now the problem

insert into empty B+ tree with order d=2 the values:

20, 40, 37, 27, 25, 30, 15, 60, 42, 35, 5, 22, 50, 11, 32, 36

recall with order d=2

- leaf node can have 2-4 keys, with 2-4 child pointers
- internal node can have 2-4 keys with 3-5 child pointers
- the root is an exception: it can have a single key
- we are using the B+ tree "rules" from our text
- so on search, "ties" go to the right
- other texts **will** have slightly different rules so beware

starting to insert

after 20:

(20)

after 40:

(20, 40)

after 37:

(20, 37, 40)

after 27:

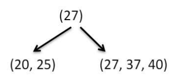
(20, 27, 37, 40)

the first split

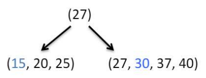
after 25 overflow, so need to split :

(20, 25, 27, 37, 40)

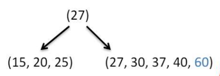
splits to

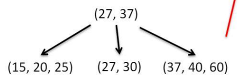


then inserting 30 and 15 is boring:

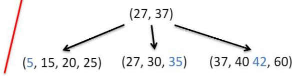


after 60 overflow again:

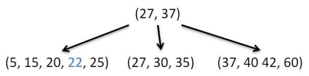


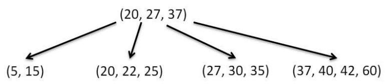


then inserting 42, 35, 5 is simple:

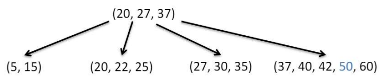


inserting 22 causes a split:

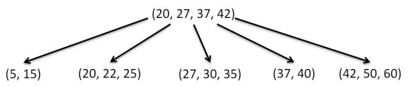




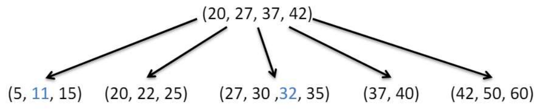
50 also causes a split:



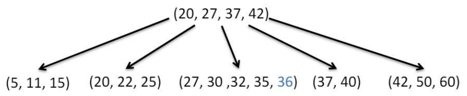
becoming



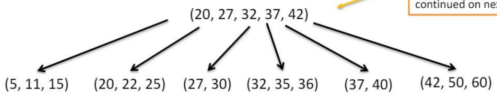
inserting 11 and 32 again simple



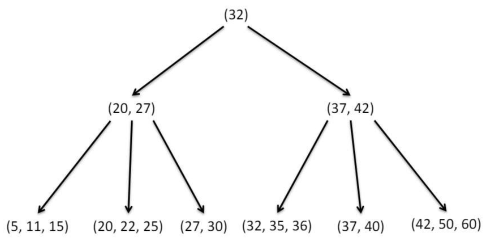
now inserting 36 will cause two splits



32 is promoted into parent, but then the root node overflows, continued on next page



final tree



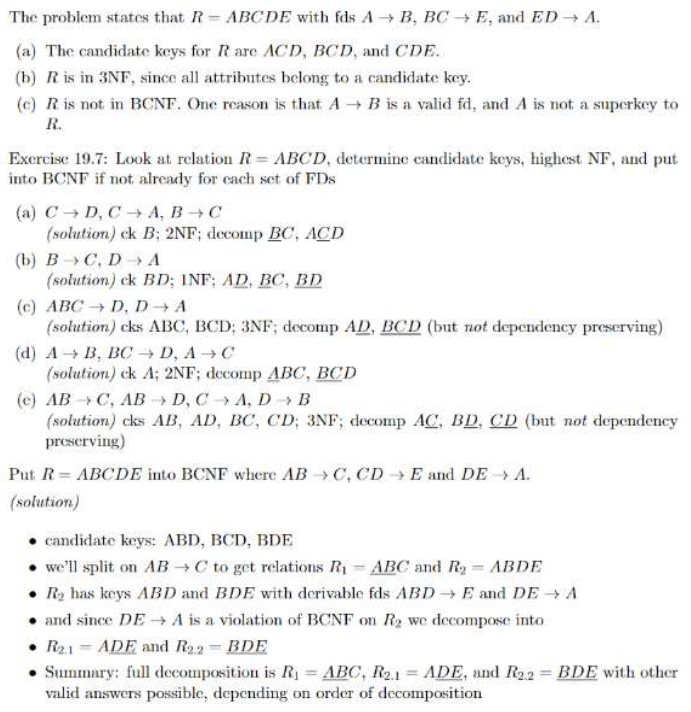
note
- the root may have only one key (and two children)
- the leaf nodes are linked together (not shown here)

- 20,000 records, each with a key to go into dense index
- key is 40 bytes
- pointer 10 bytes
- page is 1000 bytes

For degree d a full (internal and leaf) node has $2d$ keys and $2d+1$ (or $2d+2$) pointers. Largest d can be is solution to $40 \cdot 2d + 10 \cdot (2d+1) = 100d + 10 \leq 1000$ (node must fit in page), or $d = 9$. For a 70% full node, $100d + 10 \leq 700$, or $d = 6$. Thus a full node has $2 \cdot d = 18$ keys and a 70% full node has 12 keys.

Part 1: There will be $l_0 = \lceil \frac{20000}{18} \rceil = 1112$ leaf nodes. An internal node has 19 children, so the lowest internal level will have $l_1 = \lceil \frac{1112}{19} \rceil = 59$ nodes. Next up the tree, level 2 has $l_2 = \lceil \frac{59}{19} \rceil = 4$ nodes, and level 3 has $l_3 = \lceil \frac{4}{19} \rceil = 1$ node, which is the root. So there are 4 levels in the tree, one leaf and three internal.

- 1NF = all entries are atomic (all entries contain one piece of information; can't be broken up; no sets, subfields in an attribute); 2NF = remove partial dependencies; 3NF = remove transitive dependencies; BCNF = like 3NF but deal with problems from multiple primary keys
- Functional Dependency: is taken to mean that the value of B depends on the value of A, in other words, if any two rows in a table have the same A entry, they must always have the same B entry (A->B; zipcode->state; related to concept of keys)
- Partial Dependency: a full functional dependency X->Y is the case when removing anything from X will cause the dependency to not hold (irreducibly dependent; X and Y can be sets of attributes)
- R is in 2NF if every attribute not belonging to the primary key is fully functionally dependent on the primary key (no partial dependencies; STUDENT_ENROLL: ssn, crn, name, bdate with partial dependency ssn->name, bdate)
- R is in 3NF if it is in 2NF and has no transitive dependencies (SUPPLIER: s#, city, status; FDs: s#->city,state, city->status; in other words, the statuse of the supplier depends only on the city they are in city->status is a transitive dependency and thus SUPPLIER is not in 3NF)
- Armstrong's Axioms (let X, Y, Z be a set of attributes):
  - (reflexivity) if X⊆Y, then X → Y
  - (transitivity) if X → Y and Y→ Z, then X → Z
  - (augmentation) if X → Y, then XZ → YZ
- A relation R is in 3NF if for all fds alpha → beta ∈ F^+ that hold on R at least one of the following conditions is true
  - alpha → beta is trivial
  - alpha is a superkey for R, or
  - each attribute A ∈ beta belongs to a candidate key of R
- SUPPLIER: s#, city, status (FDs: s#->city,state, city->status)
- CK (candidate keys): s#
- 3NF violation: city→ status
  - it is not trivial
  - city is not a superkey
  - status does not belong to any candidate key
- • So SUPPLIER is not in 3NF according to the formal definition either
- R: student, prof, dept
  - F: p→ d, sd → p
  - CK: sd, sp
  - p → d is a violation of BCNF
    - not trivial
    - p not a superkey of R
  - BCNF decomposition = R1:(student, prof); R2: prof, dept
- a **superkey** is a unique identifier (note that it may not be minimal)
  - CAR: license, make, model, year, vin
  - license is a superkey, so is (license, make) and (license, vin, year)
- a **candidate key** is a minimal superkey
  - CAR has two candidate keys: 1. license 2. vin
- the **primary key** is a designated candidate key
  - any candidate key which is not the primary key is an alternate key
  - alternate keys can be identified/enforced using the UNIQUE keyword in SQL
- If no incoming edges/arrows = CK



The problem states that $R = ABCDE$ with fds $A \to B$, $BC \to E$, and $ED \to A$.

(a) The candidate keys for R are ACD, BCD, and CDE.

(b) R is in 3NF, since all attributes belong to a candidate key.

(c) R is not in BCNF. One reason is that $A \to B$ is a valid fd, and A is not a superkey to R.

. Exercise 19.7: Look at relation $R = ABCD$, determine candidate keys, highest NF, and put into BCNF if not already for each set of FDs

(a) $C \to D, C \to A, B \to C$
(solution) ck B; 2NF; decomp $\underline{B}C$, $A\underline{C}D$

(b) $B \to C, D \to A$
(solution) ck BD; 1NF; $A\underline{D}$, $\underline{B}C$, $\underline{BD}$

(c) $ABC \to D, D \to A$
(solution) cks ABC, BCD; 3NF; decomp $A\underline{D}$, $\underline{BCD}$ (but *not* dependency preserving)

(d) $A \to B, BC \to D, A \to C$
(solution) ck A; 2NF; decomp $\underline{A}BC$, $\underline{BC}D$

(e) $AB \to C, AB \to D, C \to A, D \to B$
(solution) cks AB, AD, BC, CD; 3NF; decomp $A\underline{C}$, $B\underline{D}$, $\underline{CD}$ (but *not* dependency preserving)

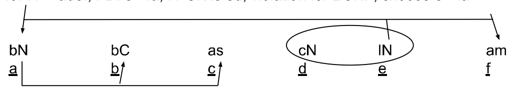
. Put $R = ABCDE$ into BCNF where $AB \to C$, $CD \to E$ and $DE \to A$.

(solution)

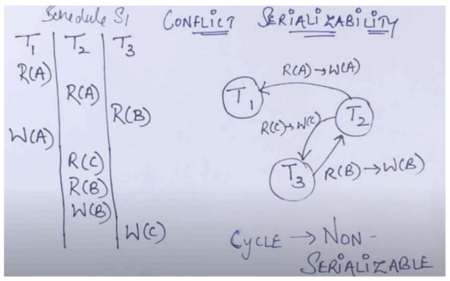
- candidate keys: ABD, BCD, BDE
- we'll split on $AB \to C$ to get relations $R_1 = \underline{AB}C$ and $R_2 = ABDE$
- R_2 has keys ABD and BDE with derivable fds $ABD \to E$ and $DE \to A$
- and since $DE \to A$ is a violation of BCNF on R_2 we decompose into
- $R_{2.1} = A\underline{DE}$ and $R_{2.2} = \underline{BDE}$
- Summary: full decomposition is $R_1 = \underline{AB}C$, $R_{2.1} = A\underline{DE}$, and $R_{2.2} = \underline{BDE}$ with other valid answers possible, depending on order of decomposition

convert the relation into BCNF (if it is not already). Be sure to identify all candidate keys
relation LENDING with attributes branchName, branchCity, assets, customerName, loanNumber, amount and with FDs
branchName → branchCity assets
loanNumber → amount branchName

Answer: <u>CK:</u> cN, lN (de); FDs: a->bc, e->fa; BCNF: no (neither a or e is a superkey; neither FD is trivial); abcdef to adef/abc; choose a->bc
for R' = adef, FD: e->fa; R' CK is ed; violation for BCNF; choose e->fa



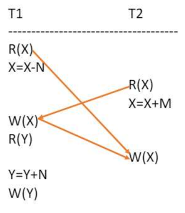
- we say two schedules are **conflict equivalent** ($S_1 \approx S_2$) if they involve the same set of actions on the same xacts and they order every pair of conflicting actions in the same way
- a schedule is **conflict serializable** if it is conflict equivalent to some serial schedule
- <u>If no cycle, then serializable; if cycle then non-serializable</u>



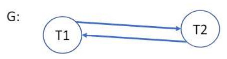
test for conflict serializability

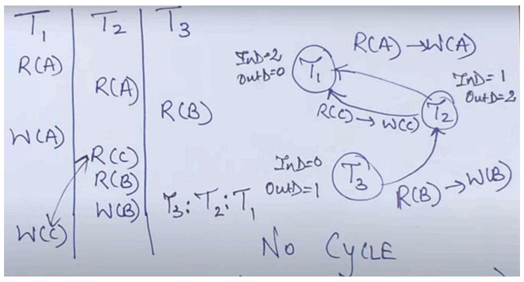
algorithm:

1. build a conflict graph
   - look at each pair of xacts Ti and Tj
   - draw an edge from Ti to Tj if
     - both xacts act (R/W) on the same data item X
     - one action is a W
     - Ti does it before Tj
2. test to see whether the graph G has a cycle
   - if cycle, "NO"
   - else "YES"



T1
T2
R(X)
X=X-N
R(X)
X=X+M
W(X)
R(Y)
W(X)
Y=Y+N
W(Y)





T1
T2
T3
R(A)
R(A)
R(B)
W(A)
R(C)
R(B)
W(B)
W(C)
T3: T2: T1
InD=2
OutD=0
R(A)→W(A)
InD=1
OutD=2
R(C)→W(C)
InD=0
OutD=1
R(B)→W(B)
T1
T2
T3
No CYCLE