ASD 2014/2015

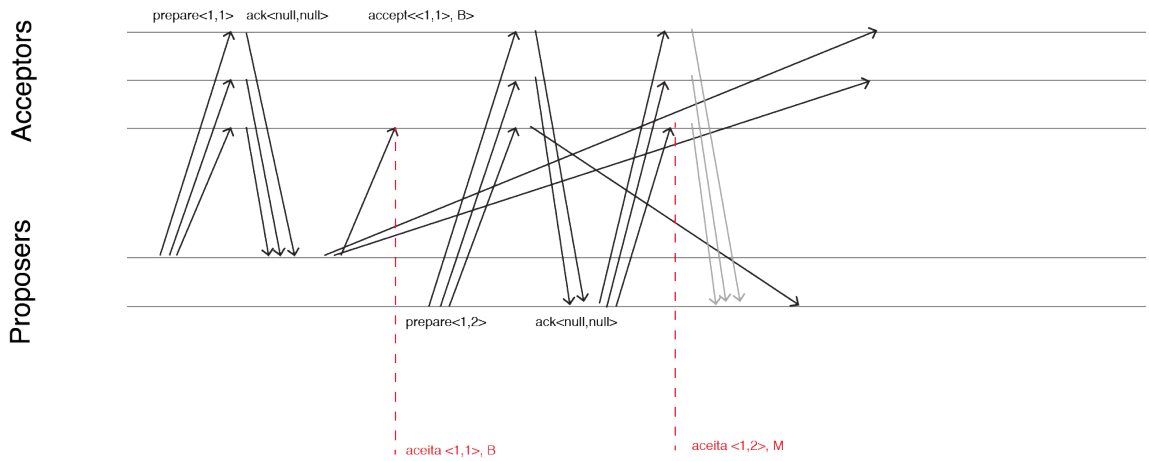
Test 2, 2013

Questão 1.

(a)

```
n = 2f + 1
n = 7 => f = (7-1)/2 = 3
```

(b)



(c)

Formal

Não => supor (absurdo) que era possível.

```
Existem 2 maiorias, M1, M2
M1 aceita <x,B>
M2 aceita <y,M>
--
Supor y > x
y -> fez prepare (M3), não retornou banana
M1 \int M3 ≈ 0
Existe um acceptor a:
    - ou já aceitou B: teria de retornar B no prepare
    - ou ainda não aceitou B: prometeu não aceitar nada com n° seq ≤ y
```

(d)

Mantém-se tudo igual. O `acceptor` apenas dá falsa esperança a quem quer propor algo inferior ao `n_h`

que pode avançar porque existem hipóteses de o valor que ele quer propôr seja aceite. No entanto, ele já prometeu não aceitar nada menor que `n_h`, condenando a proposta ao fracasso.

- Validade: Sim
- Acordo: Sim
- Integridade: Sim

Caso fosse o caso `accept`:

```
acceptor accept(n,v) handler:
    if n >= n_h
        n_a = n
        v_a = v
    reply accept_ok(n)  // linha fora do if, é sempre respondido accept_ok
```

São decididos valores errados, porque é sempre respondido `accept_ok`.

- Validade: Sim (trivialmente satisfeito, porque se todos propuserem o mesmo, isso será decidido)
- Acordo: Não (Poderão haver decisões diferentes)
- Integridade: Sim (trivialmente satisfeito porque não são inventados valores)

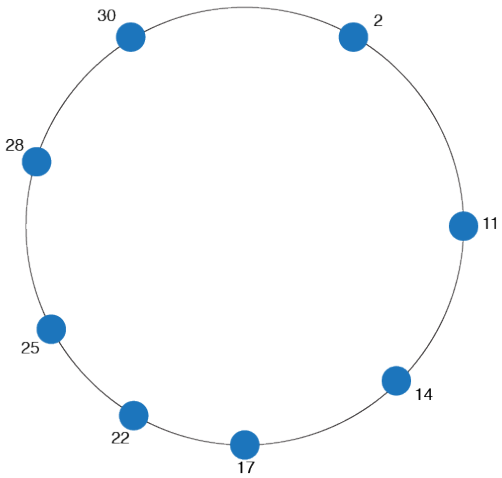
(e)

(empty)

(f)

- **AC1**: Sim, pois o Paxos resolve consenso (mesma propriedade de acordo)
- **AC2**: Não, pois pode acontecer `p1` propõe `sim`, `p2` propõe `não` e ultimamente ambos decidirem `sim` (`COMMIT`).
- **AC3**: Sim, Paxos é mais forte: se todos têm `sim`, só pode ser escolhido `sim` (`COMMIT`).

Questão 2.



(o professor trocou o `15` pelo `14` na aula, para não haver colisões)

(a)

Tabela de *fingers* do nó `25` :

O finger `i` é dado por `succ( (25 + 2^{i-1}) % 2^m ) , i = 1..m`

`i`	`finger[i]`
1	`succ((25 + 1) % 32 ) = succ(26) = 28`
2	`succ((25 + 2) % 32 ) = succ(27) = 28`
3	`succ((25 + 4) % 32 ) = succ(29) = 30`
4	`succ((25 + 8) % 32 ) = succ(1) = 2`
5	`succ((25 + 16) % 32 ) = succ(9) = 11`

(b)

`25.lookpup(20)`

Maior *finger* que precede `20`: `finger[5] = 11`

Tabela de *fingers* do nó `11` :

`i`	`finger[i]`
1	`succ((11 + 1) % 32 ) = succ(12) = 14`
2	`succ((11 + 2) % 32 ) = succ(13) = 14`
3	`succ((11 + 4) % 32 ) = succ(15) = 17`
4	`succ((11 + 8) % 32 ) = succ(19) = 22`
5	`succ((11 + 16) % 32 ) = succ(27) = 28`

Maior *finger* que precede `20`: `finger[3] = 17`

Como a chave se situa entre o `17` e o seu sucessor, `22`, paramos.

`25.loopkup(20) = 25 -> 11 -> 17 -> 22`

(c)

Os subscritores deverão fazer o lookup da fonte. União dos caminhos inversos do lookup especifica a árvore de disseminação.

`25.loopkup(20) = 25 -> 11 -> 17 -> 22`

`2.loopkup(20) = ?`

Tabela de *fingers* do nó `2` :

`i`	`finger[i]`
1	`succ((2 + 1) % 32 ) = succ(3) = 11`
2	`succ((2 + 2) % 32 ) = succ(4) = 11`
3	`succ((2 + 4) % 32 ) = succ(6) = 11`
4	`succ((2 + 8) % 32 ) = succ(10) = 11`
5	`succ((2 + 16) % 32 ) = succ(18) = 22`

Maior *finger* que precede `20`: `finger[5] = 11`

Nota: muito embora seja o 22 (finger[5]) o responsável pelo 20, não podemos ir directos porque a tabela de fingers pode estar desactualizada.

Então,

`2.loopkup(20) = 25 -> 11 -> 17 -> 22`

Árvore P2P:

```
                  /------- 2
20 ---- 17 ----- 11
                  \------- 25
```

Questão 3.

3.1

(a)

C

(b)

C

(c)

A+P

3.2

Usando a alínea (c):

Dois users, `u1` e `u2` estão a consultar o feed de um user `u3`. `u1` está na Europa e `u3` está nos EUA.

Com um sistema como o Dynamo, pode acontecer que num dado instante, `u1` esteja a ver um tweet que o `u2` ainda não viu. Tal vai s

3.3