Programming assignment 1

Getting started

All your submissions should be implemented in Go unless the problem specifies something different. You can download Go at https://go.dev/dl/. You can also use various package managers to install it, e.g., HomeBrew on macOS.

Problem 1

Implement a binary heap that multiple goroutines can safely use. It is ok to use coarse-grained locking, i.e., lock the heap in each method that accesses it.

Problem 2

Implement a deque using a linked list that multiple goroutines can safely use. Your locking strategy should be as fine-grained as possible (but you must use mutex locking). Describe the strategy in the report and argue for why it is minimal.

Problem 3

You know that the string `a74277500228f7b4cfa8694098443fc5` is an md5 hash of a password. You also know that the password is six characters; the only characters allowed are `a-z0-9`. Your task is to create a parallel password cracker in Go.

You can use `crypto/md5` to create an md5-hash of a string. You have cracked the password when you find a string that generates the same md5-hash as the one above.

Your solution should use goroutines and channels. Describe your design in the report as well as benchmarks of how well it scales with an increasing number of goroutines.

Problem 4 (required for C+)

A mutex lock or a binary semaphore can be used to allow a single thread or goroutine to access a critical region. A counting semaphore can allow up to N threads or goroutines to access a critical region. Implement a counting semaphore using mutex locks. N should be a parameter. Describe your design in the report and provide a simple example showing it works.

Problem 5 (required for C+)

Modify the heap in problem 1 to use a reader/writer lock (`RWMutex`). The methods should be modified to use the right type of lock/unlock operations.

Du vet att strängen `a74277500228f7b4cfa8694098443fc5` är en md5-hash av ett lösenord. Du vet också att lösenordet är 6 tecken och att de tecken som används är `a-z0-9`. Din uppgift är att skapa en parallell lösenordsknäckare i Python. Om 6 tecken tar för lång tid, kan du använda följande sträng som är 5 tecken lång: `755afdd46a18a25bd85ddd4004d5cfea` ("hej11").

Du kan skapa en md5-hash från en sträng med `hashlib.md5`, t.ex. `hashlib.md5(b'abc')`. Tänk på att du måste skicka in en en byte-sträng, så använd `encode` på strängar du skapar, t.ex. `'abc'.encode()`. Du kan komma åt md5-hashen med hjälp av `hexdigest`, t.ex. `hashlib.md5(b'abc').hexdigest()`.

När du hittar en sträng som genererar samma md5-hash som den givna strängen har du knäckt lösenordet. Din uppgift är alltså en att skriva ett program som testar alla kombinationer av gemener och siffror av längd 6 tills du hittar rätt lösenord. Då lösenordet är en kombination av 26 bokstäver och 10 siffror måste du alltså i värsta fall testa 36^6 kombinationer.

Använd processer och trådar för att snabba upp sökningen. Skapa en version som använder ett antal processer och en som använder ett antal trådar. Du kan t.ex. använda en `concurrent.futures.Executor` (se Föreläsning 1).

Write a program that simulates that you are rolling two dice 10 000 times. At the same time, keep track of the number of times you get the result (adding the dice values) 2, 3, ..., 11, 12. (Use a list to store a count of the numbers.) After the simulation, present the frequencies for the different numbers.

Problem 2

A random walk is a sequence of steps in some enclosed plane, where the direction of each step is random. The walk terminates when a maximal number of steps have been taken or when a step goes outside the given boundary of the plane.

For this task, assume a plane given by a grid, with the point $(0, 0)$ at the center. The size of the plane is given by an integer; if the given integer is k, then the values of the x and y coordinates can vary from $-k$ to k. Each step will be one unit up, one unit down, one unit to the right, or one unit to the left (no diagonal movements).

Problem 3

Implement a generic BST. Your implementation should use structs and methods on those. Your BST should at least support adding, removing, and finding values. It should also support at least one way to walk the tree and apply a function that is passed as a parameter on each node.

Problem 4

Implement Quicksort on slices.

Submission guidelines

Submit your solutions as a single zip file via Moodle no later than 17:00 on February 16, 2024 (cutoff 08:00 February 19). This is a group assignment that can be done in groups of one or two students. Your submission should contain well-structured and organized Go code for the problems with a README.txt (or .md) file that describes how to compile and run the Go programs.