Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

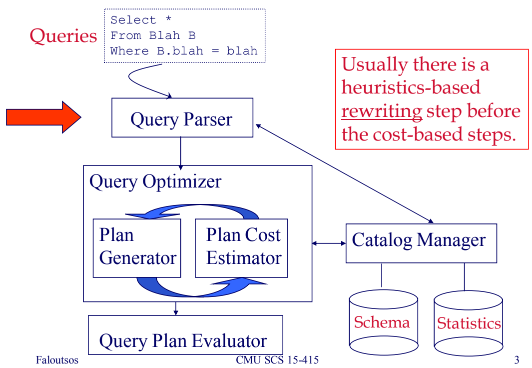
Query Optimization

(R&G ch. 15; Sys. R q-opt paper)

Overview - detailed

- Why q-opt?
- Equivalence of expressions
- Cost estimation
- Plan generation
- Plan evaluation

Cost-based Query Sub-System



Why Q-opt?

- SQL: ~declarative
- good q-opt -> big difference
  - eg., seq. Scan vs
  - B-tree index, on P=1,000 pages

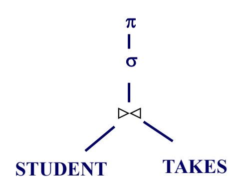
Q-opt steps

- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best

Q-opt - example

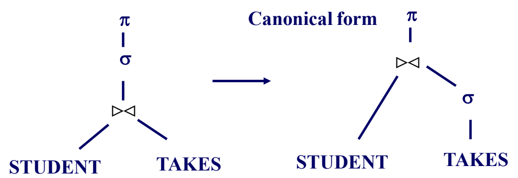
```
select name
from STUDENT, TAKES
where c-id='415' and
STUDENT.ssn=TAKES.ssn
```

→



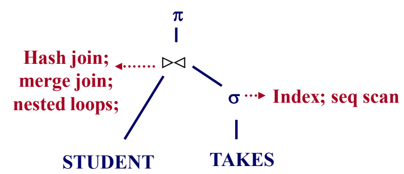
CMU SCS

Q-opt - example



CMU SCS

Q-opt - example



CMU SCS

Overview - detailed

- Why q-opt?
- **Equivalence of expressions**
- Cost estimation
- ...

CMU SCS

Equivalence of expressions

- A.k.a.: syntactic q-opt
- in short: perform selections and projections early
- More details: see transf. rules in text

CMU SCS

Equivalence of expressions

- Q: How to prove a transf. rule?

  $\sigma_P(R1 \bowtie R2) \stackrel{?}{=} \sigma_P(R1) \bowtie \sigma_P(R2)$
- A: use RTC, to show that LHS = RHS, eg:

  $\sigma_P(R1 \cup R2) \stackrel{?}{=} \sigma_P(R1) \cup \sigma_P(R2)$

CMU SCS

Equivalence of expressions

$\sigma_P(R1 \cup R2) \stackrel{?}{=} \sigma_P(R1) \cup \sigma_P(R2)$

$t \in LHS \Leftrightarrow$

$t \in \quad (R1 \cup R2) \wedge P(t) \quad \Leftrightarrow$

$(t \in \quad R1 \quad \vee \quad t \in R2) \wedge P(t) \quad \Leftrightarrow$

$(t \in \quad R1 \wedge P(t)) \quad \vee \quad (t \in R2) \wedge P(t)) \quad \Leftrightarrow$

CMU SCS

Equivalence of expressions

$$\sigma_P(R1 \cup R2) \stackrel{?}{=} \sigma_P(R1) \cup \sigma_P(R2)$$

...

$$(t \in R1 \wedge P(t)) \quad \vee \quad (t \in R2) \wedge P(t)) \quad \Leftrightarrow$$

$$(t \in \sigma_P(R1)) \quad \vee \quad (t \in \sigma_P(R2)) \quad \Leftrightarrow$$

$$t \in \sigma_P(R1) \cup \sigma_P(R2) \quad \Leftrightarrow$$

$$t \in RHS$$

$$QED$$

Faloutsos CMU SCS 15-415 13

CMU SCS

Equivalence of expressions

- Q: how to disprove a rule??

$$\pi_A(R1 - R2) \stackrel{?}{=} \pi_A(R1) - \pi_A(R2)$$

Faloutsos CMU SCS 15-415 14

CMU SCS

Equivalence of expressions

- Selections
  - perform them early
  - break a complex predicate, and push

$$\sigma_{p1\wedge p2\wedge ... pn}(R) = \sigma_{p1}(\sigma_{p2}(...\sigma_{pn}(R))...)$$

  - simplify a complex predicate
    - ('X=Y and Y=3') -> 'X=3 and Y=3'

Faloutsos CMU SCS 15-415 15

CMU SCS

Equivalence of expressions

- Projections
  - perform them early (but carefully…)
    - Smaller tuples
    - Fewer tuples (if duplicates are eliminated)
  - project out all attributes except the ones requested or required (e.g., joining attr.)

Faloutsos CMU SCS 15-415 16

CMU SCS

Equivalence of expressions

- Joins
  - Commutative , associative

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

  - Q: n-way join - how many diff. orderings?

Faloutsos CMU SCS 15-415 17

CMU SCS

Equivalence of expressions

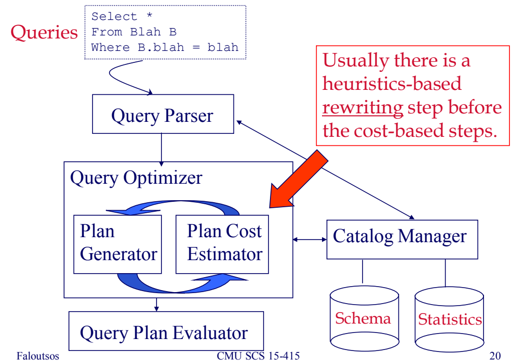
- Joins - Q: n-way join - how many diff. orderings?
- A: Catalan number ~ 4^n
  - Exhaustive enumeration: too slow.

Faloutsos CMU SCS 15-415 18

Q-opt steps

- bring query in internal form (eg., parse tree)
- … into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- **estimate cost**; pick best

Cost-based Query Sub-System

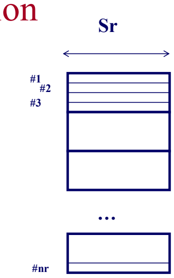


Cost estimation

- Eg., find ssn’s of students with an ‘A’ in 415 (using seq. scanning)
- How long will a query take?
  - CPU (but: small cost; decreasing; tough to estimate)
  - Disk (mainly, # block transfers)
- How many tuples will qualify?
- (what statistics do we need to keep?)

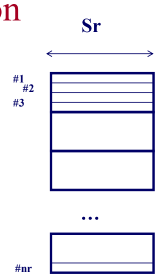
Cost estimation

- Statistics: for each relation ‘r’ we keep
  - nr : # tuples;
  - Sr : size of tuple in bytes



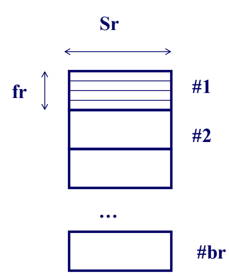
Cost estimation

- Statistics: for each relation ‘r’ we keep
  - …
  - V(A,r): number of distinct values of attr. ‘A’
  - (recently, histograms, too)



Derivable statistics

- blocking factor = max# records/block (=??)
- br: # blocks (=??)
- SC(A,r) = selection cardinality = avg# of records with A=given (=??)



CMU SCS

Derivable statistics

- blocking factor = max# records/block (= B/Sr ; B: block size in bytes)
- br: # blocks (= nr / (blocking-factor))

CMU SCS

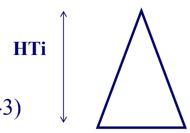
Derivable statistics

- SC(A,r) = selection cardinality = avg# of records with A=given (= nr / V(A,r)) (assumes uniformity...) – eg: 10,000 students, 10 colleges – how many students in SCS?

CMU SCS

Additional quantities we need:

- For index ‘i’:
  - fi: average fanout (~50-100)
  - HTi: # levels of index ‘i’ (~2-3)
    - ~ log(#entries)/log(fi)
  - LBi: # blocks at leaf level



CMU SCS

Statistics

- Where do we store them?
- How often do we update them?

CMU SCS

Q-opt steps

- bring query in internal form (eg., parse tree)
- … into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- **estimate cost**; pick best

CMU SCS

Selections

- we saw simple predicates (A=constant; eg., ‘name=Smith’)
- how about more complex predicates, like
  - ‘salary > 10K’
  - ‘age = 30 **and** job-code=“analyst” ’
- what is their selectivity?

CMU SCS

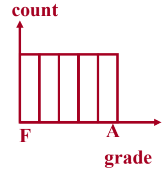
Selections – complex predicates

- selectivity sel(P) of predicate P :
  == fraction of tuples that qualify
  sel(P) = SC(P) / nr

CMU SCS

Selections – complex predicates

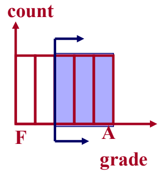
- eg., assume that V(grade, TAKES)=5 distinct values
- simple predicate P: A=constant
  - sel(A=constant) = 1/V(A,r)
  - eg., sel(grade=‘B’) = 1/5
- (what if V(A,r) is unknown??)



CMU SCS

Selections – complex predicates

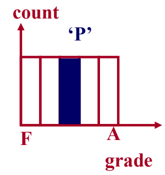
- range query: sel(grade >= ‘C’)
  - sel(A>a) = (Amax – a) / (Amax – Amin)



CMU SCS

Selections - complex predicates

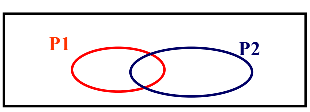
- negation: sel(grade != ‘C’)
  - sel(**not** P) = 1 – sel(P)
  - (Observation: selectivity =~ probability)



CMU SCS

Selections – complex predicates

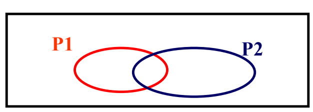
conjunction:
- sel(grade = ‘C’ and course = ‘415’)
- sel(P1 **and** P2) = sel(P1) * sel(P2)
- INDEPENDENCE ASSUMPTION



CMU SCS

Selections – complex predicates

disjunction:
- sel(grade = ‘C’ or course = ‘415’)
- sel(P1 **or** P2) = sel(P1) + sel(P2) – sel(P1 **and** P2)
- = sel(P1) + sel(P2) – sel(P1)*sel(P2)
- INDEPENDENCE ASSUMPTION, again



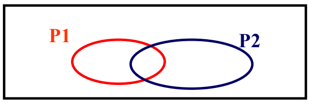
CMU SCS

Selections – complex predicates

disjunction: in general

sel(P1 **or** P2 **or** … Pn) =

1 - (1- sel(P1)) * (1 - sel(P2)) * … (1 - sel(Pn))



CMU SCS

Selections – summary

- sel(A=constant) = 1/V(A,r)
- sel(A>a) = (Amax – a) / (Amax – Amin)
- sel(not P) = 1 – sel(P)
- sel(P1 **and** P2) = sel(P1) * sel(P2)
- sel(P1 **or** P2) = sel(P1) + sel(P2) – sel(P1)*sel(P2)
- sel(P1 **or** ... **or** Pn) = 1 - (1-sel(P1))*...*(1-sel(Pn))
- UNIFORMITY and INDEPENDENCE ASSUMPTIONS

CMU SCS

Result Size Estimation for Joins

- Q: Given a join of R and S, what is the range of possible result sizes (in #of tuples)?
  - Hint: what if R_cols∩S_cols = ∅?
  - R_cols∩S_cols is a key for R (and a Foreign Key in S)?

CMU SCS

Result Size Estimation for Joins

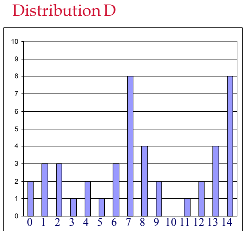
- General case: R_cols∩S_cols = {A} (and A is key for neither)
  - match each R-tuple with S-tuples
    est_size <~ NTuples(R) * NTuples(S)/NKeys(A,**S**)
    <~ nr * ns / V(A,S)
  - symmetrically, for S:
    est_size <~ NTuples(R) * NTuples(S)/NKeys(A,**R**)
    <~ nr * ns / V(A,R)
  - Overall:
    est_size = NTuples(R)*NTuples(S)/MAX{NKeys(A,**S**), NKeys(A,**R**)}

CMU SCS

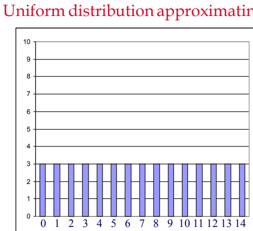
On the Uniform Distribution Assumption

- Assuming uniform distribution is rather crude

Distribution D



Uniform distribution approximating D

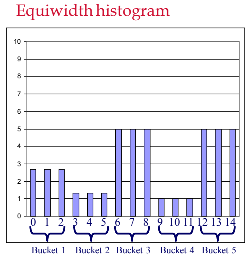


CMU SCS

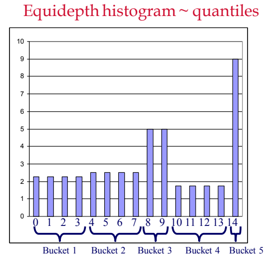
Histograms

- For better estimation, use a *histogram*

Equiwidth histogram



Equidepth histogram ~ quantiles



CMU SCS

Q-opt steps

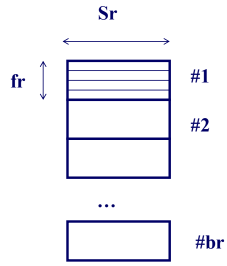
- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- **generate alt. plans**
  - **single relation**
  - multiple relations
- estimate cost; pick best

Faloutsos CMU SCS 15-415 43

CMU SCS

plan generation

- Selections – eg.,
  select *
  from TAKES
  where grade = 'A'
- Plans?

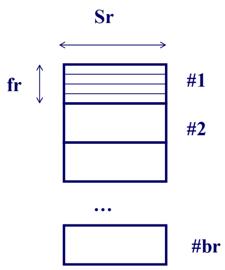


Faloutsos CMU SCS 15-415 44

CMU SCS

plan generation

- Plans?
  - seq. scan
  - binary search
    - (if sorted & consecutive)
  - index search
    - if an index exists



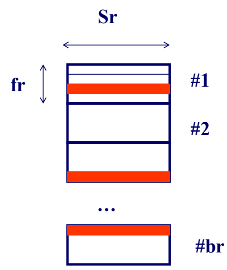
Faloutsos CMU SCS 15-415 45

CMU SCS

plan generation

seq. scan – cost?

- br (worst case)
- br/2 (average, if we search for primary key)



Faloutsos CMU SCS 15-415 46

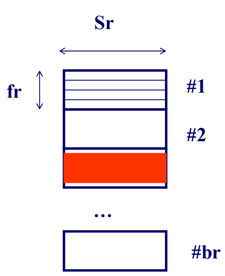
CMU SCS

plan generation

binary search – cost?

if sorted and consecutive:

- ~log(br) +
- SC(A,r)/fr (=blocks spanned by qual. tuples)



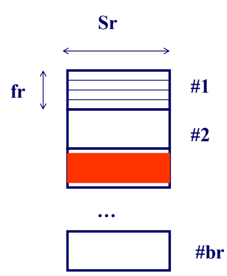
Faloutsos CMU SCS 15-415 47

CMU SCS

plan generation

estimation of selection cardinalities SC(A,r):

non-trivial – we saw it earlier



Faloutsos CMU SCS 15-415 48

CMU SCS

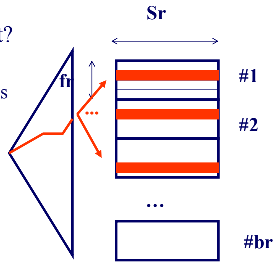
plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

case#1: primary key

case#2: sec. key – clustering index

case#3: sec. key – non-clust. index



Faloutsos CMU SCS 15-415 49

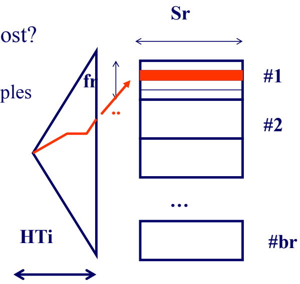
CMU SCS

plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

case#1: primary key – cost:

HTi + 1



Faloutsos CMU SCS 15-415 50

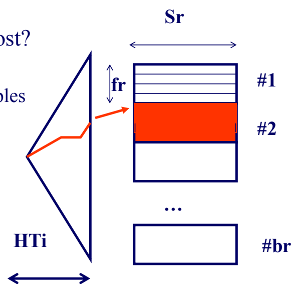
CMU SCS

plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

case#2: sec. key – clustering index

HTi + SC(A,r)/fr



Faloutsos CMU SCS 15-415 51

CMU SCS

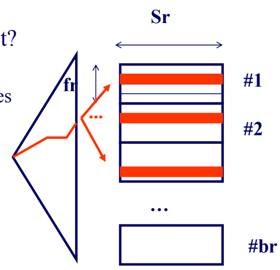
plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

case#3: sec. key – non-clust. index

HTi + SC(A,r)

(actually, pessimistic...)



Faloutsos CMU SCS 15-415 52

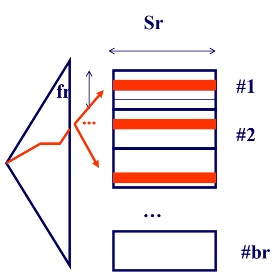
CMU SCS

plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

(actually, pessimistic...

better estimates: Cardenas' formula)



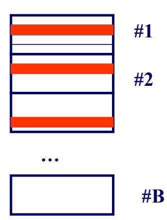
Faloutsos CMU SCS 15-415 53

CMU SCS

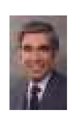
Cardena's formula

- q: # qual records
- Q: # qual. blocks
- N: # records total
- B: # blocks total

- Q=??



Alfonso Cardenas (UCLA)

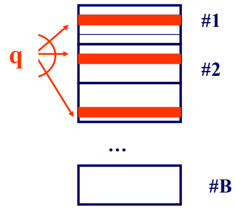


54

CMU SCS

Cardena's formula

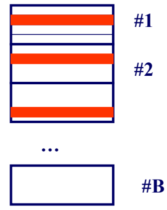
- Pessimistic:
  - Q = q



CMU SCS

Cardena's formula

- Pessimistic:
  - Q = q
- More realistic
  - Q = q if q<=B
  - Q = B otherwise

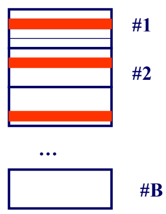


CMU SCS

Cardena's formula

- Cardenas' formula

$$Q = B\,[1-(1-1/B)^q]$$



CMU SCS

Plans for single relation - summary

- no index: scan (dup-elim; sort)
- with index:
  - single index access path
  - multiple index access path
  - sorted index access path
  - index-only access path

CMU SCS

Citation

- P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. *Access path selection in a relational database management system*. In SIGMOD Conference, pages 23--34, 1979.

CMU SCS

Frequently cited database publications

http://www.informatik.uni-trier.de/~ley/db/about/top.html

#	Publication
608	Peter P. Chen: The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. Database Syst. 1(1): 9-36(1976)
580	E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Commun. ACM 13(6): 377-387(1970)
371	Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, Thomas G. Price: Access Path Selection in a Relational Database Management System. SIGMOD Conference 1979: 23-34
366	Jeffrey D. Ullman: Principles of Database and Knowledge Base Systems, Volume I. Computer Science Press 1988, ISBN 0-7167-8158-1
…	…

CMU SCS

Statistics for Optimization

- NCARD (T) - cardinality of relation T in tuples
- TCARD (T) - number of pages containing tuples from T
- P(T) = TCARD(T)/(# of non-empty pages in the segment)
  - If segments only held tuples from one relation there would be no need for P(T)
- ICARD(I) - number of distinct keys in index I
- NINDX(I) - number of pages in index I

Faloutsos CMU SCS 15-415 61

CMU SCS

Predicate Selectivity Estimation

attr = value	**F = 1/ICARD(attr index)** – if index exists **F = 1/10** otherwise
attr1 = attr2	**F = 1/max(ICARD(I1),ICARD(I2))** or **F = 1/ICARD(Ii)** – if only index i exists, or **F = 1/10**
val1 < attr < val2	**F = (value2-value1)/(high key-low key)** **F = 1/4** otherwise
expr1 or expr2	**F = F(expr1)+F(expr2)–F(expr1)*F(expr2)**
expr1 and expr2	**F = F(expr1) * F(expr2)**
NOT expr	**F = 1 – F(expr)**

Faloutsos CMU SCS 15-415 62

CMU SCS

Costs per Access Path Case

Unique index matching equal predicate	**1+1+W**
Clustered index I matching >=1 preds	**F(preds)*(NINDX(I)+TCARD)+W*RSICARD**
Non-clustered index I matching >=1 preds	**F(preds)*(NINDX(I)+NCARD)+W*RSICARD**
Segment scan	**TCARD/P + W*RSICARD**

Faloutsos CMU SCS 15-415 63

CMU SCS

Q-opt steps

- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
  - single relation
  - **multiple relations**
    - Main idea
    - Dynamic programming – reminder
    - Example
- estimate cost; pick best

Faloutsos CMU SCS 15-415 64

CMU SCS

n-way joins

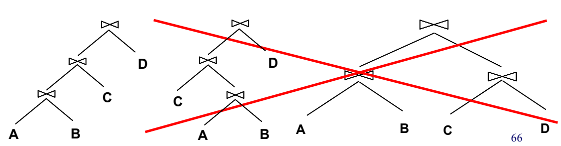
- r1 JOIN r2 JOIN ... JOIN rn
- typically, break problem into 2-way joins
  - choose between NL, sort merge, hash join, ...

Faloutsos CMU SCS 15-415 65

CMU SCS

Queries Over Multiple Relations

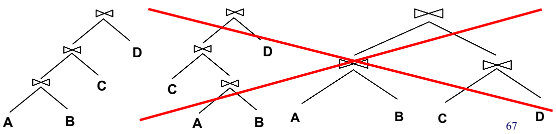
- As number of joins increases, number of alternative plans grows rapidly → *need to restrict search space*
- Fundamental decision in System R: *only left-deep join trees* are considered. Advantages?



66

Queries Over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly → *need to restrict search space*
- Fundamental decision in System R: *only left-deep join trees* are considered. Advantages?
  - *fully pipelined* plans.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).



Queries over Multiple Relations

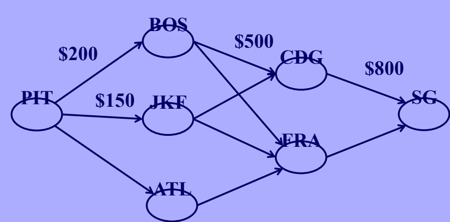
- Enumerate the orderings (= left deep tree)
- enumerate the plans for each operator
- enumerate the access paths for each table

Dynamic programming, to save cost estimations

Q-opt steps

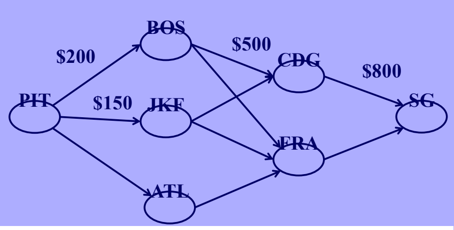
- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
    - Main idea
    - **Dynamic programming – reminder**
    - Example
- estimate cost; pick best

(Reminder: Dynamic Programming)



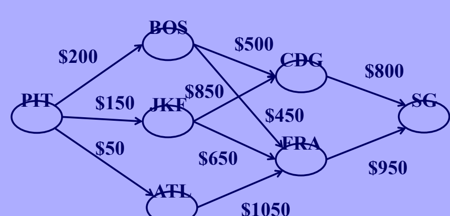
Cheapest flight PIT -> SG ?

(Reminder: Dynamic Programming)



Assumption: NO package deals: cost CDG->SG is always $800, no matter how reached CDG

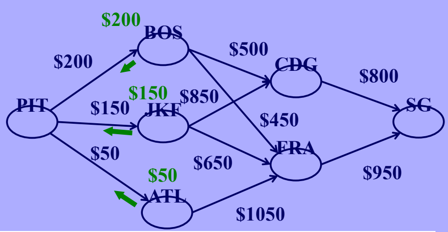
(Reminder: Dynamic Programming)



Solution: compute partial optimal, left-to-right:

CMU SCS

(Reminder: Dynamic Programming)

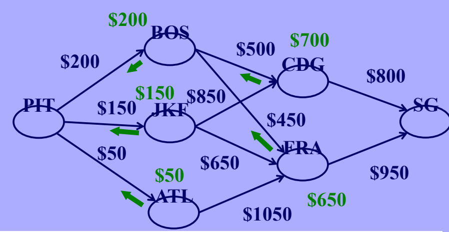


Solution: compute partial optimal, left-to-right:

Faloutsos CMU SCS 15-415 73

CMU SCS

(Reminder: Dynamic Programming)

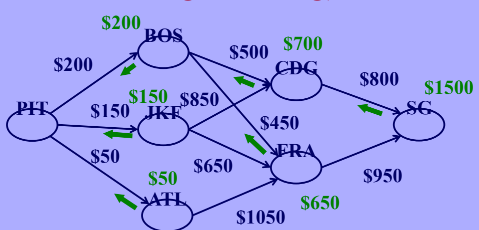


Solution: compute partial optimal, left-to-right:

Faloutsos CMU SCS 15-415 74

CMU SCS

(Reminder: Dynamic Programming)

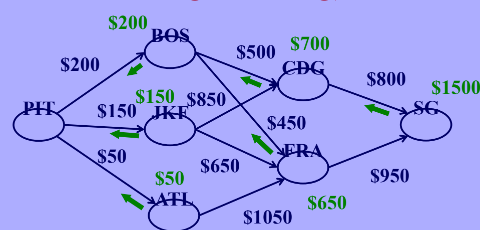


Solution: compute partial optimal, left-to-right:

Faloutsos CMU SCS 15-415 75

CMU SCS

(Reminder: Dynamic Programming)

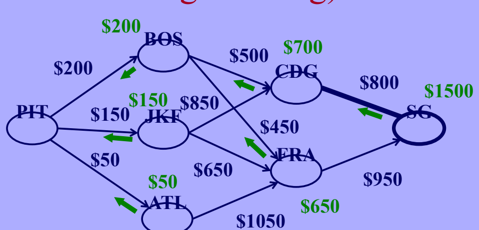


So, best price is $1,500 – which legs?

Faloutsos CMU SCS 15-415 76

CMU SCS

(Reminder: Dynamic Programming)



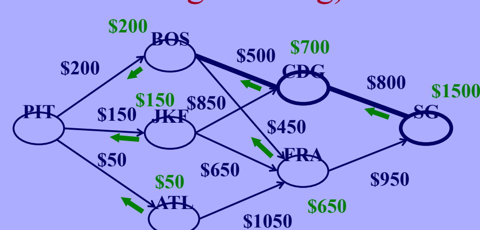
So, best price is $1,500 – which legs?

A: follow the winning edges, backwards

77

CMU SCS

(Reminder: Dynamic Programming)

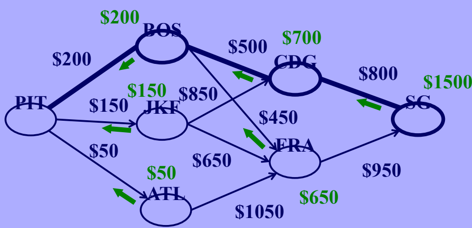


So, best price is $1,500 – which legs?

A: follow the winning edges, backwards

78

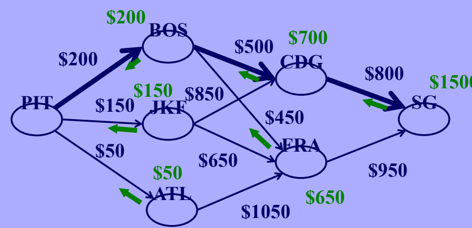
(Reminder: Dynamic Programming)



So, best price is $1,500 – which legs?

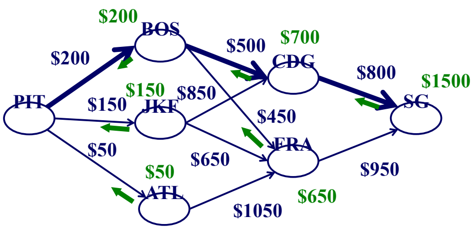
A: follow the winning edges, backwards

(Reminder: Dynamic Programming)



Q: what are the states, costs and arrows, in q-opt?

(Reminder: Dynamic Programming)

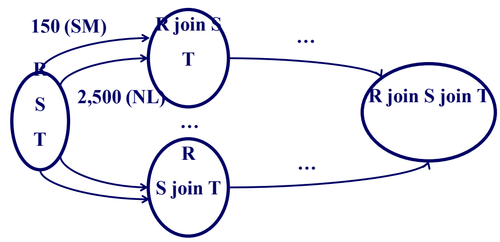


Q: what are the states (and costs and arrows), in q-opt?

A: set of intermediate result tables

Q-opt and Dyn. Programming

- E.g., compute R join S join T



Q-opt and Dyn. Programming

- Details: how to record the fact that, say R is sorted on R.a? or that the user requires sorted output?
- A:
- E.g., consider the query

  select *
  from R, S, T
  where R.a = S.a **and** S.b = T.b
  order by R.a

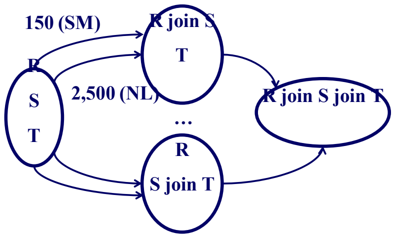
Q-opt and Dyn. Programming

- Details: how to record the fact that, say R is sorted on R.a? or that the user requires sorted output?
- A: record orderings, in the state
- E.g., consider the query

  select *
  from R, S, T
  where R.a = S.a **and** S.b = T.b
  order by R.a

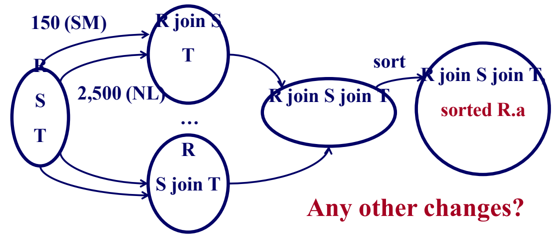
Q-opt and Dyn. Programming

- E.g., compute R join S join T order by R.a



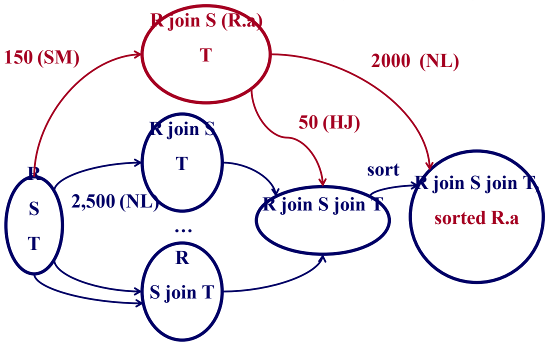
Q-opt and Dyn. Programming

- E.g., compute R join S join T order by R.a



Any other changes?

Q-opt and Dyn. Programming



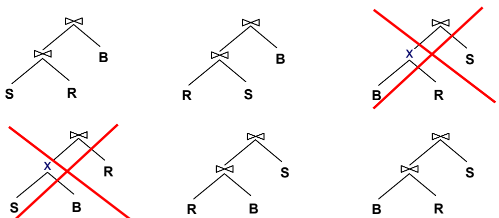
Q-opt steps

- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
    - Main idea
    - Dynamic programming – reminder
    - **Example**
- estimate cost; pick best

Candidate Plans

```
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
```

1. Enumerate relation orderings:

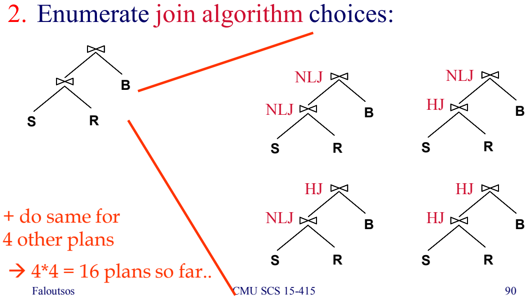


Prune plans with cross-products immediately!

Candidate Plans

```
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
```

2. Enumerate join algorithm choices:



+ do same for 4 other plans

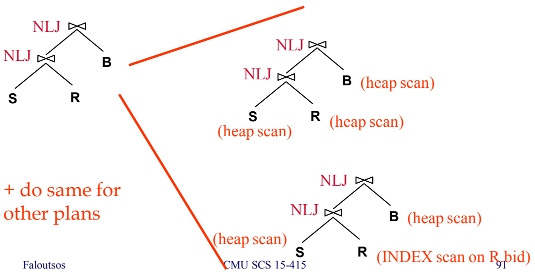
→ 4*4 = 16 plans so far..

CMU SCS

Candidate Plans

SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid

3. Enumerate access method choices:



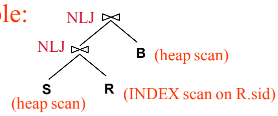
+ do same for other plans

Faloutsos CMU SCS 15-415 91

CMU SCS

Now estimate the cost of each plan

Example:



Faloutsos CMU SCS 15-415 92

CMU SCS

Q-opt steps

- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
  - **nested subqueries**
- estimate cost; pick best

Faloutsos CMU SCS 15-415 93

CMU SCS

Q-opt steps

- Everything so far: about a single query block

Faloutsos CMU SCS 15-415 94

CMU SCS

Query Rewriting

- Re-write nested queries
- to: de-correlate and/or flatten them

Faloutsos CMU SCS 15-415 95

CMU SCS

Example: Decorrelating a Query

SELECT S.sid
FROM Sailors S
WHERE EXISTS
  *(SELECT **
  FROM Reserves R
  WHERE R.bid=103
  AND R.sid=S.sid)

Equivalent uncorrelated query:
SELECT S.sid
FROM Sailors S
WHERE S.sid IN
  (SELECT R.sid
  FROM Reserves R
  WHERE R.bid=103)

- Advantage: nested block only needs to be executed once (rather than once per S tuple)

Faloutsos CMU SCS 15-415 96

Example: "Flattening" a Query

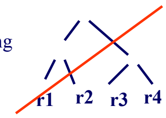
SELECT S.sid
FROM Sailors S
WHERE S.sid IN
(SELECT R.sid
FROM Reserves R
WHERE R.bid=103)

Equivalent non-nested query:
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
AND R.bid=103

- Advantage: can use a join algorithm + optimizer can select among join algorithms & reorder freely

Structure of query optimizers:

System R:
- break query in query blocks
- simple queries (ie., no joins): look at stats
- n-way joins: left-deep join trees; ie., only one intermediate result at a time
  - pros: smaller search space; pipelining
  - cons: may miss optimal
- 2-way joins: NL and sort-merge



Structure of query optimizers:

More heuristics by Oracle, Sybase and Starburst (-> DB2)

In general: q-opt is very important for large databases.

('**explain select** <sql-statement>' gives plan)

Q-opt steps

- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best

Conclusions

- Ideas to remember:
  - syntactic q-opt – do selections early
  - selectivity estimations (uniformity, indep.; histograms; join selectivity)
  - hash join (nested loops; sort-merge)
  - left-deep joins
  - dynamic programming