DPccp
standard DP algorithms consider many pairs that are discarded due to disconnectedness or failing disjointness tests

- taking the query graph into account avoids this [2]
Nodes in the query graph are ordered according to a BFS
Start with the last node, all the nodes with smaller ID are forbidden
At every step: compute neighborhood, get forbidden nodes, enumerate subsets of non-forbidden nodes $N$
Recursive calls for subsets of $N$
EnumerateCsg + EnumerateCmp produce all ccp
resulting algorithm DPccp considers exactly \#ccp pairs
which is the lower bound for all DP enumeration algorithms
Hypergraphs
- Complex Join Predicates
- Reordering constraints due to non-inner Joins
  \( \Rightarrow \) DPHyp [3]
- Edges connect sets of nodes
- Similar to DPccp but hyperedges 'lead' to the smallest edge within the set
- Subproblems may be disconnected
Query Simplification
Simplify the query graph if it is too complex [4].

- GOO: greedily choose joins to perform
- Simplification: greedily choose joins that must be avoided (we can start with ‘obvious’ decisions)
- \[ \text{benefit}(X \bowtie R_1, X \bowtie R_2) = \frac{C((X \bowtie R_1) \bowtie R_2)}{C((X \bowtie R_2) \bowtie R_1)} \]
Homework
Homework

A \leftarrow A.x=B.y

B \leftarrow B.x=C.y

C \leftarrow \begin{align*} C.y &= D.x \\ E \leftarrow E.x &= F.y \end{align*}

D \leftarrow

E \leftarrow

F \leftarrow

- Syntactic eligibility set - relations that have to be in the input
- Total eligibility set - captures also reordering restrictions, construct bottom-up
- Conflicts: \( C.x=E.y \) and \( C.y=D.x, C.x=E.y \) and \( B.x=C.y, A.x=B.y \) and \( C.x=E.y \)
Important: consider all possible edge combinations, that is, 
\( \text{benefit}(R_0 \Join R_1, R_0 \Join R_2) \) together with \( \text{benefit}(R_0 \Join R_2, R_0 \Join R_1) \)
\[ \text{benefit}(R_0 \Join R_1, R_0 \Join R_3) = \frac{202}{300} \]
\[ b(R_0 \Join R_3, R_0 \Join R_1) = \frac{300}{202} \]
\[ b(R_1 \Join R_2, R_1 \Join R_0) = \frac{20}{12} \]
\[ b(R_3 \Join R_0, R_3 \Join R_2) = \frac{2}{1} \]
\[ b(R_2 \Join R_3, R_2 \Join R_1) = \frac{5}{4} \]
\[ b(R_1 \Join R_4, R_1 \Join R_0) = \frac{500}{251} \]
\[ b(R_1 \Join R_4, R_1 \Join R_3) = \frac{300}{251} \]
\[ R_3 \Join R_2 \text{ before } R_3 \Join R_0. \]
Replace \( R_0 - R_3 \) by \{R_0\} - \{R_2, R_3\}
benefit\( (R_0 \Join R_1, R_0 \Join R_3) = \frac{202}{300} \)

\( b(R_0 \Join R_3, R_0 \Join R_1) = \frac{300}{202} \)

\( b(R_1 \Join R_2, R_1 \Join R_0) = \frac{20}{12} \)

\( b(R_2 \Join R_3, R_2 \Join R_1) = \frac{5}{4} \)

\( b(R_1 \Join R_4, R_1 \Join R_0) = \frac{500}{251} \)

\( b(R_1 \Join R_4, R_1 \Join R_3) = \frac{300}{251} \)

\( b(R_0 \Join (R_3 \Join R_2), R_0 \Join R_1) = \frac{C((R_0 \Join (R_3 \Join R_2)) \Join R_1)}{C((R_0 \Join R_1) \Join (R_3 \Join R_2))} = \frac{850}{370} \)

\( b((R_2 \Join R_3) \Join R_0, R_2 \Join R_1) = \frac{C(((R_2 \Join R_3) \Join R_0) \Join R_1)}{C((R_2 \Join R_3) \Join R_1) \Join R_0} = 1 \)

\( R_0 \Join R_1 \) before \( R_0 \Join (R_3 \Join R_2) \).

Replace \( \{ R_0 \} - \{ R_2, R_3 \} \) by \( \{ R_0, R_1 \} - \{ R_2, R_3 \} \)
Next Homework

▶ implement DP (either DPsize or DPsub)
▶ Slides: db.in.tum.de/teaching/ws1819/queryopt
▶ Exercise task: gitlab
▶ Questions, Comments, etc:
  mattermost @ mattermost.db.in.tum.de/qo18
▶ Exercise due: 9 AM next monday
On the correct and complete enumeration of the core search space. 

Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. 

Dynamic programming strikes back. 
Query simplification: graceful degradation for join-order optimization. 