Wstęp do programowania w języku C

Grupa MSz w czwartki

Lista 8 na zajęcia 8.12.2022

Zadanie 1. ***(15 punktów na pierwszej pracowni, 10 punktów na drugiej)***

Napisz program, który przy pomocy drzewa trie[1] sortuje leksykograficznie dany ciąg słów. Na wejściu dane są słowa złożone z liter 'a'–'z', każde w osobnym wierszu. Program powinien zbudować drzewo, wypisać posortowany ciąg, a na końcu zwolnić pamięć.

Drzewo trie składa się z węzłów, z których każdy może mieć do 26 dzieci odpowiadających kolejnym literom. Każda ścieżka od korzenia do węzła reprezentuje słowo złożone z liter które wybieramy na krawędziach.

Przydatna definicja węzła:

```
typedef struct Node {
  int count;
  struct Node* edges[26];
} Node;
```

Węzeł drzewa zawiera więc wskaźniki do dzieci oraz licznik słów, które kończą się w tym węźle (to są wtedy takie same słowa).

Można założyć, że długość słów to maksymalnie 10^6.

Samo zbudowanie drzewa w trakcie pierwszej pracowni pozwala zachować bonus punktowy na zdobycie 15 pkt. później.

Po utworzeniu, przeglądając drzewo rekurencyjnie od lewej do prawej (tzw. porządek pre-order) znajdziemy słowa w porządku leksykograficznym. Nie trzeba przejmować się potencjalnym brakiem pamięci na stercie, ale na stosie już tak, więc trzeba ograniczyć potencjalne wywołania rekurencyjne do małej stałej głębokości.

Przykład 1:

```
bb
```

[1] https://pl.wikipedia.org/wiki/Drzewo_trie

```
aaa
bb
a
```

Wynik:

```
a
aaa
bb
bb
```

Wskazówka: Przykład jak można ręcznie utworzyć drzewo dla dwóch słów `x` i `xy`:

```
Node *root = (Node*)calloc(sizeof(Node),1);

root->edges['x'-'a'] = (Node*)calloc(sizeof(Node),1);
root->edges['x'-'a'] -> count = 1;

root->edges['x'-'a'] -> edges['y'-'a'] = (Node*)calloc(sizeof(Node),1);
root->edges['x'-'a'] -> edges['y'-'a'] -> count = 1;
```

Zadanie 2. *Przypominam, że do każdej listy w SKOSie jest jeszcze do zrobienia zadanie dla sprawdzaczki, które ma osobny termin.*

Propozycje tematów

Co do zasady, projekt powinien być napisany w języku C. Szczegóły jak zawsze do indywidualnego ustalenia ze mną.

1. Gra w trybie tekstowym. Np. tekstowe RPG, gra planszowa z komputerem (może być coś z: `https://en.wikipedia.org/wiki/List_of_abstract_strategy_games`), albo gra logiczna. Trzeba zadbać o oprawę wizualną wykorzystując różne znaczki i kolory w konsoli (używając ANSI escape sequences).

2. Gra w trybie graficznym. Np. platformówka, RPG w 2D, coś z budowaniem (sim, tycoon), tower defense, albo gra jak w poprzednim punkcie. W zależności od typu gry, powinna być w czasie rzeczywistym lub turowa.

3. Zdobyć znaczący wynik w tym problemie:
   `https://www.codingame.com/multiplayer/optimization/samegame`.
   Lub ewentualnie wybrać inny dostępny problem optymalizacyjny:
   `https://www.codingame.com/multiplayer/optimization`.

4. Przeglądarka fraktali (w trybie graficznym): zbiór znanych fraktali z możliwością interaktywnej zmiany ich parametrów; do tego powiększanie, przesuwanie, itp. Można użyć np. biblioteki allegro dla C: `https://liballeg.org`.

5. Biblioteka i program testujący wydajność wybranych trików bitowych ze strony:
   `http://graphics.stanford.edu/~seander/bithacks.html`.
   Program powinien testować różne implementacje tych samych funkcji dla losowych danych z różnymi rozkładami i określić, które są najszybsze dla używanego kompilatora i sprzętu.

6. Zapoznać się z narzędziem `perf`:
   `https://perf.wiki.kernel.org/index.php/Main_Page`
   `https://man7.org/linux/man-pages/man2/perf_event_open.2.html`
   Opracować sposób użycia takich statystyk jako substytut czasu wykonania dla danego programu na danym sprzęcie. Rzeczywisty czas wykonania zmienia się istotnie zależnie od obciążenia komputera, ale niektóre statystyki perf mniej od niego zależą. Dałoby się więc zastąpić pomiar czasu czymś bardziej stabilnym, o ile znajdziemy przełożenie tych statystyk (jakieś wagi, itp.) na orientacyjny czas wykonania programu lub funkcji gdyby została ona obliczona bez żadnego dodatkowego obciążenia.

7. Wypróbować `gcc-jitlib` do przyspieszania obliczeń (`https://gcc.gnu.org/onlinedocs/jit`). Wybrać jakiś problem obliczeniowy (np. spełnialność formuł) i porównać algorytm rozwiązujący go w dwóch implementacjach, z JIT i bez JIT.