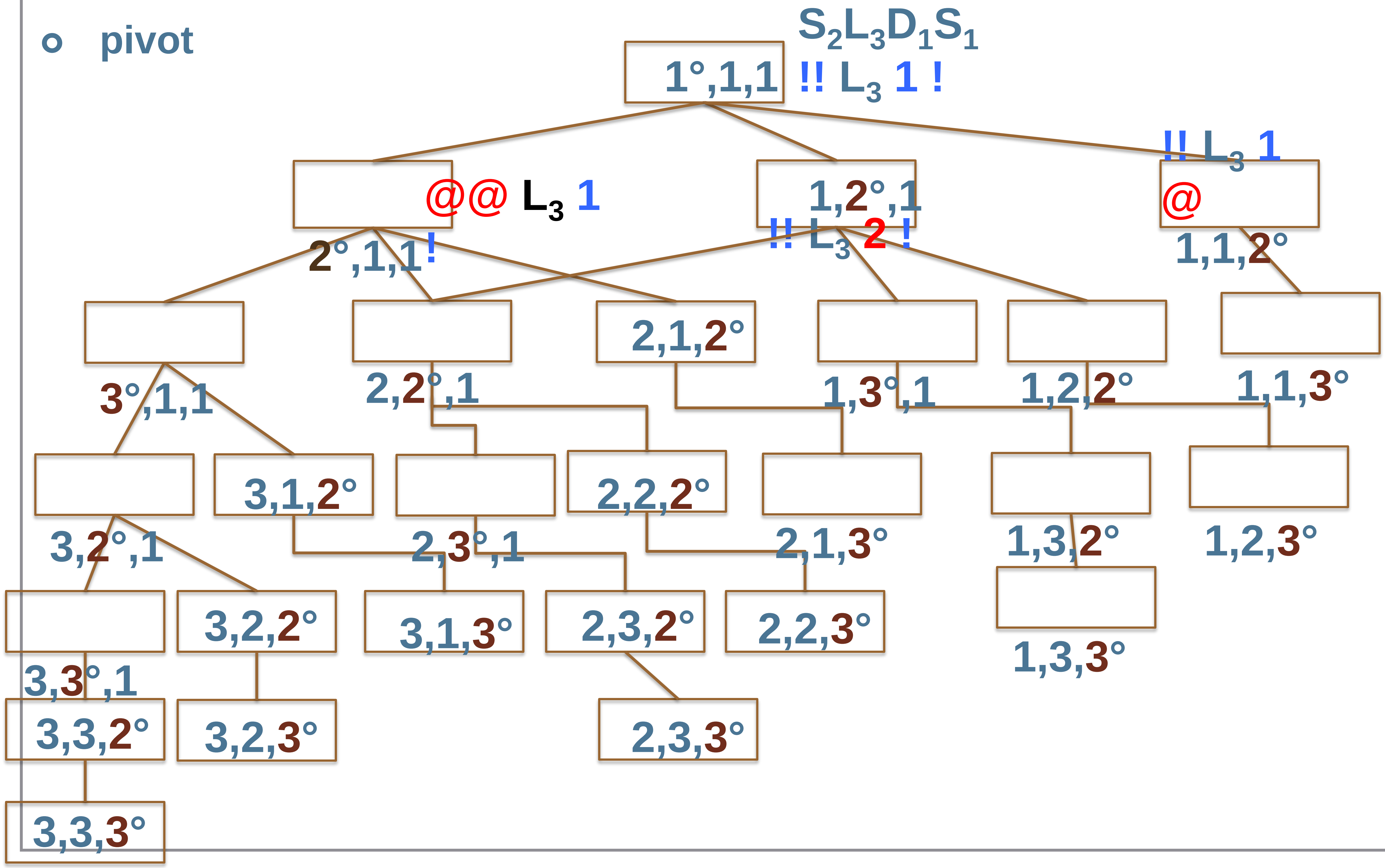
Operations: Insert, Maximum, Extract-Max, Increase Key
Complexity of these operations?

The “Next” Function

- The pivot value (or position) is an index value of a component starting from left to right in the node: it helps determine which new pre-terminal structures should be inserted into the priority queue next.
- Goal: create children pre-terminal structures in a systematic way, without creating duplicates. Need only insert 1 level descendants for each node popped as each child has smaller or equal probability to the parent in one component based on the pivot position.
- A node need only push those children nodes whose components change in the node’s pivot position or greater.

The “Next” Function

- pivot



Generating Guesses in probability order

$S_2 \rightarrow$	!!	0.5
	@@	0.3
	##	0.2

$D_1 \rightarrow$	1	0.45
	2	0.3
	3	0.25

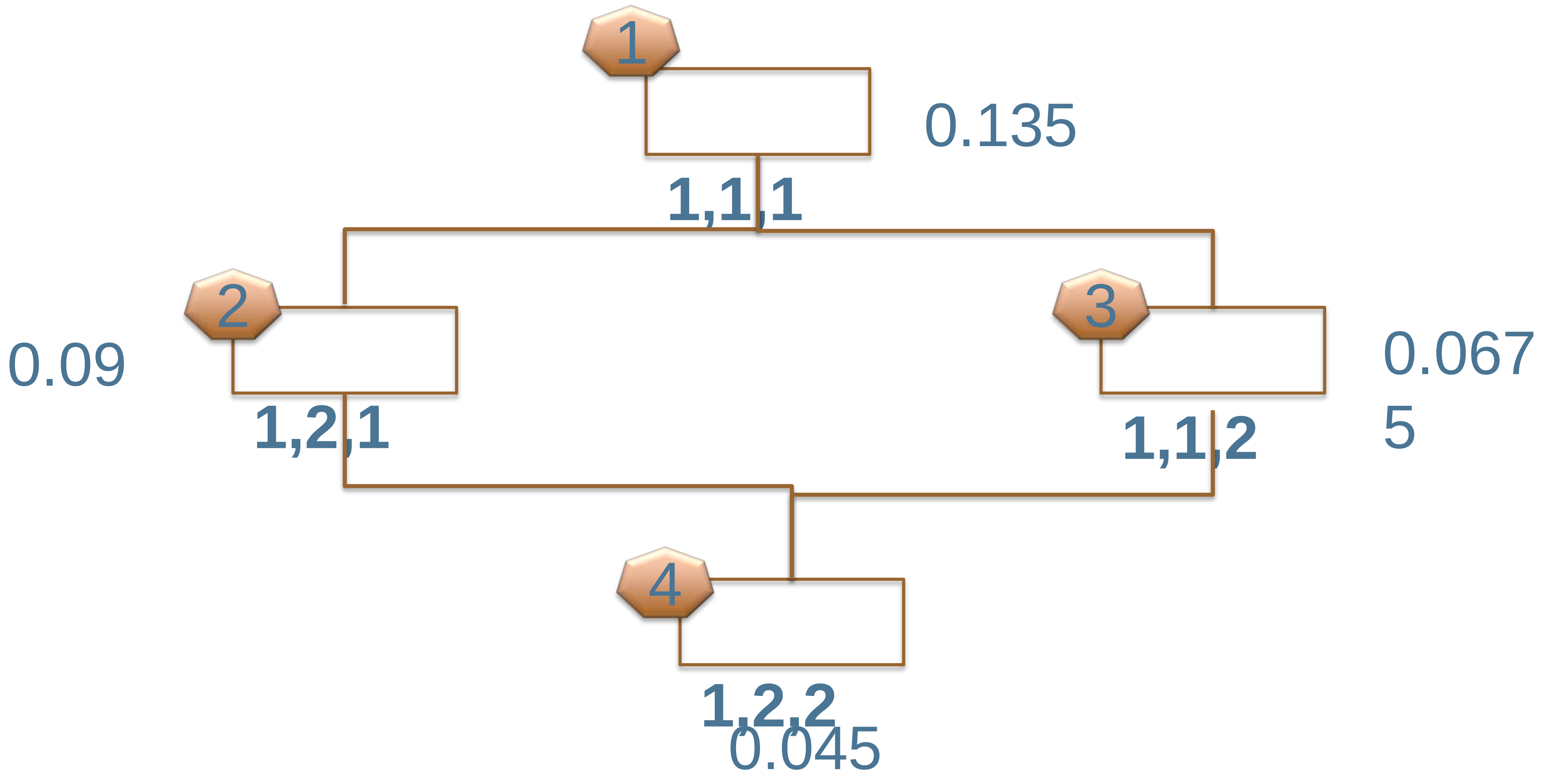
$S_1 \rightarrow$	!	0.6
	@	0.3
	#	0.1

Consider base structure $S_2L_3D_1S_1$

!!$L_3$1!	0.135
!! $L_3$2!	0.09
@@$L_3$1!	0.081
!!$L_3$1@	0.0675

- Push the highest probability pre-terminal into the queue: !! l_3 1!
- Pop the top value from the priority queue and print the guesses :
- !! cat1! , !!dog1!
- Create children of popped: (@@ l_3 1 !), (!! l_3 2 !), (!! L_3 1 @) and push them into the priority queue.
- Pop the next top value.
- Continue until queue is empty

Deadbeat dad algorithm



When node 1 is popped nodes 2,3 are pushed. In the previous Next algorithm, when 2 is popped, its child node 4 is pushed. In the deadbeat dad algorithm however, 4 is not pushed since 2 knows there is another dad (3) responsible for 4 and therefore abandons 4 for 3 to take care of it.

Container

A structure to optimize computations related to a set of terminals of similar type that all have identical probabilities.

$D_3 \rightarrow$	123	0.37
$D_3 \rightarrow$	222	0.33
$D_3 \rightarrow$	987	0.06
$D_3 \rightarrow$	451	0.04
$D_3 \rightarrow$	006	0.04
$D_3 \rightarrow$	584	0.04
$D_3 \rightarrow$	392	0.04
$D_3 \rightarrow$	943	0.04
$D_3 \rightarrow$	144	0.03
$D_3 \rightarrow$	155	0.01

451
006
584
392
943

Prob = 0.04

bird
pass
time
ball
tree
wind

Prob = 0.01

The Cracking Code

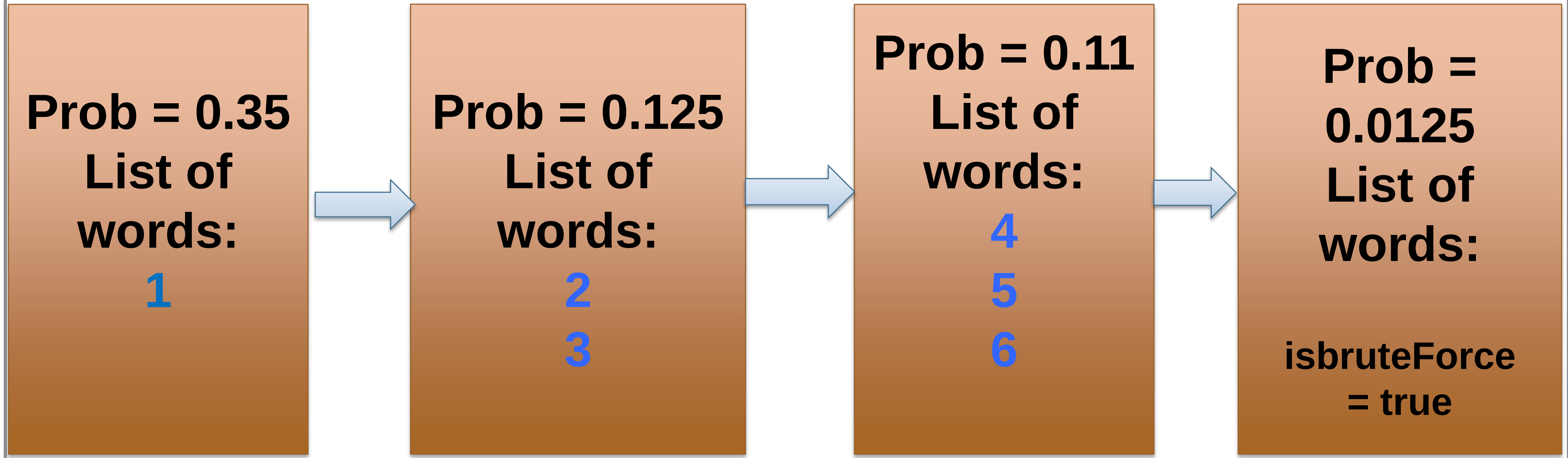
ntContainerType

```
typedef struct ntContainerStruct {
list <string> word;
double probability;    //the probability of this group
bool isBruteForce;
int bruteForceType;
//1=digits, 2=special, 3=letters
int bruteForceSize;
ntContainerStruct *next;
ntContainerStruct *prev;
}ntContainerType;
```

Type	Name
List of string	Word
Double	Probability
Bool	isBruteForce
Int	bruteForceType
Int	bruteforceSize

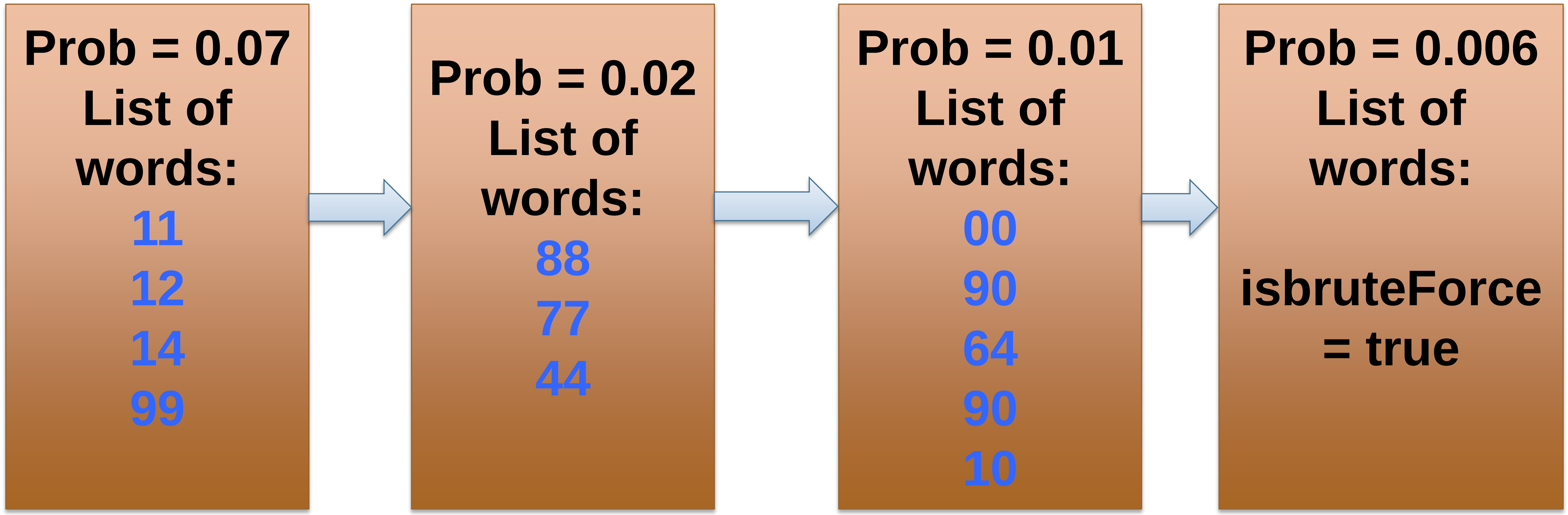
ntContainerType

- numWords[1] 1: length of the digits



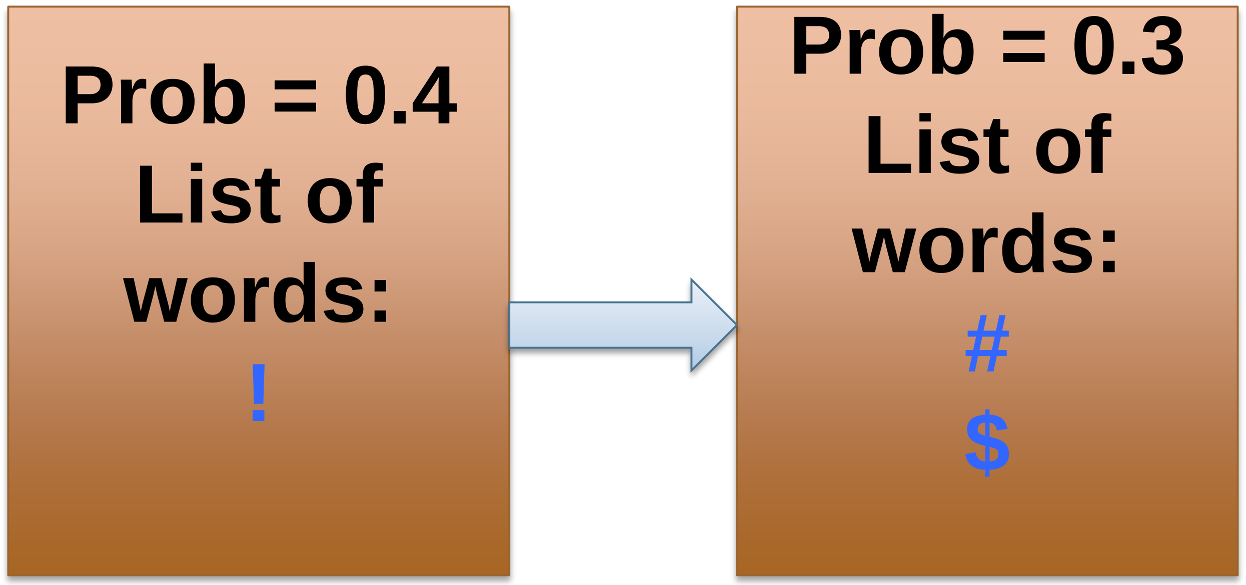
ntContainerType

- numWords[2] 2: length of the digits



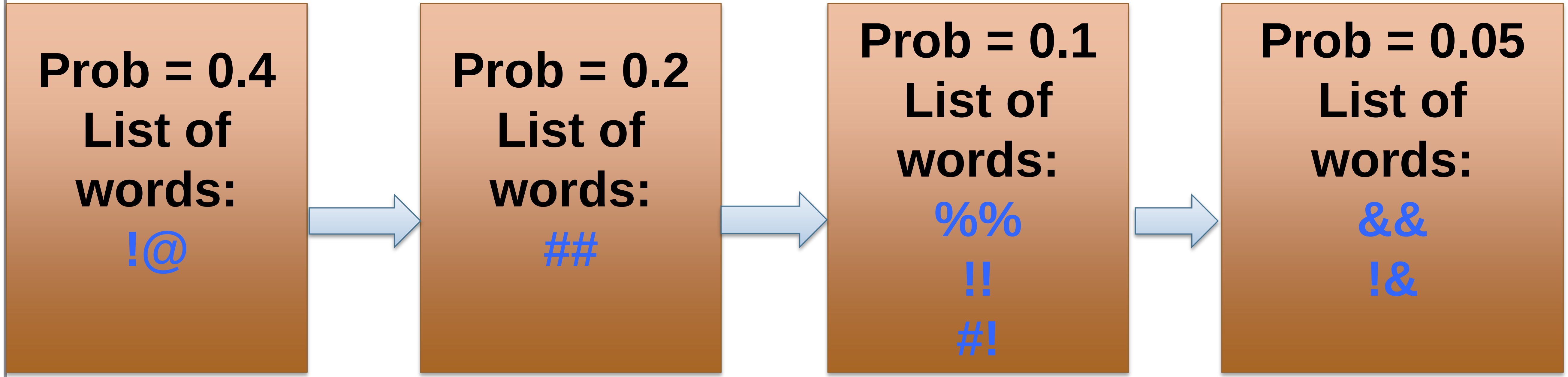
processProbFromFile (specialWords, Special)

- $S_1 \rightarrow$! 0.4 | # 0.3 | $ 0.3
- specialWords[1]:



processProbFromFile (specialWords, Special)

- $S_2 \rightarrow$!@ 0.4 | ## 0.2 | %% 0.1 | !! 0.1 | #! 0.1 | && 0.05 | !& 0.05
- specialWords[2]:



Cracker code

processBasicStruct()

- Read in all the base structures
- Pushes the highest probability pre-terminal into the queue
- The data structure used for this is pqReplacementType

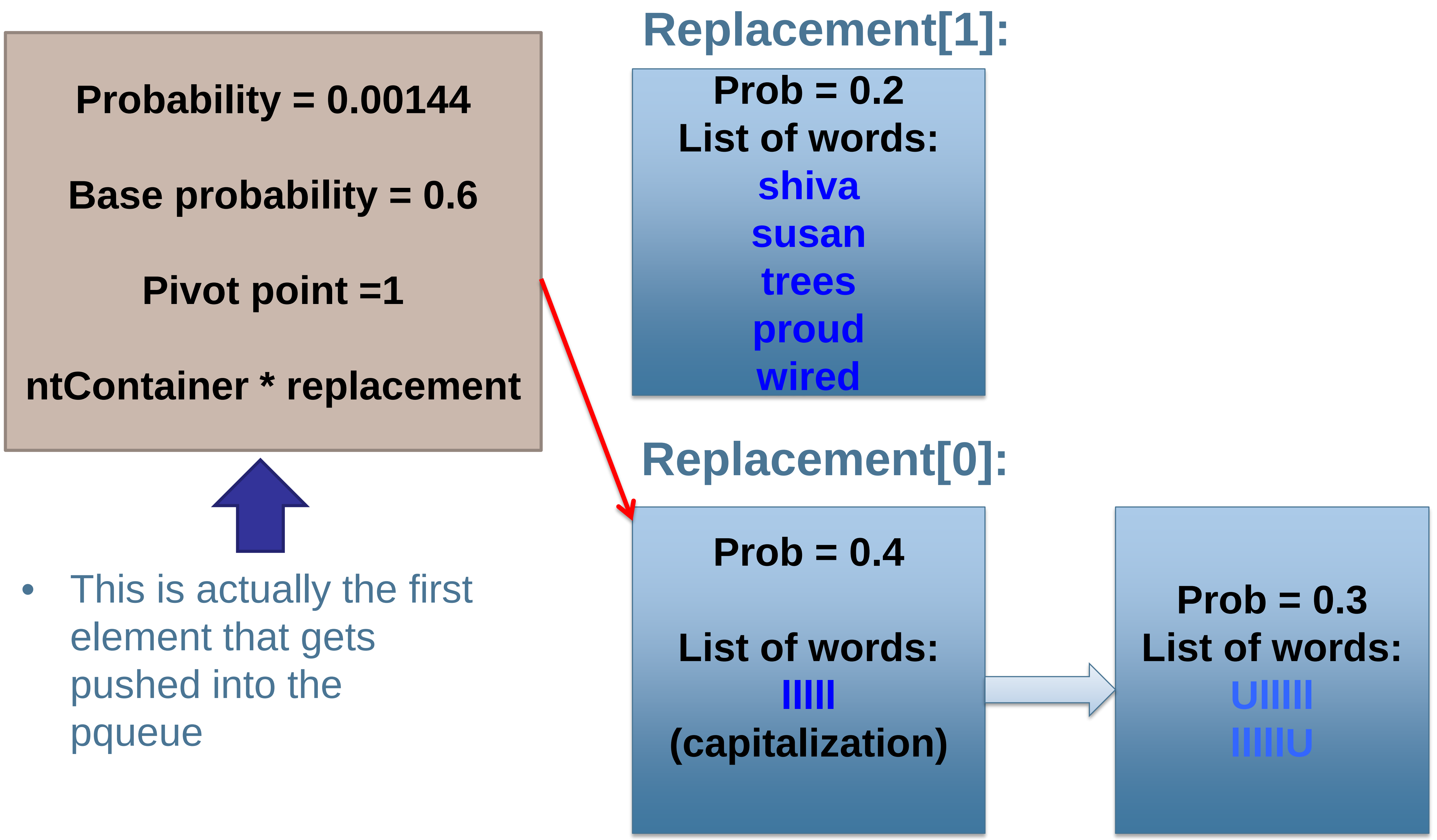
pqReplacementType

```
typedef struct pqReplacementStruct {
    double probability;      //preterminal
    double base probability;  //base structure
    int pivotPoint;
    deque <ntContainerStruct *> replacement;
}pqReplacementType;
```

pqReplacementType

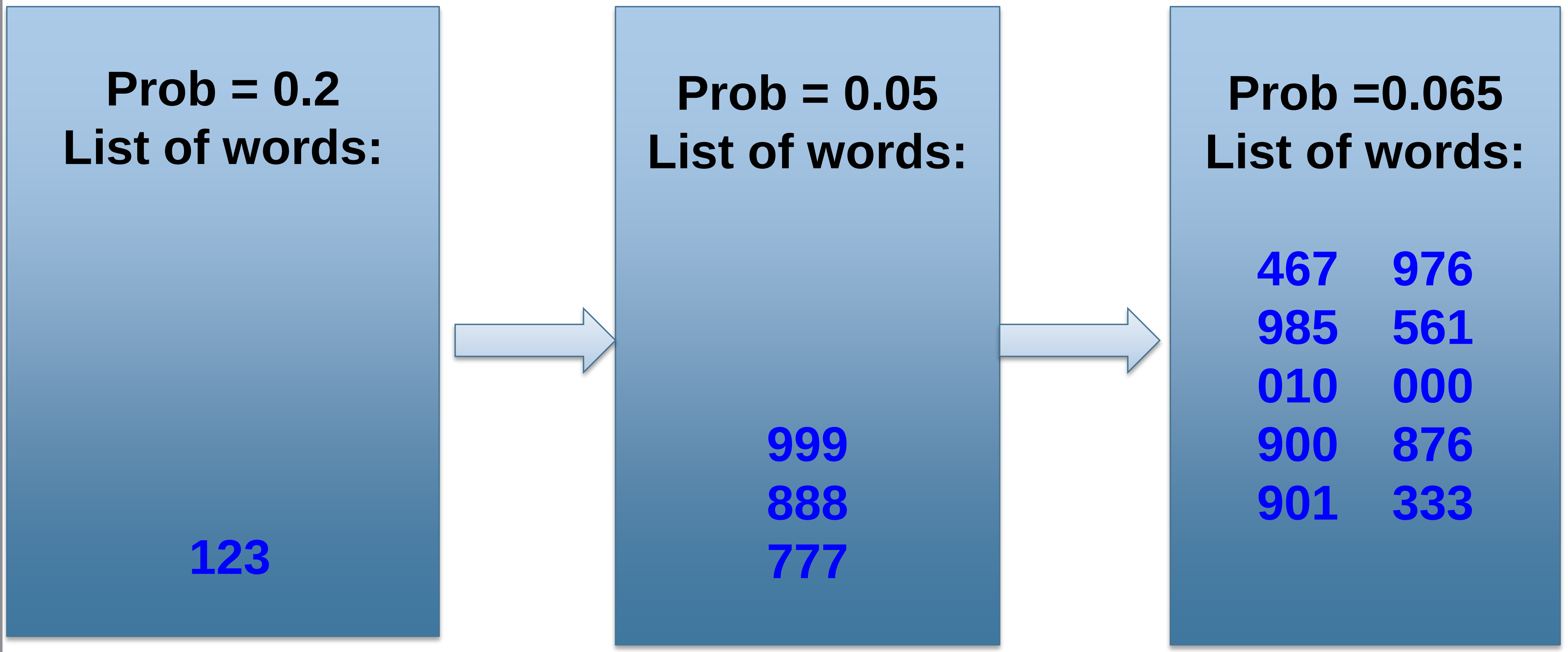
Type	Name
Double	probability
Double	Base probability
Int	pivotPoint
Deque <ntContainerStruct *>	replacement

pqReplacementType: example $L_5D_3S_1$ with probability 0.6

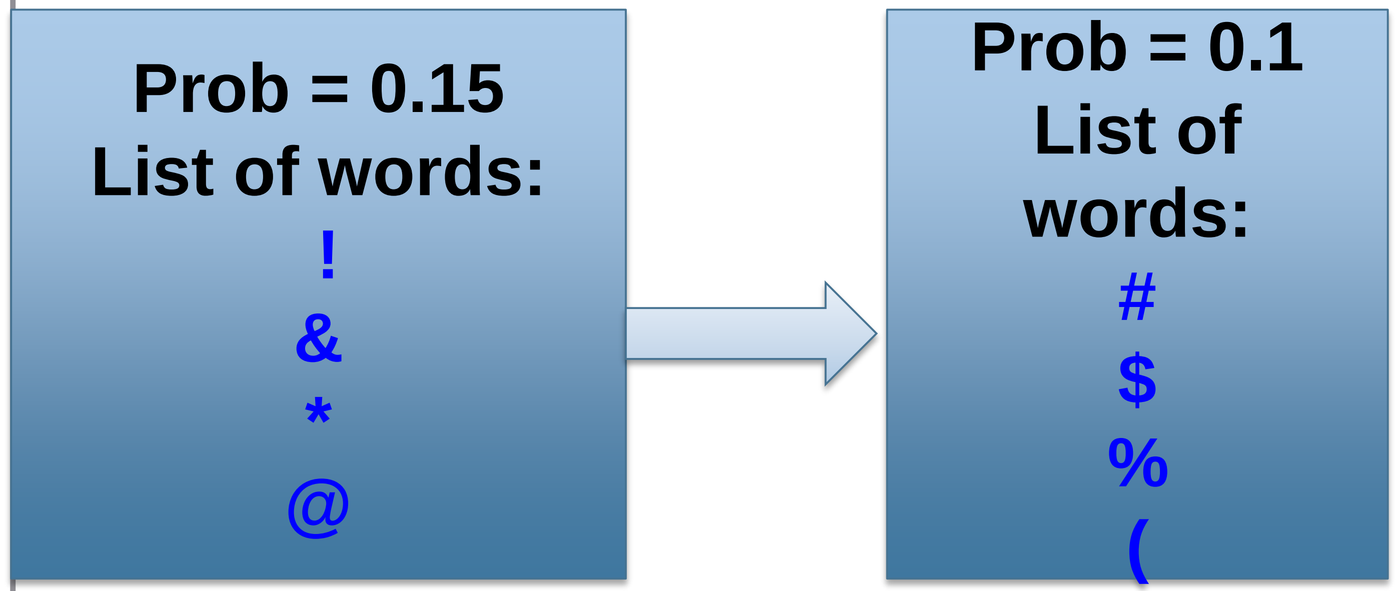


- This is actually the first element that gets pushed into the pqueue

Replacement[2]:



Replacement[3]:



Cracker Code

GenerateGuesses()

- pqueue->pop();
- createTerminal(); print the actual guesses for this pre-terminal
- pushDeadbeat();

The Training Code

Arrays of ItemInfo

- public class ItemInfo {

```
        public String value;
        public int number;
        public double percentage;
        public int length;
       }
```

Some Arrays

- grammarArray: contains the base structures
- KeyboardShapeArray: contains “rrr” stuff
  KeyboardPatternArray: “qwerty” and such
  DigitArray
  SpecialArray
  MultiwordArray
  DoubleWordArray
  CapArray