明治大学理工学部情報科学科

ソフトウェア実習 レポート

8章 フィルター

提出日 2025 年 11 月 27 日

3 年 15組 91番 157R257501

SEO BEOMGYO

はじめに

本課題の目的は，5種類のフィルタプログラムを作成し，それらを UNIX のパイプで組み合わせることで，大規模な英語テキストから単語の出現頻度表を生成する処理系を構成することである。入力テキストから英字のみを抽出して正規化し，行単位でソートしたうえで重複行を集約し，さらにページ・列単位に整形するまでの一連の処理を，課題1～5を通して段階的に実装した。

課題1

課題1では，第8章で定義されている5つのフィルタのうち，最初の3つを実装し，教科書の例と同じ結果が得られることを確認することが求められた。1番目のフィルタ（1.c）は標準入力から文字を1つずつ読み，英字であればそのまま出力し，それ以外の文字は改行に置き換えることで「1行＝1単語」の形式に変換するようにした。

2番目のフィルタ（2.c）は，連続する改行を検出して空行を削除し，内容をもつ行のみを通過させる構成とした。

3番目のフィルタ（3.c）は英大文字だけを小文字に変換し，小文字とその他の文字は変更せずに出力することで，大文字小文字を区別しない単語列を得るようにした。これら3プログラムを prog1 | prog2 | prog3 の順に接続し，教科書の短い入力例に対して期待どおりの単語列が得られることを確認した。

課題2

課題2では，残りの2つのフィルタを追加実装し，5本すべてを組み合わせたパイプラインが仕様どおり動作することを確認することが目的であった。4番目のフィルタ（4.c）は，最大100万行の単語列を2次元配列に読み込み，先頭アドレスを格納したポインタ配列を qsort と strcmp により辞書順にソートするプログラムとした。

5番目のフィルタ（5.c）は，ソート済みで同一単語が連続して現れることを前提とし，現在の行と直前の行を比較しながら出現回数を数え，異なる単語に切り替わった時点で「回数 単語」という形式で1行出力することで，重複行の集約と頻度カウントを実現した。

prog1 | prog2 | prog3 | prog4 | prog5 というパイプラインを構成し，テキスト中の単語がアルファベット順に並び，各単語に正しい出現回数が付いていることを小さなテストデータで確認した。

課題3

課題3では，大文字と小文字を区別する仕様に変更するよう指示されているが，各フィルタプログラム自体は変更せず，組み合わせだけで対応することが求められた。本レポートでは，3番目のフィルタが大文字を小文字に統一する役割を担っていることに着目し，パイプラインから prog3 を取り除き，prog1 | prog2 | prog4 | prog5 という接続に変更した。

これにより，Apple と apple のような綴りが同じでも大文字小文字の違う単語が別々の行として集計されるようになり，仕様変更をフィルタ同士の接続順を変えるだけで達成できることを確認した。

課題4

課題4では，大規模テキストファイル shaks12.txt を用いて，作成したフィルタ群が大きな入力に対しても正しく動作するかを検証するとともに，実行時間を測定することが課題であった。./prog1 < shaks12.txt | ./prog2 | ./prog3 | ./prog4 | ./prog5 > result.txt として得られた出力と，配布された正解ファイル shaks12_result.txt とを diff -w で比較した結果，すべての行が一致し，特に単語 a の出現回数など頻度の大きい単語についても差異がないことを確認した。

さらに，パイプライン全体の前に time コマンドを付与して実行することで，real・user・sys の各時間を計測し，本課題で用いた単純な配列・qsort ベースの実装でも，数十万語規模のテキストを現実的な時間内に処理できることを確認した。

課題5（オプション）

課題5では，これまでに得られた単語と出現回数の一覧を，印刷しやすいように複数列・複数ページに整形する6番目のフィルタプログラムを作成することが求められた。6番目のフィルタ（6.c）は，まずすべての入力行を動的配列に読み込み，行の最大長から列幅を決定したうえで，「1ページあたりの行数」と「列数」をコマンドライン引数から受け取る構成とした。

ページごとに rows × cols 個までの行を取り出し，縦方向にインデックスをずらしながら再配置することで，各ページ内で上から下に列を埋め，次に右隣の列へ進むという順序で出力するレイアウトを実現した。教科書に示された one～nineteen の例については，./prog6 4 3 を実行することで，例と同じ 4 行×3 列の整形結果が得られることを確認した。

実装上の問題点と修正過程

本課題の実装では，とくに第4フィルタと大規模テキストに対する動作確認の段階で，いくつかの問題に直面した。最初に shaks12.txt を用いて頻度表を生成したところ，単語 a の出現回数が正解ファイルより少なく，ほかの頻出語についても一定の比率で値が小さくなるという現象が見られた。

原因調査のために，小さな入力で prog4 だけを単独実行し，読み込んだ行数を表示させる簡単なデバッグ用コードを一時的に挿入した。その結果，プログラムがファイル終端に達する前に読み込みを打ち切っており，配列 buf と line に対して設定した行数上限が shaks12.txt に含まれる単語総数よりも小さいことが判明した。

この問題に対しては，MAX_LINE を 1000000 にまで拡張することで，全単語を格納できるように修正した。再度パイプライン全体を実行し，diff -w で結果を比較したところ，単語 a を含むすべての行が正解ファイルと一致することを確認した。この過程を通じて，単にアルゴリズムの正しさを考えるだけでなく，実際のデータ規模に見合った配列サイズやメモリ上限を設計する重要性を理解した。

考察

本課題では，個々のフィルタプログラムは比較的単純であっても，それらをパイプラインと

して組み合わせることで，大規模テキストからの単語頻度表作成という実用的な処理系を構築できることを体験した。とくに，課題3で示されたように，仕様変更をフィルタ同士の接続順を変えるだけで達成できる点は，UNIX 的な小さなプログラムの設計思想の利点をよく示していると感じた。