Indywidualny projekt programistyczny (Info, I rok) 16/17

Zadanie Drzewo

Zmiany w treści zadania po 13.03.2017 zostały zaznaczone kolorem czerwonym.

Celem zadania jest implementacja drzewiastej struktury danych w oparciu o listy dwukierunkowe. Początkowo struktura zawiera jeden wierzchołek (korzeń) o numerze 0.

Operacje na strukturze

- ADD_NODE <k>
  Dodaje nowy wierzchołek o numerze o jeden większym od poprzednio utworzonego. Jego rodzicem staje się wierzchołek o numerze <k>. Ustalamy, że nowy wierzchołek umiejscowiony zostaje na prawo od pozostałych dzieci <k>.
- RIGHTMOST_CHILD <k>
  Wypisuje numer wierzchołka, który jest skrajnie prawym dzieckiem <k>. Jeśli <k> jest liściem, wypisuje -1.
- DELETE_NODE <k>
  Usuwa wierzchołek o numerze <k> (k różne od 0). Krawędzie wychodzące z usuniętego wierzchołka w stronę dzieci zostają zaczepione w jego rodzicu, w tym samym miejscu, z którego wychodziła krawędź w stronę <k>. Kolejność pomiędzy krawędziami zostaje zachowana.
- DELETE_SUBTREE <k>
  Usuwa wierzchołek o numerze <k> (k różne od 0) razem z całym jego poddrzewem.
- SPLIT_NODE <k> <w>
  Dodaje nowy wierzchołek o numerze o jeden większym od poprzednio utworzonego. Jego rodzicem staje się wierzchołek o numerze <k>. Nowy wierzchołek umiejscowiony zostaje na prawo od wierzchołka <w> (można założyć, że <k> jest rodzicem <w>). Dodatkowo wszystkie krawędzie wychodzące z <k>, będące na prawo od wierzchołka o numerze <w>, zostają zaczepione w nowym wierzchołku, z zachowaniem kolejności.

Można założyć, że wierzchołki, których numery występują w poleceniach, znajdują się w strukturze. Wymagamy użycia listy do reprezentacji krawędzi wychodzących z wierzchołków. Każda pojedyncza operacja powinna działać w czasie stałym, za wyjątkiem DELETE_SUBTREE, gdzie czas działania powinien być proporcjonalny do rozmiaru usuwanego poddrzewa. Ponadto po wykonaniu operacji różnej od RIGHTMOST_CHILD należy wypisać jedną linię ze słowem OK.

Wyjście diagnostyczne

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

NODES: <n>

gdzie <n> to liczba wierzchołków aktualnie znajdujących się w strukturze. Nie dopuszczamy innych parametrów wywołania – w przypadku napotkania innego parametru, program powinien wypisać na standardowe wyjście komunikat ERROR i zakończyć działanie z kodem wyjścia 1. Przykład użycia parametrów wywołania programu znajduje się w zasobach na moodle w pliku params.c.

Skrypt testujący

Osobną częścią zadania jest napisanie skryptu test.sh. Po wywołaniu

./test.sh <prog> <dir>

skrypt powinien uruchomić program <prog> dla wszystkich plików wejściowych postaci <dir>/*.in, porównać wyniki z odpowiadającymi im plikami .out, a następnie wypisać, które testy zakończyły się niepowodzeniem. Wywołanie ./test.sh -v <prog> <dir> powinno sprawdzić także poprawność programu uruchomionego z opcją -v. Wyniki z wyjścia diagnostycznego należy porównać z plikami .err.

Dane wejściowe

Program powinien czytać ze standardowego wejścia. Można przyjąć następujące założenia o danych wejściowych:

- Parametry <k> i <w> są nieujemnymi liczbami całkowitymi mniejszymi niż 2^{31}.
- Rozmiar pliku wejściowego nie przekroczy 5 MB.
- Polecenia i liczby są pooddzielane pojedynczymi spacjami, a każdy wiersz pliku wejściowego kończy się linuksowym znakiem końca linii (kod ASCII 10). Są to jedyne białe znaki występujące w plikach wejściowych.

Program będzie miał do dyspozycji 32 MB pamięci. Przed zakończeniem należy zwolnić całą zaalokowaną pamięć.

Podział na pliki

Rozwiązanie powinno zawierać następujące pliki:

- tree.h
  Plik nagłówkowy biblioteki wykonującej operacje na drzewiastej strukturze danych. Nie wymagamy tworzenia dodatkowych plików bibliotecznych, niemniej zachęcamy, aby wydzielić implementację różnych części projektu do osobnych plików np. list.h + list.c, parse.h + parse.c, w zależności od podjętych decyzji projektowych.
- tree.c
  Implementacja biblioteki wykonującej operacje na drzewiastej strukturze danych.
- solution.c
  Główny plik programu, w którym wczytujemy wejście i wywołujemy funkcje z pliku tree.h. Plik ten nie powinien znać definicji typów, użytych do implementacji struktury danych.
- test.sh
  Patrz punkt „skrypt testujący”.
- Makefile
  W wyniku wywołania polecenia make powinien zostać wytworzony program wykonywalny solution. Dodatkowo w wyniku wywołania polecenia make debug powinien zostać wytworzony plik solution.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.

Jeżeli wykonamy polecanie make i make debug w dowolnej kolejności, to powinny pojawić się oba pliki wykonywalne solution i solution.dbg.

Zachęcamy, by Makefile działał w następujący sposób:

- osobno kompiluje każdy plik .c i osobno linkuje,
- przy zmianie w pliku źródłowym wykonuje tylko potrzebne polecenia kompilacji,
- make clean usuwa wszystkie pliki wykonywalne i dodatkowe pliki powstałe podczas kompilowania.

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 można stracić co najwyżej 6 punktów.
- Brak obsługi parametrów wywołania lub błędne wyniki na wyjściu diagnostycznym grożą utratą 4 punktów.
- 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 błędy w skrypcie testującym można stracić maksymalnie 2 punkty.
- Programy o złożoności pamięciowej lub obliczeniowej gorszej od oczekiwanej są narażone na utratę do 10 punktów.
- Rozwiązania oparte na innych strukturach danych niż dynamicznie alokowane listy będą oceniane na 0 punktów.

Rozwiązania należy implementować samodzielnie pod rygorem niezaliczenia przedmiotu.

Status przesłanego zadania

Numer próby	To jest próba nr 1.
Status przesłanego zadania	Wersja robocza
Stan oceniania	Ocenione
Termin oddania	czwartek, 30 marzec 2017, 23:55
Pozostały czas	Opóźnienie w przesłaniu: 37 dni 20 godz.
Ostatnio modyfikowane	czwartek, 30 marzec 2017, 17:08
Przesyłane pliki	drzewo-kk385830.zip
Komentarz do przesłanego zadania	Komentarze (0)

Edytuj zadanie

Dodaj lub edytuj swoje zadanie

Zgłoś zadanie

Po zgłoszeniu tego zadania, nie będzie można wprowadzić zmian.

Informacja zwrotna

Ocena	17,00 / 20,00
Ocenione dnia	czwartek, 13 kwiecień 2017, 21:03
Ocenione przez	Krzysztof Kotuła
Komentarz zwrotny	* make && make debug produkuje solution.dbg bez symboli debugowych * test.sh niepoprawnie obsługuje spacje i kropki w drugim (trzecim) ...

WPI.INFO.I.16/17
WPI.LAB.INFO.I.16/17
PO.INFO.I.16/17
MD.INFO.I.16/17

ADMINISTRACJA

Administracja kursem

Jesteś zalogowany(a) jako Krzysztof Kowalczyk (Wyloguj)
IPP.INFO.I.16/17

Moodle, wersja 3.2.2+ | moodle@mimuw.edu.pl