Wstęp do programowania, potok imperatywny (Info, I rok) 18/19, laboratorium

Kompresja

Wprowadzenie

Kompresją drzewa nazwiemy przekształcenie go w skierowany graf acykliczny (DAG) przez połączenie wystąpień powtarzających się niepustych poddrzew.

Powiemy, że kompresja jest maksymalna, jeśli łączy wszystkie wystąpienia powtarzających się poddrzew.

Polecenie

Napisz program, który wczyta z wejścia tekstową reprezentację drzewa binarnego dodatnich liczb całkowitych i wypisze na wyjście efekt jego maksymalnej kompresji.

Postać danych

Puste drzewo jest reprezentowane przez wiersz z liczbą `0` a drzewo niepuste przez ciąg wierszy, z których pierwszy zawiera wartość korzenia, w kolejnych jest reprezentacja lewego a po niej prawego poddrzewa.

Postać wyniku

Wynik programu otrzymujemy z danych wejściowych, zastępując drugie i każde kolejne wystąpienie ciągu wierszy reprezentujących takie samo niepuste poddrzewo jednym wierszem. Umieszczamy w nim wartość `K - N`, gdzie `N` to numer tego wiersza w tekście wynikowym, liczony od `1`, a `K` to numer wiersza wyniku, w którym zaczyna się reprezentacja pierwszego wystąpienia poddrzewa.

Przykłady

Do treści zadania dołączone są pliki `.in` z przykładowymi danymi i pliki `.out` z wynikami wzorcowymi.

- Dla danych przyklad1.in poprawny wynik to przyklad1.out.
- Dla danych przyklad2.in poprawny wynik to przyklad2.out.

- Dla danych przyklad3.in poprawny wynik to przyklad3.out.

Walidacja i testy

- Rozwiązania zostaną poddane walidacji, wstępnie badającej zgodność ze specyfikacją.

  Walidacja sprawdzi działanie programu na przykładach dołączonych do treści zadania.

  Pomyślne przejście walidacji jest warunkiem dopuszczenia programu do testów poprawności. Program, który walidacji nie przejdzie, dostanie zerową ocenę poprawności.

- Walidacja i testy zostaną przeprowadzone na komputerze `students`.

- Programy będą kompilowane poleceniem:

  ```
  gcc -std=c11 -pedantic -Wall -Wextra -Werror -fstack-protector-strong -g nazwa.c
   -o nazwa
  ```

  gdzie `nazwa.c` to nazwa pliku z kodem źródłowym.

  Wymagane są wszystkie wymienione opcje kompilatora. Nie będą do nich dodawane żadne inne.

  Zwracamy uwagę, że poszczególne wersje kompilatora `gcc` mogą się różnić sposobem obsługi tych samych opcji. Przed wysłaniem rozwiązania warto więc skompilować je i przetestować na `students`, w sposób opisany powyżej.

- Podczas walidacji i testów, program `nazwa` z rozwiązaniem będzie uruchamiany pod kontrolą programu Valgrind poleceniem:

  ```
  valgrind --leak-check=full -q ./nazwa
  ```

  Jeśli Valgrind wykryje błąd, to nawet, gdyby wynik był prawidłowy uznamy, że program testu nie przeszedł.

  Opcja `--leak-check=full` wskazuje Valgrindowi, że powinien, między innymi, szukać wycieków pamięci.

- Przyjmujemy, że wynik funkcji `main()` inny niż `0` informuje o błędzie wykonania programu.

- Poprawność wyniku sprawdzamy, przekierowując na wejście programu zawartość pliku `.in` i porównując rezultat, za pomocą programu `diff`, z plikiem `.out`, np.:

  ```
  < przyklad.in ./nazwa | diff - przyklad.out
  ```

  Ocena poprawności wyniku jest binarna. Uznajemy go za poprawny, jeżeli program `diff` nie wypisze żadnej różnicy między wynikiem programu a wynikiem wzorcowym.

Uwagi i wskazówki

- Wolno założyć, że dane są poprawne.
- Wolno założyć, że każdy wiersz danych, także ostatni, będzie zakończony reprezentacją końca wiersza `\n`.
- Efektywność rozwiązania może mieć wpływ na ocenę jego jakości.