



Zadanie 2

Otwarto: poniedziałek, 5 maja 2025, 00:00

Wymagane do: piątek, 6 czerwca 2025, 23:59

Wielki Aproksymator

W tym zadaniu implementujemy programy serwera i klienta gry o tytuł Wielkiego Aproksymatora. Celem każdego gracza (reprezentowanego przez program klienta) jest jak najlepsze aproksymowanie otrzymanego od serwera wielomianu

$f(x) = \sum_{i=0}^N a_i x^i$ stopnia N w punktach będących liczbami całkowitymi od 0 do K . Początkowo wartości aproksymującej funkcji \hat{f} są zerowe.

Aproksymowanie odbywa się przez dodawanie w podanym przez klienta punkcie podanej wartości, np. dla $K = 3$ i poleceń dodania

- 4,5 w punkcie 0,
- 2 w punkcie 2,
- 3,75 w punkcie 2,

wartości aproksymującej funkcji wynoszą odpowiednio

- $\hat{f}(0) = 4,5$,
- $\hat{f}(1) = 0$,
- $\hat{f}(2) = 5,75$,
- $\hat{f}(3) = 0$.

Wynik gracza to $\sum_{x=0}^K (\hat{f}(x) - f(x))^2$, czyli suma kwadratów odchyleń od oczekiwanej wartości. Do tej wartości dodawane są opisane poniżej kary.

Gra kończy się po wykonaniu przez serwer M dodawań.

Parametry uruchamiania serwera i klienta

Serwer uruchamiamy poleceniem `approx-server` z następującymi parametrami:

- `-p port` - numer portu serwera, liczba całkowita od 0 do 65535 zapisana przy podstawie 10, parametr opcjonalny, domyślnie 0;
- `-k value` - wartość stałej K , liczba całkowita z przedziału od 1 do 10000 zapisana przy podstawie 10, parametr opcjonalny, domyślnie 100;
- `-n value` - wartość stałej N , liczba całkowita z przedziału od 1 do 8 zapisana przy podstawie 10, parametr opcjonalny, domyślnie 4;
- `-m value` - wartość stałej M , liczba całkowita z przedziału od 1 do 12341234 zapisana przy podstawie 10, parametr opcjonalny, domyślnie 131;
- `-f file` - nazwa pliku zawierającego komunikaty serwera ze współczynnikami wielomianów (patrz opis komunikatu `COEFF`), parametr obowiązkowy.

Serwer nasłuchuje na podanym porcie połączeń TCP od klientów. Jeśli numer portu jest równy zeru, serwer wybiera dowolny port. Powinno być możliwe nawiązanie zarówno połączeń IPv6, jak i IPv4 z serwerem. Jeśli komputer, na

?

którym uruchomiono serwer, nie ma możliwości komunikacji po IPv6, to powinna być możliwa komunikacja po IPv4.

Klienta uruchamiamy poleceniem `approx-client` z następującymi parametrami:

- `-u player_id` - identyfikator gracza składający się z cyfr oraz małych i dużych liter alfabetu angielskiego, parametr obowiązkowy;
- `-s server` – adres lub nazwa serwera, parametr obowiązkowy;
- `-p port` – port serwera, liczba całkowita od 1 do 65535 zapisana przy podstawie 10, parametr obowiązkowy;
- `-4` – wymuszenie komunikacji z serwerem za pomocą IPv4, parametr opcjonalny;
- `-6` – wymuszenie komunikacji z serwerem za pomocą IPv6, parametr opcjonalny;
- `-a` – wybór rodzaju strategii (patrz punkt „Strategia działania klienta”), parametr opcjonalny.

Jeśli podano oba parametry `-4` i `-6` lub nie podano żadnego z nich, klient powinien wybrać wersję protokołu wynikającą z pierwszego przypisanego serwerowi adresu IP (patrz funkcja `getaddrinfo`).

Parametry mogą być podawane w dowolnej kolejności. Zachowanie programu, gdy któryś z parametrów został podany wielokrotnie, powinno być rozsądne.

Przebieg gry

Konwencje

Komunikacja między klientem a serwerem odbywa się tekstowo w połączeniu TCP z wykorzystaniem IPv4 lub IPv6. Każdy komunikat kończy się sekwencją znaków powrotu karetki (kod ASCII 13) i nowej linii (kod ASCII 10), oznaczanej dalej jako `\r\n`. Pola komunikatów oddzielane są pojedynczą spacją. W komunikatach nie ma innych białych znaków. Pola komunikatów o zmiennych wartościach mają nazwy zaczynające się znakiem dolara `$`.

Ileć w treści zadania jest mowa o liczbach wymiernych, przesyłamy je zapisane przy podstawie dziesięć z opcjonalną częścią ułamkową oddzieloną kropką. Po kropce może znajdować się co najwyżej 7 cyfr. Poprawne są na przykład takie liczby: `42`, `12.78`, `-123.456789`, `1.0`, `-2.`, `0`.

Zgłoszenie gracza

Serwer czeka na połączenia TCP. Klient otwiera połączenie do serwera i wysyła komunikat

```
HELLO $player_id\r\n
```

Identyfikatory graczy mogą się powtarzać. Serwer nie może mylić klientów, nawet jeśli mają identyczne identyfikatory graczy.

Jeśli serwer nie otrzyma komunikatu `HELLO` w ciągu 3 sekund od podłączenia się klienta, rozłącza się z nim i usuwa wszystkie pamiętane dane związane z tym klientem.

Komunikat `HELLO` może wystąpić tylko jako pierwszy komunikat klienta po nawiązaniu połączenia z serwerem.

Na komunikat `HELLO` serwer od razu odpowiada komunikatem

```
COEFF $a_0 $a_1 ... $a_N\r\n
```

gdzie `$a_0`, `$a_1`, ..., `$a_N` są liczbami wymiernymi w rozwinięciu dziesiętnym, i oznacza, że ten klient ma tworzyć aproksymację wielomianu $f(x) = \sum_{i=0}^N a_i x^i$.
W tym celu serwer czyta kolejną linię tekstu z pliku, którego nazwa została podana

?

w parametrze `-f`. Wolno założyć, że każda linia tego pliku zawiera poprawny komunikat `COEFF` kończący się sekwencją `\r\n` oraz że plik zawiera wystarczającą liczbę takich komunikatów. Każda z liczb `$a_...` mieści się w przedziale domkniętym od -100 do 100 .

Komunikat `COEFF` może wystąpić tylko jako pierwszy komunikat serwera do klienta.

Gra

Gra rozpoczyna się, gdy któryś klient przysła polecenie

```
PUT $point $value\r\n
```

gdzie `$point` jest liczbą całkowitą z przedziału od 0 do K , a `$value` jest liczbą wymierną w rozwinięciu dziesiętnym z przedziału domkniętego od -5 do 5 .

Jeśli któraś z wartości `$point` lub `$value` jest niepoprawna, serwer po odczekaniu 1 sekundy odpowiada klientowi komunikatem

```
BAD_PUT $point $value\r\n
```

z powtórzonymi wartościami `$point` i `$value` z komunikatu `PUT` i dolicza klientowi karę 10 .

Jeśli wartości `$point` i `$value` są poprawne, serwer dodaje wartość `$value` w punkcie `$point` funkcji aproksymującej klienta, który przysłał ten komunikat, a w odpowiedzi wysyła klientowi bieżące wartości jego funkcji aproksymującej, opóźniając odpowiedź o tyle sekund, ile jest małych liter w identyfikatorze gracza, wysyłając komunikat

```
STATE $r_0 ... $r_K\r\n
```

gdzie $r_x = \hat{f}(x)$ jest wartością funkcji aproksymującej w punkcie x .

Nie wolno wysłać komunikatu `PUT` przed otrzymaniem komunikatu `COEFF` i przed odebraniem odpowiedzi na poprzedni komunikat `PUT`. Jeśli serwer wykryje taką sytuację, to wysyła natychmiast komunikat

```
PENALTY $point $value\r\n
```

i dolicza klientowi karę 20 .

W trakcie gry kolejne klienty mogą do niej dołączać.

Koniec gry

Klient może w dowolnym momencie rozłączyć się z serwerem. W takiej sytuacji serwer usuwa wszystkie pamiętane dane związane z tym klientem i kontynuuje działanie, tak jakby tego klienta nigdy nie było. [Dotyczy to także liczenia komunikatów `PUT` poniżej.](#)

Gdy klienty przysła w sumie M poprawnych komunikatów `PUT`, serwer kończy grę, wysyłając do wszystkich klientów komunikat zawierający listę zawierającą identyfikatory graczy i uzyskane przez nich wyniki:

```
SCORING $player_id_1 $result_1 $player_id_2 $result_2 ... \r\n
```

Identyfikatory kolejnych graczy w powyższym komunikacie należy posortować leksykograficznie rosnąco według kodów ASCII. Wyniki graczy są liczbami wymiernymi w rozwinięciu dziesiętnym.

Po wysłaniu komunikatu `SCORING` serwer rozłącza wszystkie klienty, usuwa wszystkie pamiętane dane związane z klientami, odczekuje 1 sekundę i rozpoczyna pracę od początku. **Pozycja w pliku ze współczynnikami *nie* jest**

?

resetowana. Klient po otrzymaniu komunikatu **SCORING** kończy działanie kodem 0.

Jeśli serwer rozłączy się bez wysyłania komunikatu **SCORING**, to klient wypisuje informację diagnostyczną: **ERROR: unexpected server disconnect\n** i kończy działanie z kodem 1.

Strategia działania klienta

Należy zaimplementować dwa tryby działania klienta:

- klient uruchomiony z opcją **-a** implementuje automatyczną strategię gry i sam wysyła polecenia **PUT**; strategia powinna być lepsza niż losowa;
- klient uruchomiony bez opcji **-a** wczytuje ze standardowego wejścia pary liczb, każda para liczb powoduje wysłanie polecenie **PUT**, pierwsza z tych liczb to **\$point**, a druga **\$value**; wczytywanie ze standardowego wejścia nie może blokować komunikacji z serwerem. **Polecenie PUT jest wysyłane od razu po wczytaniu liczb, niezależnie od tego, czy dostaliśmy odpowiedź na poprzedni komunikat PUT.** Jeśli użytkownik wprowadzi liczby przed otrzymaniem komunikatu **COEFF**, to wysyłamy je po otrzymaniu tego komunikatu. **Pary liczb wczytywanych ze standardowego wejścia są rozdzielone pojedynczymi spacjami. Linie kończą się standardowo (znakami \n), a liczby są formatowane tak samo jak liczby wymierne w zadaniu. Jeśli linia nie jest poprawna to wypisujemy komunikat o błędzie: ERROR: invalid input line \$cała_niepoprawna_linia i kontynuujemy przetwarzanie kolejnej linii.**

Obsługa błędów

Programy powinny dokładnie sprawdzać poprawność parametrów, a informację o błędzie powinny wypisywać na standardowe wyjście diagnostyczne, kończąc działanie programu kodem 1.

Błędy związane z wywoływaniem funkcji systemowych lub bibliotecznych uniemożliwiające dalsze działanie programu należy obsługiwać, wypisując na standardowe wyjście diagnostyczne stosowną informację i kończąc działanie programu kodem 1.

Informacje wypisywane na standardowe wyjście diagnostyczne powinny mieć format

```
ERROR: $error_description\n
```

Nie specyfikujemy dokładnie zawartości pola **\$error_description**, ale powinno ono zawierać rozsądny komunikat o błędzie.

Jeśli serwer albo klient dostanie błędny lub nieoczekiwany komunikat **\$komunikat**, to wypisuje na standardowe wyjście diagnostyczne informację

```
ERROR: bad message from [$ip_address]:$port, $player_id: $komunikat\n
```

Pola **\$ip_address** i **\$port** zawierają odpowiednio adres IP i numer portu nadawcy komunikatu. W szczególności, jeśli komunikat zawiera liczbę dziesiętną zapisaną niezgodnie ze specyfikacją z punkcie „Konwencje”, to jest on błędny. **Jeżeli \$player_id nie jest znane, to wstawiamy zamiast niego UNKNOWN.**

Jeśli pierwszy komunikat od klienta nie jest poprawny, serwer po wypisaniu wyżej opisanej informacji diagnostycznej zamyka połączenie z tym klientem. Jeśli pierwszy komunikat od serwera nie jest poprawny, klient po wypisaniu wyżej opisanej informacji diagnostycznej zamyka połączenie z serwerem i kończy działanie kodem 1. W pozostałych przypadkach błędny komunikat, poza wypisaniem informacji o nim, jest ignorowany.

Informacje diagnostyczne

?

Programy wypisują na standardowe wyjście informacje o przebiegu gry.

Serwer

Przykładowo serwer wypisuje na standardowe wyjście następujące informacje:

```
New client [$ip]:$port.  
$ip:$port is now known as $player_id.  
$player_id get coefficients $a_0 ... $a_N.  
$player_id puts $value in $point, current state $r_0 ... $r_K.  
Sending state $r_0 ... $r_K to $player_id.  
Game end, scoring: $player_id_1 $result_1 $player_id_2 $result_2 ...
```

Klient

Przykładowo klient wypisuje na standardowe wyjście następujące informacje:

```
Connected to [$ip]:$port.  
Received coefficients $a_0 ... $a_N.  
Putting $value in $point.  
Received state $r_0 ... $r_K.  
Game end, scoring: $player_id_1 $result_1 $player_id_2 $result_2 ...
```

Dodatkowe wymagania

Serwer obsługuje wiele klientów równolegle. Komunikacja z danym klientem nie powinna blokować komunikacji z innymi klientami. Podłączanie i odłączanie się klientów nie powinno wpływać na komunikację z innymi klientami. Należy dokładnie sprawdzać, czy komunikaty są zgodne z formatem podanym w treści zadania. Oczekiwanie klienta względem serwera i na odwrót określa protokół komunikacyjny. Klient może być testowany z serwerem, który stosuje inne opóźnienia niż ten z treści zadania. Nieprawidłowe działanie jednego klienta nie może wpływać na obsługę innych klientów.

Do obliczeń zmiennoprzecinkowych wystarczy użyć typu `double`. Nie trzeba zajmować się obsługą przekroczenia zakresu.

Rozwiązanie


Rozwiązanie należy zaimplementować w języku C lub C++, korzystając z interfejsu gniazd. Nie wolno korzystać z żadnych innych bibliotek realizujących komunikację sieciową. Programy mają się kompilować i uruchamiać w laboratorium komputerowym zarówno na maszynie students, jak na maszynach w labach.

Programy powinny być napisane zgodnie ze sztuką. Brak oczywistych oczekiwań wobec programów (np. że nie sformatują dysku, czy że wartości zwracane przez funkcje systemowe są sprawdzane) w treści zadania nie oznacza, że program, który ich nie spełnia, będzie uznany za dobry. Również kod programu będzie podlegał ocenie. Protokół będzie testowany rygorystycznie.

Jako rozwiązanie należy przysłać archiwum zawierające pliki niezbędne do zbudowania rozwiązania. Nie wolno załączać plików binarnych i innych zbędnych. Do stworzenia archiwum należy użyć programu `zip`, `rar`, `7z` lub pary programów `tar` i `gzip`. Archiwum powinno mieć odpowiednio rozszerzenie `.zip`, `.rar`, `.7z` lub `.tgz`. Po rozpakowaniu wszystkie pliki powinny znajdować się w katalogu, w którym jest plik archiwum. Archiwum nie może zawierać podkatalogów. Archiwum powinno zawierać plik `makefile` lub `Makefile`. Wykonanie polecenia `make` powinno stworzyć pliki wykonywalne `approx-server` i `approx-client`. Wykonanie polecenia `make clean` powinno usunąć wszystkie pliki powstałe podczas kompilowania.

 [opis.md](#)

16 czerwca 2025, 12:58

Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Ocenione
Pozostały czas	Zadanie zostało przesłane 9 min. 33 sek. przed terminem
Ostatnio modyfikowane	piątek, 6 czerwca 2025, 23:49
Przesyłane pliki	<div><div> kh459048.zip</div><div>6 czerwca 2025, 23:49</div></div>
Komentarz do przesłanego zadania	> Komentarze (0)


Informacja zwrotna

Ocena	4,00 / 4,00
Ocenione dnia	środa, 18 czerwca 2025, 08:52
Ocenione przez	AJ Agata Janowska

Skontaktuj się z nami



Obserwuj nas

 Skontaktuj się z pomocą techniczną

Jesteś zalogowany(a) jako [Krzysztof Hałubek](#) (Wyloguj)

[Podsumowanie zasad przechowywania danych](#)

[Pobierz aplikację mobilną](#)

[Pobierz aplikację mobilną](#)

Motyw został opracowany przez

conecti.me

Moodle, 4.5 LTS | moodle@mimuw.edu.pl