

Wstęp do programowania – laboratorium, potok imperatywny (Inf) 21/22.Z

[Kokpit](#)[Moje kursy](#)[WPI.Lab.Inf.21/22.Z](#)[Laboratorium 10 \(6 i 8 grudnia 2021\)](#)[Napier-NAF \(treść, rozwiązanie i ocena jakości\)](#)

Napier-NAF (treść, rozwiązanie i ocena jakości)

Wprowadzenie

Balanced binary representation (BBR) jest pozycyjnym zapisem liczby całkowitej z podstawą 2 i cyframi -1, 0, 1. Zapis BBR liczby jest w postaci [non-adjacent form \(NAF\)](#) jeśli cyfry niezerowe nie sąsiadują ze sobą.

Każda liczba całkowita ma dokładnie jeden zapis BBR-NAF bez zer wiodących. Dla 0 jest on pusty. Dla niezerowego x najmniej znaczącą cyfrą c jest:

- 0 jeśli x jest parzyste,
- 1 jeśli $x - 1$ jest podzielne przez 4,
- -1 jeśli $x + 1$ jest podzielne przez 4.

Bardziej znaczące cyfry zapisu x tworzą zapis liczby $(x - c) / 2$.

Zapis BBR-NAF minimalizuje, w klasie zapisów binarnych, [wagę Hamminga](#) liczby, czyli liczbę cyfr niezerowych. Na przykład [liczby pierwsze Mersenne'a](#) mają w tym zapisie wagę 2. Zapisu BBR-NAF używa się w [kryptografii krzywych eliptycznych](#) stosowanej, między innymi, w implementacjach kryptowaluty [Bitcoin](#) i platformy [Ethereum](#).

Tematem zadania jest reprezentacja zapisu BBR-NAF, umożliwiającą efektywne operacje arytmetyczne na dużych liczbach całkowitych z małą wagą Hamminga.

Reprezentacja jest oparta na systemie liczbowym [arithmeticæ localis](#), opisanym przez Johna Napiera, w 1617 roku, w traktacie *Rabdologiae*. Jest to system binarny, ale nie pozycyjny, lecz addytywny.

W systemach pozycyjnych wartość liczby jest określona przez ciąg cyfr jej zapisu, z uwzględnieniem pozycji, na której cyfra występuje. W systemach addytywnych wartość liczby jest sumą wartości cyfr.

W *arithmeticæ localis* rolę cyfr pełnią litery, reprezentujące potęgi dwójki. W naszej reprezentacji, którą nazwiemy *Napier-NAF*, cyframi są liczby całkowite, które reprezentują potęgi dwójki i liczby do nich przeciwne. Reprezentacją cyfry o wartości 2^{**n} jest nieujemna liczba całkowita n . Reprezentacją cyfry o wartości $-(2^{**n})$ jest ujemna liczba całkowita $-n - 1$.

Reprezentacją Napier-NAF liczby całkowitej x jest tablica liczb całkowitych, której długością jest waga Hamminga zapisu BBR-NAF liczby x . W tablicy:

- ujemna liczba n informuje o wystąpieniu cyfry -1 na pozycji $-n - 1$ zapisu BBR-NAF liczby x ,
- nieujemna liczba n informuje o wystąpieniu cyfry 1 na pozycji n zapisu BBR-NAF liczby x ,
- informacje o cyfrach są uporządkowane w kolejności od najmniej znaczących a pozycje są numerowane od 0.

Polecenie

W pliku nagłówkowym [napiernaf.h](#) jest interfejs modułu operacji arytmetycznych na liczbach całkowitych w reprezentacji Napier-NAF:

- funkcja

```
void iton(int x, int **a, int *n);
```

zapisuje reprezentację liczby x w dynamicznie utworzonej n -elementowej tablicy a ,

- funkcja

```
int ntoi(int *a, int n);
```

daje jako wynik wartość liczby reprezentowanej przez n -elementową tablicę a , lub 0 , jeśli wartość ta nie mieści się w zakresie typu int ,

- funkcja

```
void nadd(int *a, int an, int *b, int bn, int **c, int *cn);
```

zapisuje w dynamicznie utworzonej $*cn$ -elementowej tablicy $*c$ sumę liczb reprezentowanych przez an -elementową tablicę a i bn -elementową tablicę b ,

- funkcja

```
void nsub(int *a, int an, int *b, int bn, int **c, int *cn);
```

zapisuje w dynamicznie utworzonej $*cn$ -elementowej tablicy $*c$ różnicę liczb reprezentowanych przez an -elementową tablicę a i bn -elementową tablicę b ,

- funkcja

```
void nmul(int *a, int an, int *b, int bn, int **c, int *cn);
```

zapisuje w dynamicznie utworzonej $*cn$ -elementowej tablicy $*c$ iloczyn liczb reprezentowanych przez an -elementową tablicę a i bn -elementową tablicę b ,

- funkcja

```
void nexp(int *a, int an, int *b, int bn, int **c, int *cn);
```

zapisuje w dynamicznie utworzonej $*cn$ -elementowej tablicy $*c$ wynik potęgowania, którego podstawa jest reprezentowana przez an -elementową tablicę a , a nieujemny wykładnik jest reprezentowany przez bn -elementową tablicę b ,

- funkcja

```
void ndivmod(int *a, int an, int *b, int bn, int **q, int *qn, int **r, int *rn);
```

dzieli liczbę, której reprezentacją jest an -elementowa tablica a , przez niezerową liczbę, której reprezentacją jest bn -elementowa tablica b . Iloraz zapisuje w dynamicznie utworzonej $*qn$ -elementowej tablicy $*q$. Resztę z dzielenia zapisuje w dynamicznie utworzonej $*rn$ -elementowej tablicy $*r$.

Funkcja daje wynik zgodny z algorytmem [dzielenia Euklidesa](#). Wynikiem dzielenia całkowitej liczby a przez niezerową całkowitą liczbę b jest całkowity iloraz q i całkowita reszta r , które spełniają warunki:

- $a = b * q + r$,
- $0 \leq r < \text{abs}(b)$,

gdzie $\text{abs}(b)$ jest wartością bezwzględną liczby b .

Napisz moduł `napiernaf.c` implementujący interfejs [napiernaf.h](#).

Przykłady

Do treści zadania dołączone są przykłady programów korzystających z modułu `napiernaf.c` i pliki `.out` z wynikami wzorcowymi.

- Poprawnym wynikiem programu [przyklad1.c](#) jest [przyklad1.out](#).
- Poprawnym wynikiem programu [przyklad2.c](#) jest [przyklad2.out](#).
- Poprawnym wynikiem programu [przyklad3.c](#) jest [przyklad3.out](#).
- Poprawnym wynikiem programu [przyklad4.c](#) jest [przyklad4.out](#).

Walidacja i testy

- Rozwiązania podlegają walidacji, wstępnie badającej zgodność ze specyfikacją.

Walidacja sprawdza działanie programu na przykładach dołączonych do treści zadania.

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

- Walidacja i testy są prowadzone na komputerze `students`.

- Programy są kompilowane poleceniem:

```
gcc @opcje nazwa.c napiernaf.c -o nazwa
```

gdzie `nazwa.c` to nazwa programu testującego, który korzysta z modułu `napiernaf.c`. Plik `opcje` ma zawartość:

```
-std=c17
-pedantic
-wall
-Wextra
-Wformat-security
-Wduplicated-cond
-Wfloat-equal
-Wshadow
-Wconversion
-Wjump-misses-init
-Wlogical-not-parentheses
-Wnull-dereference
-Wvla
-Werror
-fstack-protector-strong
-fsanitize=undefined
-fno-sanitize-recover
-g
-fno-omit-frame-pointer
-O1
```

Opcje `-std=c17`, `-pedantic` wskazują, że kompilator ma dbać o zgodność kodu z aktualnym standardem języka C.

Dzięki opcjom `-wall`, `-Wextra` kompilator zgłosi zauważone usterki.

Opcje `-Wformat-security`, `-Wduplicated-cond`, `-Wfloat-equal`, `-Wshadow`, `-Wconversion`, `-Wjump-misses-init`, `-Wlogical-not-parentheses`, `-Wnull-dereference` umożliwiają wykrywanie dodatkowych usterek.

Opcja `-Wvla` sprawia, że użycie tablic zmiennej długości jest uznawane za usterkę.

Opcja `-Werror` wskazuje, że kompilator ma uznać usterki za błędy.

Dzięki opcji `-fstack-protector-strong`, podczas wykonania programu zostaną wykryte niektóre błędne odwołania do pamięci na stosie.

Opcje `-fsanitize=undefined`, `-fno-sanitize-recover` umożliwiają wykrywanie operacji, które mają efekt nieokreślony.

Opcje `-g`, `-fno-omit-frame-pointer` poprawiają jakość komunikatów o błędach wykonania.

Opcja `-O1` włącza optymalizacje, co zwiększa prawdopodobieństwo ujawnienia się błędów.

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` jest uruchamiany pod kontrolą programu Valgrind poleceniem:

```
valgrind --leak-check=full -q --error-exitcode=1 ./nazwa
```

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

Opcja `-q` powoduje, że jedynymi informacjami, wypisywanymi przez program Valgrind, są komunikaty o błędach.

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

Opcja `--error-exitcode=1` określa kod wyjścia programu w przypadku, gdy Valgrind wykryje błąd.

- Przyjmujemy, że niezerowy wynik funkcji `main()` informuje o błędzie wykonania programu.
- Poprawność wyniku programu sprawdzamy przez porównanie, za pomocą programu `diff`, z plikiem `.out`, np.:

```
./przyklad | diff - przyklad.out
```

Ocena poprawności wyniku jest binarna. Wynik uznajemy za poprawny, jeżeli program `diff` nie wskaże żadnej różnicy względem wyniku wzorcowego.

Uwagi i wskazówki

- Do walidacji zostaną użyte tylko programy [przyklad1.c](#), [przyklad2.c](#) i [przyklad3.c](#).

Siedem z dziesięciu testów nie korzysta z funkcji `ndivmod()`.


Rozwiązanie, w którym funkcja `ndivmod()` nie jest zaimplementowana prawidłowo, może więc pomyślnie przejść walidację i dostać maksymalnie 7 punktów za poprawność.

- Pliku [napiernaf.h](#) nie wolno zmieniać. Jako rozwiązanie należy wysłać tylko plik tekstowy `napiernaf.c` z kodem źródłowym w języku C.
- Wolno założyć, że wykładnik przekazany funkcji `nexp()` jest nieujemny a dzielnik przekazany funkcji `ndivmod()` jest niezerowy.
- Rozwiązanie nie powinno nakładać ograniczeń na wartość reprezentowanych liczb. Wolno tylko założyć, że reprezentacje zmieszczą się w pamięci a elementy tablic nie przekroczą zakresu typu `int`.
- Złożoność czasowa i pamięciowa algorytmu funkcji `iton()` powinna być logarytmiczna względem wartości pierwszego argumentu.
- Złożoności czasowe i pamięciowe algorytmów funkcji `ntoi()`, `nadd()` i `nsub()` powinny być liniowe względem łącznego rozmiaru danych.
- Złożoności czasowe i pamięciowe algorytmów funkcji `nmul()` i `ndivmod()` powinny być wielomianowe względem łącznego rozmiaru danych.
- Złożoność czasowa i pamięciowa algorytmu funkcji `nexp()` powinna być wielomianowa względem sumy wartości wykładnika i długości reprezentacji podstawy.

Status przesłanego zadania

Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Ocenione
Termin oddania	wtorek, 11 stycznia 2022, 20:00
Pozostały czas	Zadanie zostało złożone 22 godzin 39 min. przed terminem
Ostatnio modyfikowane	poniedziałek, 10 stycznia 2022, 21:20


Przesyłane pliki

 [napiernaf.c](#) 10 stycznia 2022, 21:20

Komentarz do przesłanego zadania

 [Komentarze \(0\)](#)

Informacja zwrotna

Ocena	8,50 / 10,00
Ocenione dnia	środa, 12 stycznia 2022, 21:19
Ocenione przez	 Marcin Peczarski