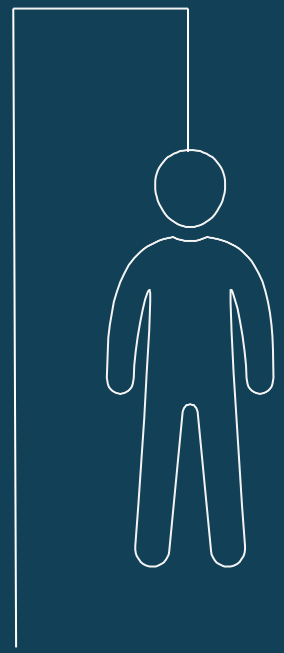
CSE 2010 Term Project
Hangman Player



04/23/19
Section: E4
Team: Fantastic For (i=0; i<4; i++)
Sarah Arends, Alejandra Escobar, Michael Hon, Josias Moupke

Initialize data structures from a dictionary file, formulate guesses for letters in an unknown word, and refine search process in response to feedback.

0

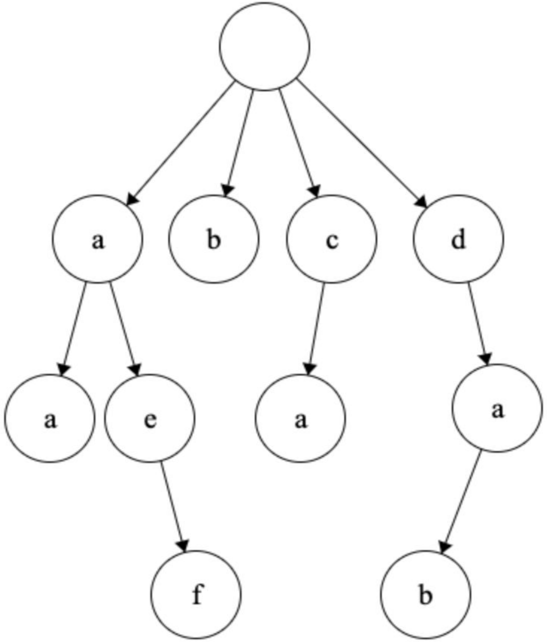
Previous Methods

Initial Method Using a Trie

Previous Data Structure

```
typedef struct anode {
   char letter;
   bool is_candidate,
        end_of_word;
   byte depth;
   struct anode *parent;
   SLList *children; // 26 children at most

} ANode;
```



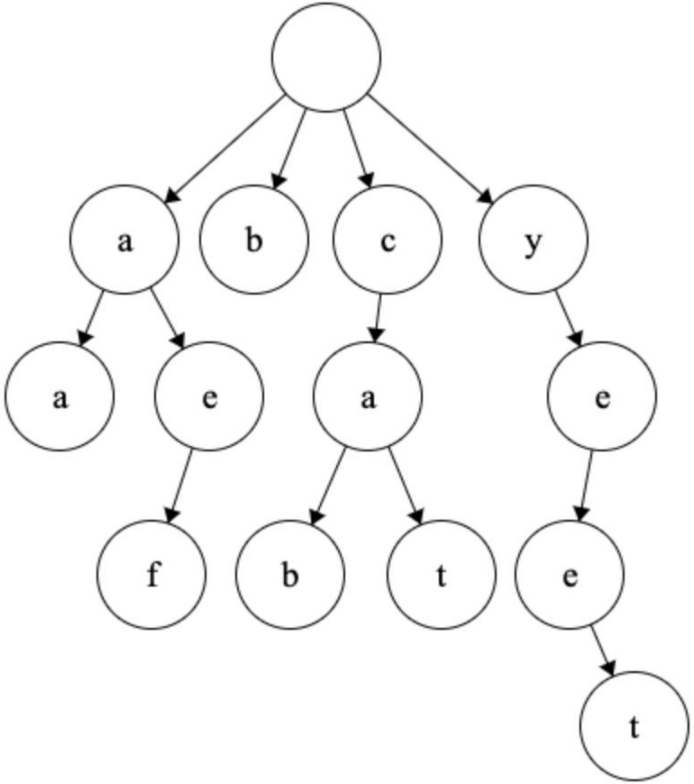
Previous Algorithm

1 **Search**

- Depth corresponds to position of letter in word

2 **Prune**

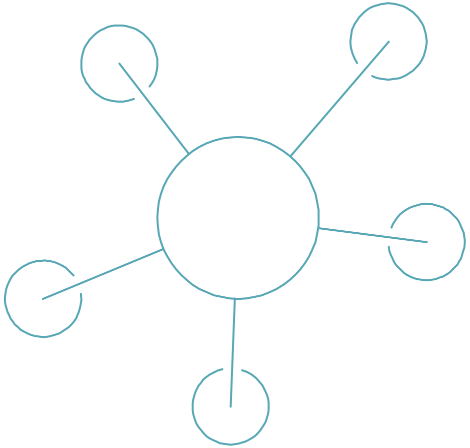
- Each node has a candidate flag
- Based on feedback from guessing prune the tree



Final Approach

Our improved algorithm & data structures

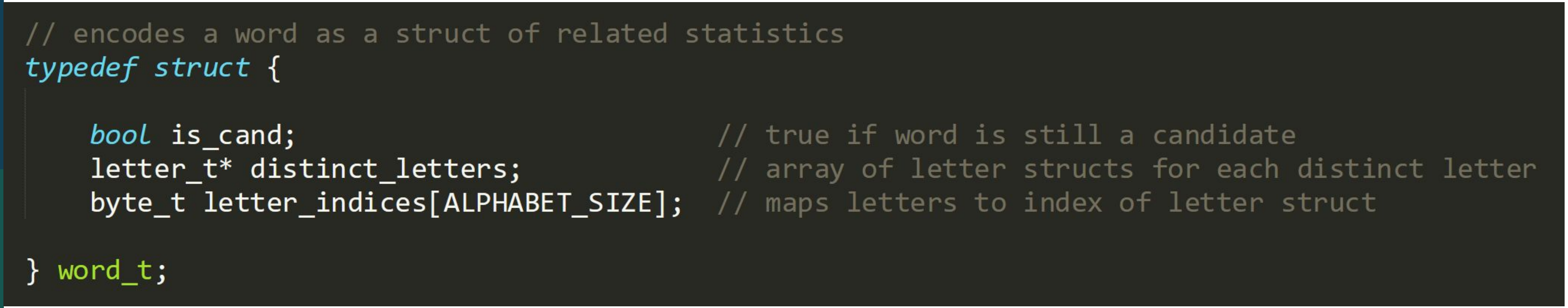
1



Data Structures

The following data structures were implemented in the final project

Word Struct: Candidate Flag & Information on Letters

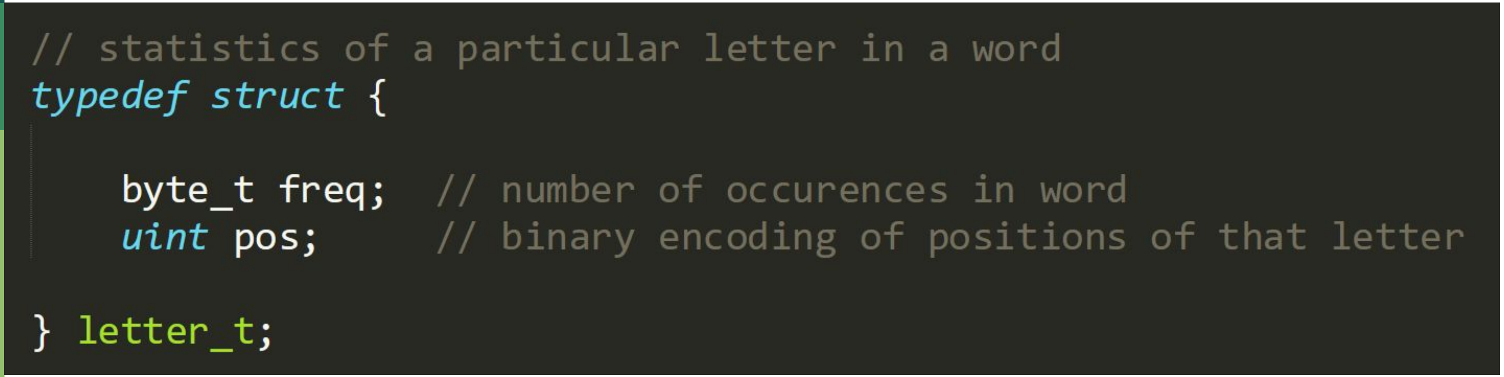


```
// encodes a word as a struct of related statistics
typedef struct {

    bool is_cand;                           // true if word is still a candidate
    letter_t* distinct_letters;             // array of letter structs for each distinct letter
    byte_t letter_indices[ALPHABET_SIZE];   // maps letters to index of letter struct

} word_t;
```

Letter Struct: Frequency and Position of Letter in Word



```
// statistics of a particular letter in a word
typedef struct {

    byte_t freq;  // number of occurences in word
    uint pos;     // binary encoding of positions of that letter

} letter_t;
```

Header Files

Hangman.h

- Declarations of letter, word, and wordlist types
- Constituent functions for implementation of “init”, “guess” and “feedback” operations
    - numDistinctLetters()
    - highestG

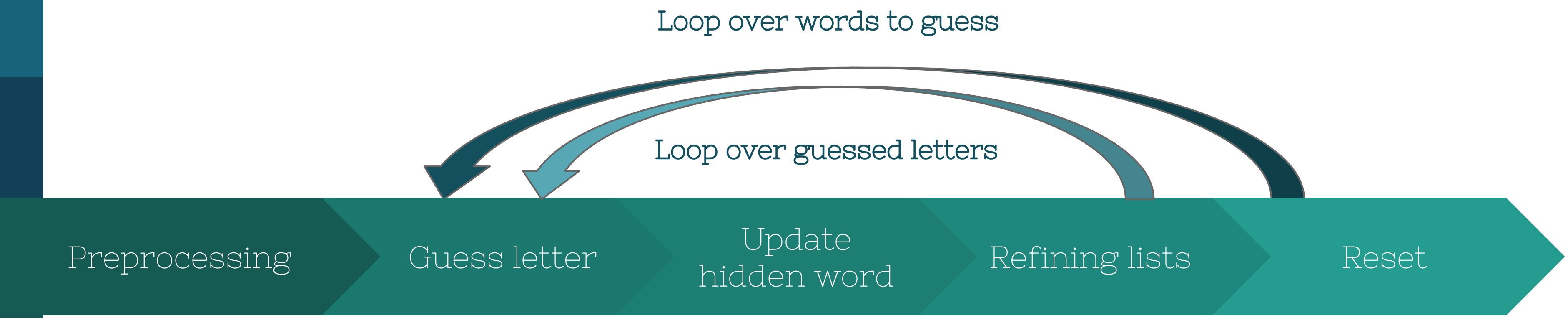
DLLlist.h

- Functions for implementation of doubly linked list
- pushfront(), pushback()
- removeFromList()

2

Algorithm Implemented

The following slides outline the algorithm used in the final submission



Loop over words to guess

Loop over guessed letters

Preprocessing

Guess letter

Update hidden word

Refining lists

Reset

Preprocessing

Initialize lists from dictionary

Linked lists store words of same length

Each word_t has frequency and positions for each distinct letter

Initially all candidates

Guess letter

Initialize guessed letter list & letter frequency list

Count how many words contain a given letter

Guess letter with highest frequency

Mark letter as guessed

Update hidden word

Receive feedback from evaluation program

Refining lists

If correct guess…

Find instances in updated hidden word

Flag words that don’t match new pattern

If incorrect guess…

Flag words with guessed letter

Always push candidates to front

Reset

Resets all words to candidates

Mark all letters not guessed, reset frequencies

Get new hidden word from evaluation program

3

Optimization

Some modifications used to optimize performance of the program

Space Optimization

Compressing Letter Data

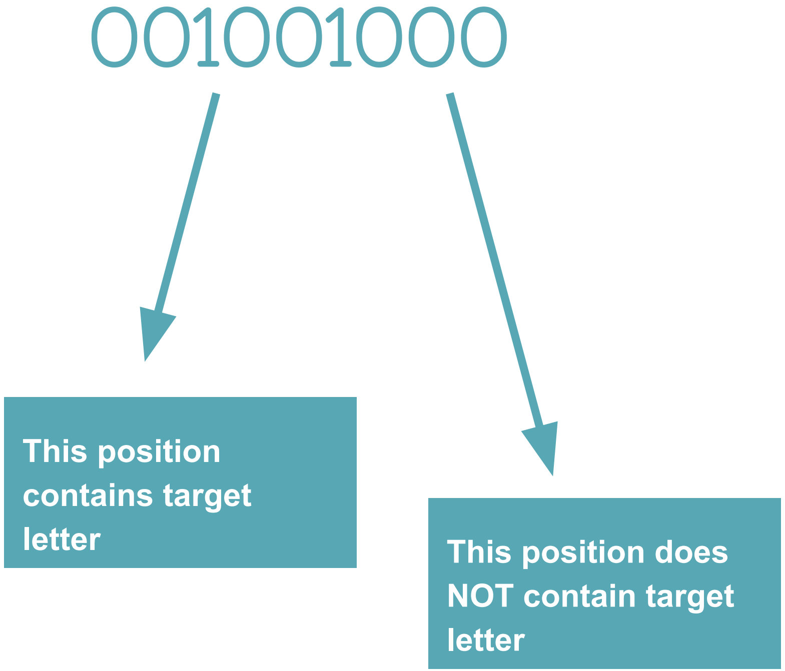
- 26 letter structs is inefficient
- Struct for N distinct letters only
- Map letters to structs
  - Index points to corresponding struct
  - Has value NONE = 255 if no struct, i.e. not in word

1	26 letter structs	● Each struct has uint and unsigned char ● 26 x 5 bytes
2	26 indices map to N structs	● 26 x 1 bytes ● N x 5 bytes

Space Optimization

Binary Position Encoding

- Used bits for position(s) of letter in the word
- Space efficient, storing bits instead of integers
- Note: Optimizes time for comparison, bit operations are constant



001001000

This position contains target letter

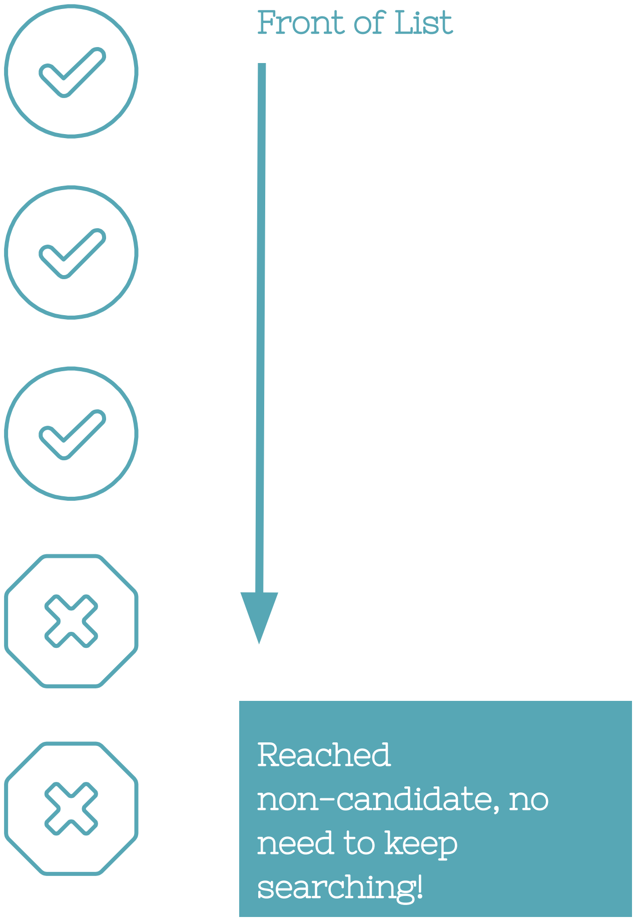
This position does NOT contain target letter

Time Optimization

List Refinement

- Candidates at front of list
- Non-candidates at back of list

- Some candidates marked bad during feedback process
- When non-candidates reached, STOP traversal



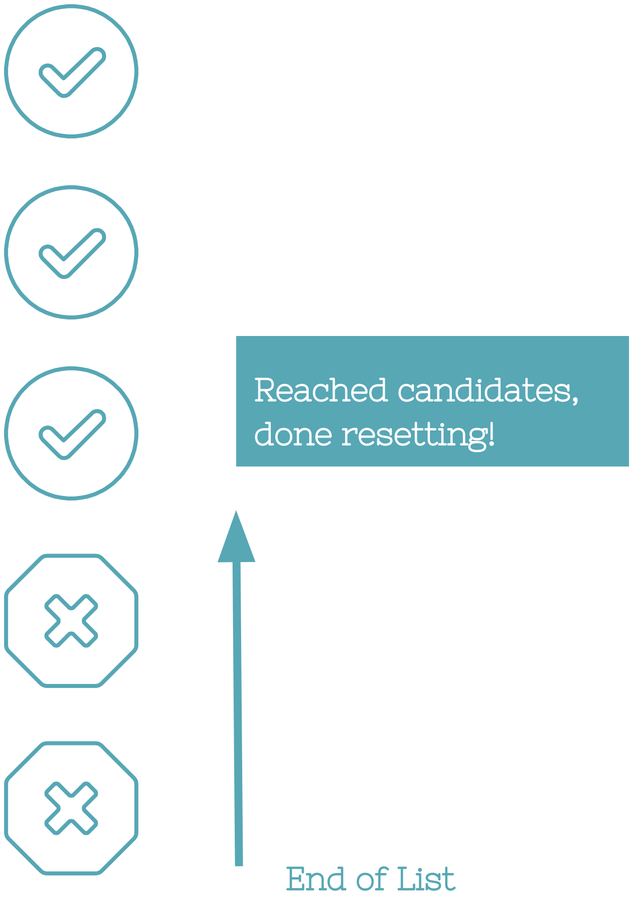
Front of List

Reached non-candidate, no need to keep searching!

Time Optimization

List Resetting

- Begin traversal from end of list, resetting nodes as candidates
- STOP traversal when candidate node reached
  - All preceding nodes already candidates



Reached candidates, done resetting!

3

Analysis

Evaluation

Hidden word file	Accuracy	Time per guess	Memory in bytes
hiddenWord1.txt	72.8500	1.5223e-03	46956544
hiddenWord2.txt	70.1333	1.6064e-03	

Comparing Submissions

	Accuracy	Time per guess	Memory in bytes
Initial Submission	Did not correctly determine most frequent letter; < 10	All words in one trie DFS traversal for frequency analysis was time consuming, called multiple times	Linked structure had node for every letter: ~200,000 words ⇒ ~700,000 nodes
Final Submission	Proper implementation of highestFreqLetter; ~70	Lists separated by word length refines candidate list Determine if word contains letter in constant time	Dynamically allocated array of structs Reduced space by storing encoded attributes of words

Further Improvements

- Use arrays instead of linked structure, no need to save pointers to nodes ⇒ less memory
- Further use of bitwise operations ⇒ save time and memory
- Diversify frequency of guessed letter to prevent accumulating common substrings ⇒ improves accuracy

THANK YOU!

Any questions?