Zadanie dla kandydatów na programistę C/C++ do DSB
Atende Software sp. z o.o.

Proszę napisać program do zarządzania bazą zakresów adresów IPv4 umożliwiający maksymalnie szybkie lookupowanie zawartości bazy.

Spróbuj opracować struktury i algorytm samodzielnie, ponieważ research literatury może zająć zbyt dużo czasu i skierować Ciebie w ślepe uliczki.

Jeśli czas nie pozwala czegoś zaimplementować, opisz szkic/koncepcję lub wskaż problemy, które już wiesz, że wynikną w czasie implementacji; być może porozmawiamy o tym w czasie rozmowy rekrutacyjnej.

Wymagania

Logika

Program wczytuje z STDIN następujące polecenia:

1. „**add p**”, dodanie prefiksu p do struktury (prefix w formacie a.b.c.d/maska_CIDR, np.: 10.1.2.0/19, maska CIDR w zakresie 0..32);
2. (opcjonalnie) „del p”, usunięcie prefiksu z listy;
3. „**check a**”, sprawdzenie, czy podany adres IPv4 (bez maski) zawiera się w liście prefiksów (werdykt: tak/nie oraz dopasowany prefiks podany w formie a.b.c.d/maska_CIDR);
4. (opcjonalnie) „dump”, zwrócenie podglądu zawartości struktury w celach debugowych;
5. (opcjonalnie) „bench x <lista adresów IPv4>”, wykonuje prosty bechmark („check a” dla każdego z adresów na liście, w dowolnej kolejności; powtarza całą taką operację x razy, ale w odróżnieniu od pojedynczego „check” nie zwraca niczego na wyjście)
    a. (opcjonalnie) zwraca na STDOUT wynik w postaci liczby milisekund, które trwała całość benchmarku;
6. **EOF** kończy pracę.

Założenia

1. Prefiksy IPv4 muszą być przechowywane w pamięci operacyjnej w strukturze danych pozwalającej **na jak najszybsze wyszukiwanie** (krytyczna jest wydajność „bench” i to głównie oceniamy!).
2. Dodawanie i usuwanie może być powolne gdyż będzie wykonywane bardzo rzadko.

3. Program jest jednowątkowy, jednoprocesowy i będzie działał na dedykowanym mu rdzeniu procesora.
4. Spodziewajmy się, że:
    a. do struktury typowo trafi około 1 milion prefiksów IPv4 o różnych maskach, przy czym dominują sieci o maskach z zakresu /16../24;
    b. zapytania „check" będą o różne adresy IP (rozkład nie jest idealnie losowy, ale pesymistycznie można się takiego spodziewać);
    c. iteracji „bench" do wykonania mogą być miliardy.
5. **Wybór takiej a nie innej struktury lub optymalizacji musi być poparty krótkim uzasadnieniem.** Jeśli widzisz kilka interesujących rozwiązań/wykluczających się koncepcji, dla oszczędności czasu zdecyduj się na jedno i krótko (kilka punktów) opisz pozostałe z potencjalnymi zaletami i wadami każdego z podejść (nie ma jedynie słusznego rozwiązania tego problemu i może on mieć wiele sprytnych optymalizacji). Rezygnacja ze wsparcia niektórych komend może również radykalnie uprościć zadanie, warto wskazać, w jaki sposób.
6. Interfejs musi opierać się o standardowe wejście/wyjście przykrywające odpowiadające mu metody API. Polecenia i dane są podawane w formie tekstowej.
7. Program powinien zostać napisany w C/C++ (może zawierać również wstawki ASM lub SIMD) z użyciem tylko biblioteki standardowej, i kompilować się za pomocą GCC w wersji 7 lub nowszej na systemach GNU/Linux (mile widziany będzie prosty Makefile oraz świadomie dobrane opcje).
8. Program działa pod GNU/Linux na platformie Intel Broadwell i ma dostępne 16GB czterokanałowej pamięci RAM, dopuszczamy aby był pod nią optymalizowany (np. przez użycie rozkazów specyficznych dla tej architektury).

Użycie

Przykład sekwencji poleceń (obowiązkowych jak i opcjonalnych – jeśli Kandydat ma chęć i czas) i ich skutków:

```
add 10.0.0.0/8 – w efekcie dodano prefiks 10.0.0.0/8

add 10.0.0.0/8 – zwraca porażkę, nie dodano prefiksu, gdyż już był wcześniej dodany

add 10.0.0.0/24 – w efekcie dodano prefiks 10.0.0.0/24, gdyż zawiera się wewnątrz 10.0.0.0/8 (ma dłuższą maskę i nie konfliktuje)

add 10.0.1.99/24 – w efekcie dodano prefiks 10.0.1.0/24, gdyż odcięto część hosta i prefiks zawiera się wewnątrz 10.0.0.0/8

add 10.0.0.d/24 – w efekcie nie dodano prefiksu gdyż niepoprawna postać prefiksu

add 10.0.0.3 – w efekcie nie dodano prefiksu gdyż niepoprawna postać prefiksu (brak maski)
```

check 10.255.255.255 – zwraca sukces i prefiks 10.0.0.0/8 jako najlepsze dopasowanie na liście

check 10.0.0.1 – zwraca sukces i prefiks 10.0.0.0/24 jako najlepsze (najdłuższe) dopasowanie (mimo że pasuje również do 10.0.0.0/8)

del 10.0.0.0/8 – w efekcie usunięto prefiks 10.0.0.0/8, ale pozostawiono prefiksy o dłuższej masce, które zawierały się w tej podsieci

del 192.168.0.0/32 – porażka, takiego prefiksu nie ma na liście więc nie da się go usunąć

check 10.255.255.255 – zwraca porażkę, adres nie zawiera się w żadnym z prefiksów

check 10.0.0.0/24 – zwraca porażkę, argument nie jest adresem IPv4

show – zwraca czytelną zawartość struktury danych w pamięci

bench 1000000 1.2.3.4 5.6.7.8 2.3.4.5 6.7.8.9 – wykonuje milion razy zapytania o adresy ze zbioru {1.2.3.4, 5.6.7.8, 2.3.4.5, 6.7.8.9}, adresy mogą być wybierane w dowolnej kolejności; natomiast NIE robi miliona sprawdzeń 1.2.3.4, potem miliona sprawdzeń 5.6.7.8 itd.

Przykłady użycia programu:

ręcznie

$ **./p**

Please type add/del/check/show/bench command.

add 10.0.0.0/8

OK

add 10.0.0.0/8

ERROR (optional reason: prefix already exists)

bench 100000000 10.0.0.1 10.0.0.2 … // tutaj lista np. 1k adresów

31337 // ile milisekund trwał benchmark

^D

$

wsadowo

cat large-command-file | time ./p >p.log 2>p.err