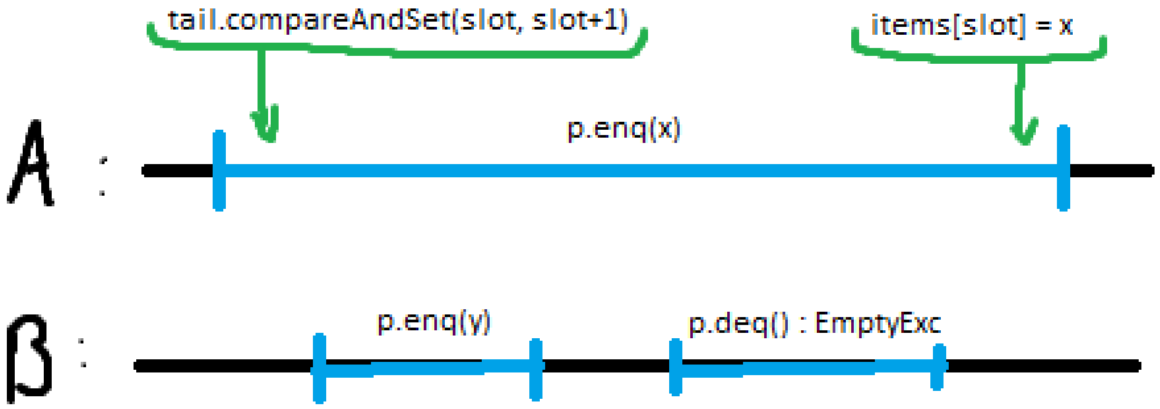
Zad. 2

Aby sprawdzić niepoprawność, pokażę przykład historii, która jest nielinearyzowalna.



H=G, bo nie ma oczekujących wywołań

G:
A: p.enq(x)
B: p.enq(y)
B: p: void
B: p.deq()
B: p: Empty Exception
A: p: void

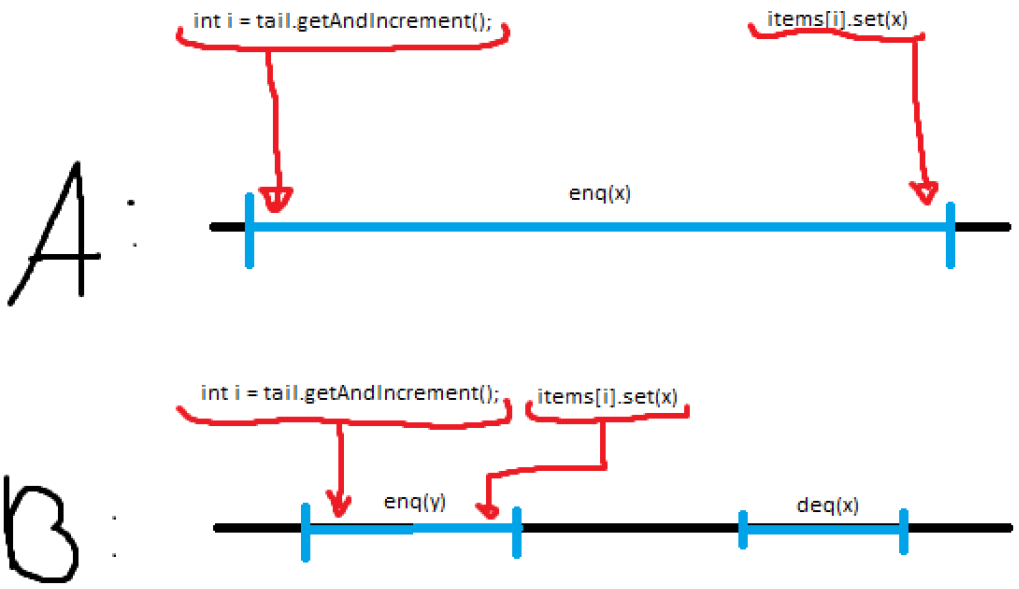
S:
(...)
B: p.enq(y)
B: p: void
(...)
B: p.deq()
B: p: Empty Exception
(...)

Zamiast któregoś (...), trzeba wstawić:
A: p.enq()
A: p: void

Jednak, nie ważne gdzie to zrobimy to można zauważyć, że S nie będzie legalną sekwencyjną historią, ponieważ nie spełnia specyfikacji IQueue (dostajemy wyjątek Empty Exception dla pustej kolejki p), czyli G nie odpowiada żadnej legalnej sekwencyjnej historii S, a co za tym idzie H = G jest nielinearyzowalne.

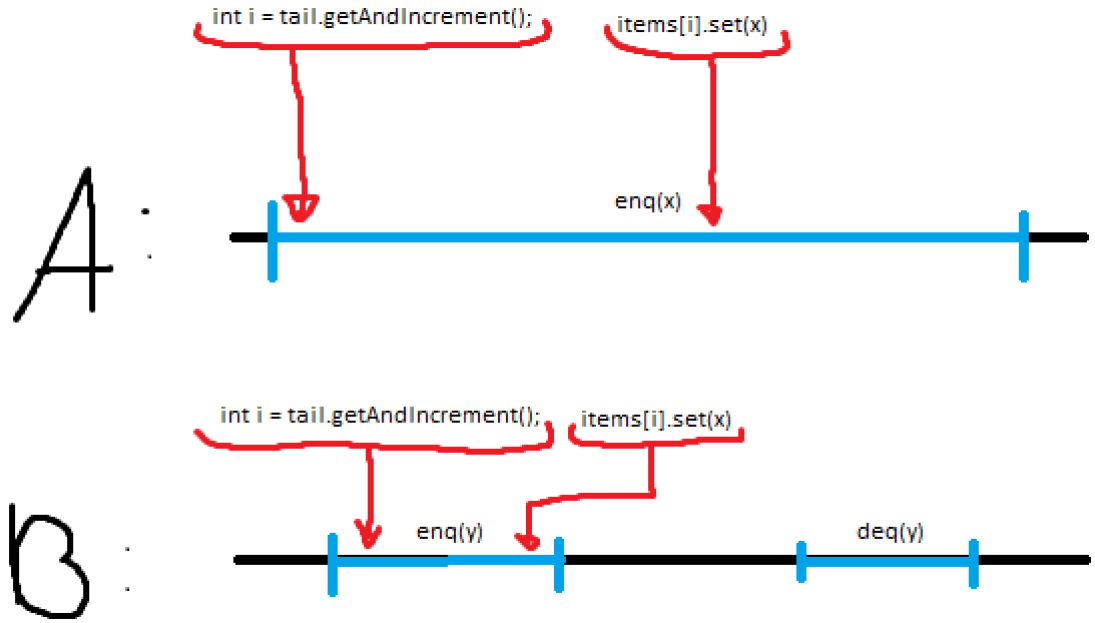
Zad. 3

int i = tail.getAndIncrement();



Przy linearyzacji względem pierwszego enq, oczekiwalibyśmy, że deq zwróci x, jednak w momencie wywołania deq tego x nie ma jeszcze w kolejce, dlatego zostanie wyciągnięte y, a to jest sprzeczne z naszą linearyzacją.

items[i].set(x)



Przy takiej linearyzacji, względem drugiego enq, oczekiwalibyśmy, że deq zwróci y, jednak miejsce przypisania dla x zostało wybrane wcześniej niż dla y, więc dostaniemy x, co jest sprzeczne z naszą linearyzacją.

Zatem enq() nie ma pojedynczego punktu linearyzacji, co nie oznacza, że nie jest linearyzowana, ponieważ punkt może być różny dla każdego wywołania funkcji.

Zad. 5

Nieczekanie – każda metoda kończy się po skończonej liczbie kroków.
Niewstrzymanie – cały system dokonuje postępu, ale dany wątek nie musi.
Nieblokujące – wstrzymanie jednego wątku, nie blokuje pozostałych.
Niezależne – gwarantują postępy niezależnie od wywoływanych wątków.

Niezakleszczenie i niezagłodzenie są blokujące, ze względu na możliwości blokady innych wątków przez jeden, a są zależne, ponieważ zależą od przydzielania czasu przez system.

Współbieżna metoda w nietrywialny sposób wykorzystująca zamki nie może być nieczekająca, ponieważ może dochodzić do sytuacji, gdzie jeden wątek chce zamknąć zamek zajęty przez inny. To sprawia, że musi on czekać, co może spowodować nieokreślone opóźnienia.

Metoda weird(), której i-te wywołanie wraca po wykonaniu 2^i instukcji:

a) Jest nieczekająca, ponieważ każde wywołanie kończy się po określonej, skończonej liczbie kroków.
b) Nie jest nieczekająca z limitem kroków, ponieważ 2^i nie jest stałą liczbą.

Zad. 6

```
public class SimRegister64 {

  private int low;

  private int high;


  public long read() {

    result = high;

    result =  (result << 32) | (low & 0xFFFFFFFFL);

    return result;

  }

  public void write(long value) {

    low = (int) (value & 0xFFFFFFFFL);

    high = (int) (value >> 32);

  }

}
```

Rejestr jest bezpieczny, ponieważ jeśli zapis na pokrywa się z odczytem to odczyt będzie zwracał ostatnią zapisaną wartość, a w.p.p. zwróci wartość w dozwolonym zakresie.

Nie jest regularny, ponieważ dwa wątki mogą pisać w tym samym czasie, co by mogło sprawić, że następny odczyt zwróciłby niezapisane dane.

Ponieważ nie jest regularny to nie będzie też atomowy.

Zad. 7

```
public void lock() {
  flag[i] = true;
  victim = i;
  while (flag[j] && victim == i) {}
}


public void unlock() {
  flag[i] = false;
}
```

1. **Utrata wzajemnego wykluczania**: Przy rejestrach regularnych, odczyt flag może nie być spójny, gdy drugi proces jednocześnie zmienia wartość swojego flag.
   Przykład:
   a) Proces 0 ustawia flag[0] = true, ale zanim Proces 1 to zobaczy, odczytuje jeszcze starą wartość flag[0] = false.
   b) W efekcie oba procesy mogą uznać, że mają prawo wejść do sekcji krytycznej, co łamie zasadę wzajemnego wykluczania.
2. **Ryzyko zagłodzenia**: Algorytm Petersona gwarantuje brak zagłodzenia, ponieważ procesy naprzemiennie ustawiają victim, która dalej jest atomowa, więc każdy proces ma ostatecznie szansę wejścia do sekcji krytycznej.