Słownik prefiksowy

Słownikiem prefiksowym nazywamy strukturę danych przechowującą napisy, pozwalającą wykonywać następujące operacje:

- wstawienie napisu do słownika
- usunięcie napisu ze słownika
- zapytanie, czy któreś słowo ze słownika ma prefiks (tj. początkowy fragment) taki jak dany wzorzec.

Twoim zadaniem będzie implementacja słownika prefiksowego.

Operacje na słowniku

Słowa wstawiane do słownika otrzymują kolejne numery, począwszy od 0. Każde ze słów w zapytaniach jest niepustym ciągiem małych liter alfabetu angielskiego. Oto pełna lista operacji na słowniku prefiksowym:

- `insert` *word*
  Wstawia podane słowo *word* do słownika.
- `prev` *number start end*
  Wstawia do słownika słowo będące podsłowem słowa o numerze *number*, zaczynającym się na pozycji *start* i kończącym się na pozycji *end* (włącznie z tymi dwiema pozycjami). W poprawnym poleceniu zakładamy, że *start* <= *end* oraz że słowo o numerze *number* znajduje się obecnie w słowniku. Pozycje liter w słowach numerujemy od 0.
- `delete` *number*
  Usuwa ze słownika słowo o numerze *number*. Nie wpływa na wartość licznika wstawianych słów.
- `find` *pattern*
  Daje w wyniku `YES` albo `NO`, w zależności od tego, czy któreś ze słów w słowniku ma prefiks równy słowu *pattern*.
- `clear`
  Całkowicie czyści słownik: usuwa wszystkie słowa ze słownika, tak że ewentualne kolejne wstawione słowo otrzyma numer 0.

Jeśli podczas wykonywania którejś z operacji `insert` lub `prev` próbowalibyśmy wstawić słowo, które znajduje się już w słowniku, to taką operację należy zignorować. Podobnie w przypadku ponownego usuwania słowa, które już raz usunęliśmy ze słownika.

Drzewo TRIE

Słownik prefiksowy należy zaimplementować za pomocą drzewa TRIE (opis w wersji anglojęzycznej). Jako że liczba wszystkich węzłów tego drzewa mogłaby być duża (wskutek wykonywania operacji `prev`), drzewo należy przechowywać w postaci skompresowanej (tzw. drzewo PATRICIA lub Radix tree), czyli pamiętać tylko następujące węzły:

- korzeń drzewa - jeśli drzewo jest niepuste, oraz
- węzły odpowiadające całym słowom ze słownika prefiksowego, oraz
- węzły mające więcej niż jedno dziecko.

Etykieta na każdej krawędzi jest niepustym ciągiem znaków. Do reprezentacji dzieci węzła można użyć zwykłej tablicy indeksowanej pierwszymi literami na krawędziach. Program powinien umożliwiać wypisywanie liczby przechowywanych węzłów drzewa.

Dane i wyniki

Twój program powinien wczytywać polecenia ze standardowego wejścia i wypisywać odpowiedzi na standardowe wyjście. Poszczególne polecenia dla programu będą podawane w osobnych wierszach. Poprawne polecenia są jednej z następujących postaci:

`insert` *word*
`prev` *number start end*
`delete` *number*
`find` *pattern*
`clear`

W opisie poprawnego polecenia są dopuszczalne wielokrotne spacje zamiast pojedynczych, a także dodatkowe spacje na początku i na końcu opisu polecenia, ale nie są dopuszczalne żadne inne dodatkowe znaki. Liczby całkowite podane na wejściu nie mogą mieć zer wiodących (Czyli liczba 001 nie jest dozwolona. Liczba 0 jest jedyną dozwoloną liczbą z wiodącym zerem.) Parametry w poprawnych poleceniach muszą odpowiadać istniejącym słowom i pozycjom w słowach i spełniać wszystkie warunki wymienione w sekcji "Operacje na słowniku". W przypadku, gdy polecenie w danym wierszu nie jest poprawne, Twój program powinien zignorować to polecenie, wypisać na standardowe wyjście wiersz:

`ignored`

i przejść do następnego wiersza wejścia.

Twój program powinien wypisywać odpowiedzi na poprawne polecenia w osobnych wierszach. Wynikiem dla zapytania typu `find` powinno być jedno słowo `YES` albo `NO`. Wynikiem dla każdej poprawnej operacji `insert` i `prev` powinien być numer, jaki został nadany w słowniku nowo wstawionemu wzorcowi, podany w postaci:

`word number:` *number*

Wynikiem dla każdej poprawnej operacji `delete` powinno być:

`deleted:` *number*

Wynikiem dla każdej poprawnej operacji `clear` powinno być jedno słowo `cleared`.

Jeśli dodatkowo program zostanie wywołany z parametrem `-v`, powinien dla każdego z poleceń poza `find` wypisać na standardowe wyjście diagnostyczne (`stderr`) następujący wiersz:

`nodes:` n

gdzie n to liczba całkowita – łączna liczba węzłów drzewa TRIE przechowywanych w pamięci po wykonaniu tego zapytania. Nie dopuszczamy innych parametrów wywołania (w przypadku napotkania innego parametru, program powinien poinformować o błędzie).

Ograniczenia

Twój program nie musi sprawdzać następujących ograniczeń:

- Długość żadnego wiersza pliku wejściowego nie przekroczy 10^5.
- Rozmiar pliku wejściowego nie przekroczy 2 MB.

- Suma długości wszystkich słów w słowniku (tj. suma długości wszystkich słów wstawianych za pomocą operacji `insert` i `prev`) nie przekroczy $3 * 10^8$. Programy o złożoności pamięciowej liniowej względem sumy długości wszystkich słów w słowniku (czyli przechowujące jawnie w pamięci wszystkie słowa wstawiane za pomocą operacji `prev`) są narażone na przekroczenie limitu czasowego lub pamięciowego.
- Każdy wiersz pliku wejściowego kończy się linuksowym znakiem końca linii (kod ASCII 10). Poza tym w pliku wejściowym występują tylko drukowalne znaki kodu ASCII.

Ponadto Twój program:

- Będzie miał do dyspozycji 256 MB pamięci.
- Przed zakończeniem musi zwolnić całą zaalokowaną pamięć.

Implementacja

Twoje rozwiązanie powinno zawierać następujące pliki:

- `trie.h`
  Plik nagłówkowy biblioteki wykonującej operacje na (jednym) drzewie TRIE.
- `trie.c`
  Implementacja biblioteki wykonującej operacje na (jednym) drzewie TRIE.
- `parse.h`
  Plik nagłówkowy biblioteki wczytującej liczby całkowite i napisy.
- `parse.c`
  Implementacja biblioteki wczytującej liczby całkowite i napisy.
- `dictionary.c`
  Główny plik programu, w którym wczytujemy wejście i wywołujemy funkcje z pliku `trie.h`. Plik ten nie powinien znać typu danych służącego do przechowywania drzewa.

Dodatkowo powinien być dołączony plik `Makefile`, tak, aby w wyniku wywołania polecenia `make` został wytworzony program wykonywalny `dictionary`. Dodatkowo w wyniku wywołania polecenia `make debug` powinien zostać wytworzony plik `dictionary.dbg`, który powinien zawierać symbole do debugowania (opcja `-g` kompilacji) tak, aby ułatwiało to śledzenie wycieków pamięci za pomocą programu `valgrind`. Jeśli któryś z plików źródłowych ulegnie zmianie, ponowne wpisanie `make` lub `make debug` powinno na nowo stworzyć odpowiedni plik wykonywalny.

Zachęcamy, by `Makefile` działał w następujący sposób:

1. osobno kompilował każdy plik `.c` i osobno linkował,
2. przy zmianie w pliku źródłowym powinny być wykonane tylko te polecenia kompilacji, które są potrzebne,
3. `make clean` powoduje usunięcie wszystkich plików wykonywalnych i dodatkowych plików kompilacji.

Jednak w tym zadaniu nie będziemy stosować kar punktowych za brak spełnienia tych trzech warunków.

Punktacja

Za w pełni poprawne rozwiązanie zadania implementujące wszystkie funkcjonalności można zdobyć maksymalnie 20 punktów. Możliwe są punkty karne za poniższe uchybienia:

- Za wycieki pamięci traci się co najwyżej 6 punktów.
- Za brak obsługi błędów na wejściu można stracić co najwyżej 5 punktów. (W takim wypadku implementacja plik `parse.c` może zawierać trywialne wczytywanie).
- Programy o złożoności pamięciowej liniowej względem sumy długości wszystkich słów w słowniku są narażone na utratę ok. 5 punktów.
- Za brak obsługi parametrów wywołania traci się co najwyżej 4 punkty.
- Za niezgodną ze specyfikacją strukturę plików w rozwiązaniu można stracić co najwyżej 4 punkty.
- Za błędy stylu kodowania można stracić co najwyżej 3 punkty.

Za rozwiązanie, które przechowuje drzewo TRIE w postaci nieskompresowanej, nie obsługuje zapytań typu `prev` i parametru wywołania `-v`, można uzyskać maksymalnie 10 punktów. Za brak obsługi błędów na wejściu taki program może stracić co najwyżej 3 punkty.

Przed pierwszym zgłoszeniem, za każde rozpoczęte 12 godzin opóźnienia przysługuje kara -1 pkt (dotyczy tylko dni roboczych). Zgłoszenie zostanie ocenione przez prowadzącego na zestawie testów, a student otrzyma raport ze sprawdzania. Za sprawdzenie każdego kolejnego zgłoszenia kara punktowa wynosi -1 punkt. Wówczas za każdy pełny dzień roboczy (0:00-23:59), który upłynie między podaniem wyniku oceny danego zgłoszenia a ponownym zgłoszeniem, kara punktowa wynosi -2 punkty. Ocena stylu rozwiązania następuje na ostatnim wykonanym zgłoszeniu (warto poinformować sprawdzającego, że rozwiązanie może zostać uznane za ostateczne).

Rozwiązania należy implementować **samodzielnie** pod rygorem niezaliczenia przedmiotu.

Przykład

Dla danych wejściowych:	poprawnym wynikiem jest:

`insert aaaa`
`insert ababc`
`FIND aa`
`  find  aa`
`find bab`
`prev 1 1`
`4`
`prev 1 1 4`
`find bab`
`insert aaa`
`prev 0 0 2`
`find aaa`
`find aa a`
`find aba`
`delete 1`
`find aba`
`delete 1`
`insert abc`
`clear`
`insert ababc`
`prev 0 1 4`
`find bab`
	`word number: 0`
	`word number: 1`
	`ignored`
	`YES`
	`NO`
	`ignored`
	`ignored`
	`word number: 2`
	`YES`
	`word number: 3`
	`ignored`
	`YES`
	`ignored`
	`YES`
	`deleted: 1`
	`NO`
	`ignored`
	`word number: 4`
	`cleared`
	`word number: 0`
	`word number: 1`
	`YES`

Jeśli program zostanie uruchomiony z parametrem `-v`, na standardowe wyjście diagnostyczne powinien wypisać:

```
nodes: 2
nodes: 4
nodes: 5
nodes: 6
nodes: 4
nodes: 6
nodes: 0
nodes: 2
nodes: 3
```