CS4100 - Project 1 Report

Jansen Craft & Kaleb Demaline

What regular expressions/tokens did you identify, and how does your program react to each one.

Macros

<table>
<tr><td>F</td><td>[a-zA-Z0-9_]</td></tr>
<tr><td>D</td><td>[0-9]</td></tr>
<tr><td>INT</td><td>"int"|"short"|"unsigned int"|"unsigned short"|"long"|"unsigned long"|"long long"|"unsigned long long"</td></tr>
<tr><td>FLOAT</td><td>"float"|"double"|"long double"</td></tr>
</table>

Regular Expressions and Reactions with Token names

<table>
<tr><th>Regular Expression</th><th>Reaction</th><th>Token Returned (If applicable)</th></tr>
<tr><td>\/\/.*\n</td><td>Increment linecount</td><td>no token – single-line comment</td></tr>
<tr><td>"/*"([^*]*|[*][^/])*"*/"</td><td>Add number of newlines to linecount</td><td>no token – multi-line comment</td></tr>
<tr><td>#include\ *<[a-z.]*></td><td>Do nothing</td><td>no token – include statement</td></tr>
<tr><td>#include\ *\"[a-z.]*\"</td><td>Do nothing</td><td>no token – include statement</td></tr>
<tr><td>\(</td><td>Return token for open parenthesis</td><td>TK_OPEN_P</td></tr>
<tr><td>\)</td><td>Return token for closed parenthesis</td><td>TK_CLOSED_P</td></tr>
<tr><td>\{</td><td>Return token for open curly brace</td><td>TK_OPEN_CB</td></tr>
<tr><td>\}</td><td>Return token for closed curly brace</td><td>TK_CLOSED_CB</td></tr>
<tr><td>\[</td><td>Return token for open square bracket</td><td>TK_OPEN_SB</td></tr>
<tr><td>\]</td><td>Return token for closed square bracket</td><td>TK_CLOSED_SB</td></tr>
<tr><td>\;</td><td>Return token for semicolon</td><td>TK_SEMICOLON</td></tr>
<tr><td>\,</td><td>Return token for comma</td><td>TK_COMMA</td></tr>
<tr><td>\"[^\"]*\"</td><td>Add number of newlines to linecount. Return token for string literal</td><td>TK_STRING_LITERAL</td></tr>
<tr><td>\'[^\']{1,2}\'</td><td>Return token for character literal</td><td>TK_CHAR_LITERAL</td></tr>
<tr><td>({D}*\.{D}*)|{D}+</td><td>Return token for number literal</td><td>TK_NUMBER_LITERAL</td></tr>
<tr><td>\+</td><td>Return token for plus</td><td>TK_PLUS</td></tr>
<tr><td>\-</td><td>Return token for minus</td><td>TK_MINUS</td></tr>
<tr><td>\!</td><td>Return token for exclamation</td><td>TK_EXCLAMATION</td></tr>
<tr><td>\~</td><td>Return token for tilde</td><td>TK_TILDA</td></tr>
<tr><td>\&</td><td>Return token for ampersand</td><td>TK_AMPERSAND</td></tr>
<tr><td>*</td><td>Return token for asterisk</td><td>TK_ASTERISK</td></tr>
<tr><td>\=</td><td>Return token for equal</td><td>TK_EQUAL</td></tr>
</table>

<table>
<tr><td>\/</td><td>Return token for forward slash</td><td>TK_FORWARD_SLASH</td></tr>
<tr><td>\%</td><td>Return token for percent</td><td>TK_PERCENT</td></tr>
<tr><td>\^</td><td>Return token for caret</td><td>TK_CARET</td></tr>
<tr><td>\></td><td>Return token for greater</td><td>TK_GREATER</td></tr>
<tr><td>\<</td><td>Return token for less</td><td>TK_LESS</td></tr>
<tr><td>\?</td><td>Return token for question</td><td>TK_QUESTION</td></tr>
<tr><td>\|</td><td>Return token for pipe</td><td>TK_PIPE</td></tr>
<tr><td>{INT}</td><td>Return token for int</td><td>TK_INT</td></tr>
<tr><td>{FLOAT}</td><td>Return token for float</td><td>TK_FLOAT</td></tr>
<tr><td>"void"</td><td>Return token for void</td><td>TK_VOID</td></tr>
<tr><td>"char"</td><td>Return token for char</td><td>TK_CHAR</td></tr>
<tr><td>"while"</td><td>Return token for while</td><td>TK_WHILE</td></tr>
<tr><td>"for"</td><td>Return token for for</td><td>TK_FOR</td></tr>
<tr><td>"do"</td><td>Return token for do</td><td>TK_DO</td></tr>
<tr><td>"if"</td><td>Return token for if</td><td>TK_IF</td></tr>
<tr><td>"else"</td><td>Return token for else</td><td>TK_ELSE</td></tr>
<tr><td>"switch"</td><td>Return token for switch</td><td>TK_SWITCH</td></tr>
<tr><td>"case"</td><td>Return token for case</td><td>TK_CASE</td></tr>
<tr><td>"default"</td><td>Return token for default</td><td>TK_DEFAULT</td></tr>
<tr><td>"break"</td><td>Return token for break</td><td>TK_BREAK</td></tr>
<tr><td>"continue"</td><td>Return token for continue</td><td>TK_CONTINUE</td></tr>
<tr><td>"return"</td><td>Return token for return</td><td>TK_RETURN</td></tr>
<tr><td>"goto"</td><td>Return token for goto</td><td>TK_GOTO</td></tr>
<tr><td>{F}+</td><td>Return token for identifier</td><td>TK_IDENTIFIER</td></tr>
<tr><td>[\t\n\f]</td><td>Add number of newlines to linecount</td><td>no token - whitespace</td></tr>
<tr><td>.</td><td>Do nothing</td><td>no token - unmatched</td></tr>
</table>

In the main of the lex program, we print out the number associated with the token (1-42) with width 2. We prepend the number with zeros if it is < 10.

A brief description in your own words of how you implemented the Winnowing algorithm

```
int main(){
  int k = 20, w = 20;
  hash<string> hasher;
  vector<string> filenames;
  vector<vector<size_t>> fingerprints;

  string filename, tokens;
```

```
while(cin >> filename && getline(cin, tokens)){
    filenames.push_back(filename);
    auto end = remove_if(tokens.begin(),tokens.end(),[](char c){return c == ' ';});
    tokens.erase(end,tokens.end());

    vector<string> k_grams;
    for (int i = 0; i <= tokens.size()-k; i++)
        k_grams.push_back(tokens.substr(i, k));

    auto fp = winnow(w,k_grams,hasher);
    fingerprints.push_back(fp);
}
```

In our main, we read in all the tokens of a file as one space-delimited string, and then remove all the whitespace. After this, we add the k_grams (substrings of this mega-string) to a vector. We have a hasher object from the stl library which maps strings to SIZE_T. We then pass the window size, the vector of k_grams, and the hasher object to out winnowing function.

```
vector<size_t> winnow (int w, const vector<string>& k_grams, hash<string>& hasher){
    deque<size_t> buffer(w, SIZE_T_MAX);
    vector<size_t> fingerprints;
    int min_hash_index = 0;

    // Load initial w-1 k grams
    for (int k_idx = 0; k_idx < w-1; k_idx++){
        buffer.push_back(hasher(k_grams[k_idx]));
        buffer.pop_front();
    }

    for (int k_idx = w; k_idx < k_grams.size(); k_idx++){
        buffer.push_back(hasher(k_grams[k_idx]));
        buffer.pop_front();
        min_hash_index--;

        if (min_hash_index == -1){
            min_hash_index = w-1;
            for(int i = w-1; i >= 0; i--)
                if (buffer[i] < buffer[min_hash_index]) min_hash_index = i;
            fingerprints.push_back(buffer[min_hash_index]);
        } else {
```

```
            if (buffer.back() <= buffer[min_hash_index]){
                min_hash_index = w-1;
                fingerprints.push_back(buffer[min_hash_index]);
            }
        }
    }
    return fingerprints;
}
```

In the winnowing function, we initialize a deque to w nodes with SIZE_T_MAX. We then load in the first w-1 hashed k_grams into the deque by adding to the back and popping off the front. After this, starting at the wth hash, we slide add the next hash, remove the last one, increment our min_hash_index and then check to see if we have a new smallest hash. Firstly, if our best hash was just popped off the front of buffer, we find the rightmost smallest hash and set that as our minimum hash, making sure to record the hash and location to a fingerprints vector. If our best hash is still in the buffer, we just check it against the newest hash. If the new one is smaller, we set that our min_hash_index and then append its information to the fingerprints vector which is returned at the end of the function.

```
// Compare Fingerprints
    for (int i = 0; i < fingerprints.size(); i++)
        sort(fingerprints[i].begin(), fingerprints[i].end());

    vector<pair<float, string>> similarity_scores;

    for (int i = 0; i < fingerprints.size(); i++){
        for (int j = i+1; j < fingerprints.size(); j++){
            vector<size_t> intersection;
            set_intersection(
                fingerprints[i].begin(), fingerprints[i].end(),
                fingerprints[j].begin(), fingerprints[j].end(),
                back_inserter(intersection)
            );
            similarity_scores.push_back({(float)intersection.size() / (float)fingerprints[i].size(), filenames[i] + ' ' + filenames[j]});
            intersection.clear();
        }
    }

    sort(similarity_scores.rbegin(), similarity_scores.rend());

    for (auto score: similarity_scores){
        cout << setprecision(3) << setfill('0') << fixed << score.first << "\t" << score.second << '\n';
    }
```

```
}
```

In the main, once we have the fingerprints vectors for each of the files passed to standard input, we start our pairwise comparison. For the set_interection from stl that we are using, we must first sort all the fingerprint vectors. After this, we can get the size of the intersection of fingerprint vectors of the files, divide that by the length of the file in the outer loop, and then use this as our percentage of common fingerprints found. This percentage is printed out for that comparison after the percentages are sorted at the end.

The results of your analysis, including identifying any submissions found by the algorithm that you believe may be plagiarism

```
1.000    bills_53.c bills_54.c
1.000    bills_09.c bills_54.c
1.000    bills_09.c bills_53.c
1.000    bills_03.c bills_54.c
1.000    bills_03.c bills_53.c
1.000    bills_03.c bills_09.c
0.971    bills_10.c bills_30.c
0.960    bills_47.c bills_48.c
0.904    bills_15.c bills_22.c
```

Here, at the top of our Plagiarism Report, we have the pairs of student submission that were most similar. The first thing that must be said is that this assignment was clearly very simple, and they were all given the same basic instructions on how to write the program. This makes it so most of the submissions are very similar by default. However, students 53, 54, 9, and 3 had the same fingerprints. This is so unlikely to happen naturally, that it should be safe to say that these students plagiarized. One could assign a cutoff of perhaps 90% similarity and could reasonably say these students cheated, but the safest bet is to at least talk to the students mentioned above.