HW13

17.6

the graph is acyclic， so it can obtain a serializable schedule

T1->T2->T4->T3->T5; or T1->T2->T3->T4->T5

17.7

cascadeless schedule use simple description, we need commit our changes to one data before another transactions use this data. It is worse for less concurrency than noncascadeless , but if a transaction is abort, it will not cause any other transaction So if failures occur rarely, we can use noncascadeless schedules for the increased concurrency

18.1

Assume the following cycle exists in the precedence graph: T0 → T1 → T2 → ... → T_{n-1} → T_0. Let a_i be the time at which T_i obtains its last lock. Then for all transactions such that $T_i \rightarrow T_j$, a_i < a_j . Then for the cycle we have

$a_0 < a_1 < a_2 <\ldots < a_{n-1} < a_0$

the lock point ordering of the transactions is also a topological sort ordering of the precedence graph. Thus transactions can be serialized according to their lock points.

18.7

a

Firstly we need to show 2PL can ensure serializability, assume that there exists a schedule :

$T_0, T_1, \ldots, T_{n-1},$

which obtain by 2PL.

If there is non-serializability , then exist a circle

$T_0 - > T_1, \ldots, T_r - > T_0,$

let a_i be the t_i check point, so we can obtain $a_o < a_r$

so 2PL can obtain a seriablity schedule

So, Lock-I is similar to Lock-S, because Lock-S,Lock-X is heritability So, Lock-S,Lock-I,and Lock-X is also seriability by check lock points.

b

it's obvious that know that increments mode cam creased concurrency Lcok-I allow multi-transcation get the Lock at the same time, if we don't have this lock, we must use Lock-X to write the value exclusive.

18.18

1. implement more easy
2. support rollback
3. allow concurrency