# Query Optimization

## Exercise Session 10

Andrey Gubichev

June 30, 2014

# Random join trees with cross products

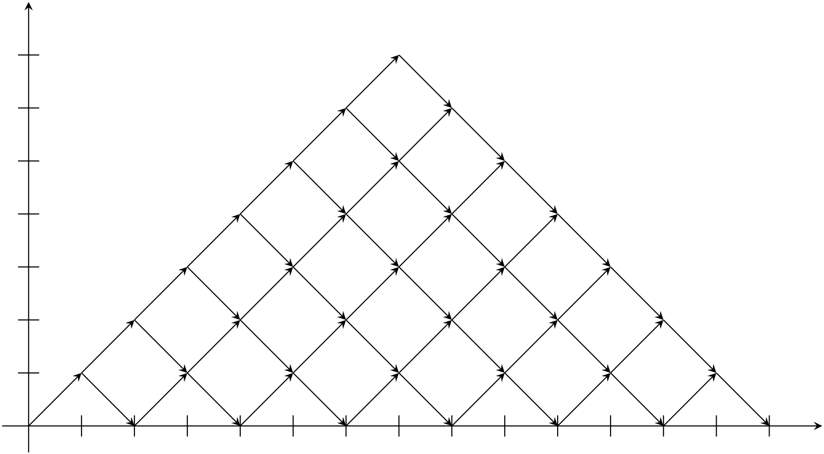
- Generate a tree, then generate a permutation: $C(n-1)$ trees, $n!$ permutations
- Pick a random number $b \in [0, C(n-1)[$, *unrank* $b$
- Pick a random number $p \in [0, n![$, *unrank* $p$
- Attach the permutation to the leaves

# Unranking

- every tree is a word in $\{(,)\}$
- map such words to the grid, every step up is (, down )

# Unranking

- every tree is a word in $\{(,)\}$
- map such words to the grid, every step up is (, down )



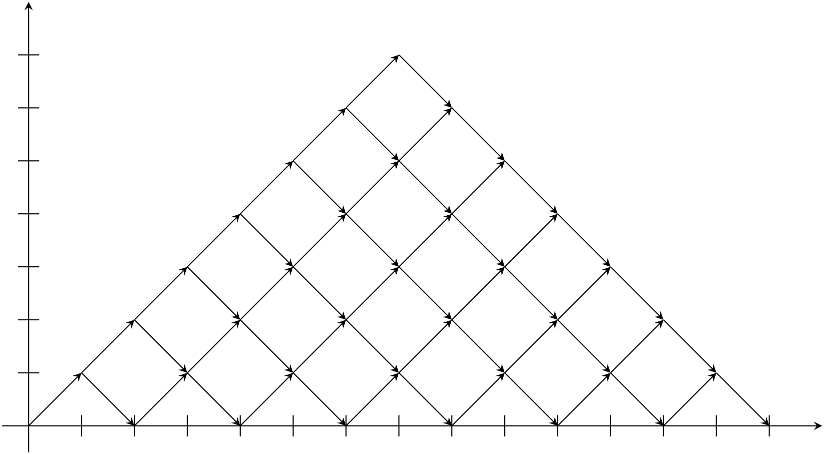
# Unranking

- every tree is a word in $\{(,)\}$
- map such words to the grid, every step up is (, down )
- the number of different paths $q$ can be computed (see lectures)
- Procedure: start in (0,0), walk up as long as rank is smaller than $q$. When it is bigger, step down, *rank*=*rank*-$q$

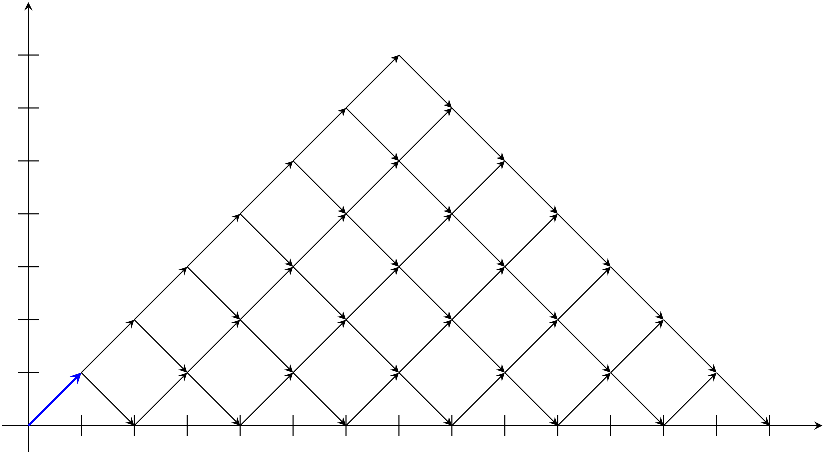
# Example

- Bushy tree number 56, 8 leaves

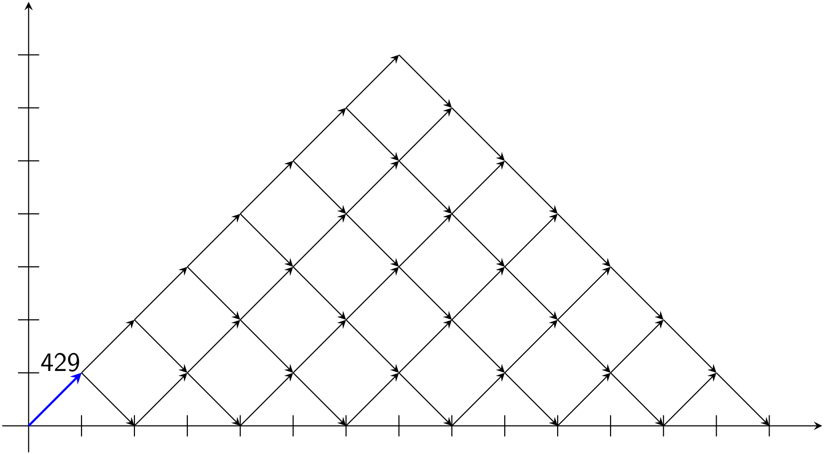
# Random Join Tree Selection



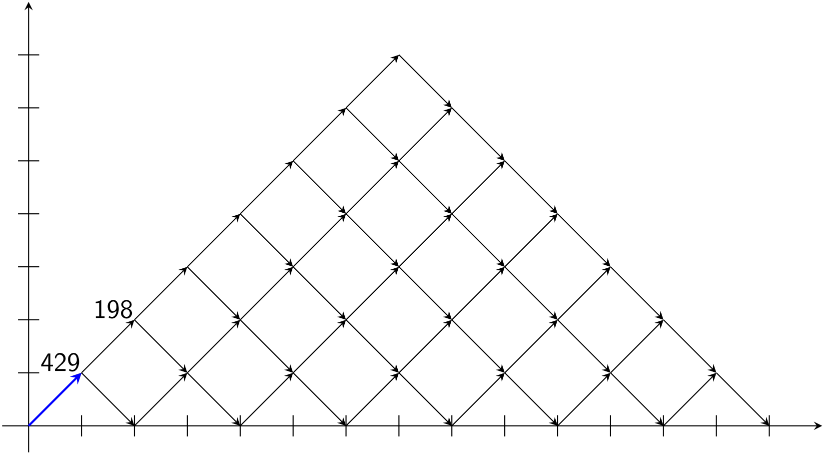
# Random Join Tree Selection



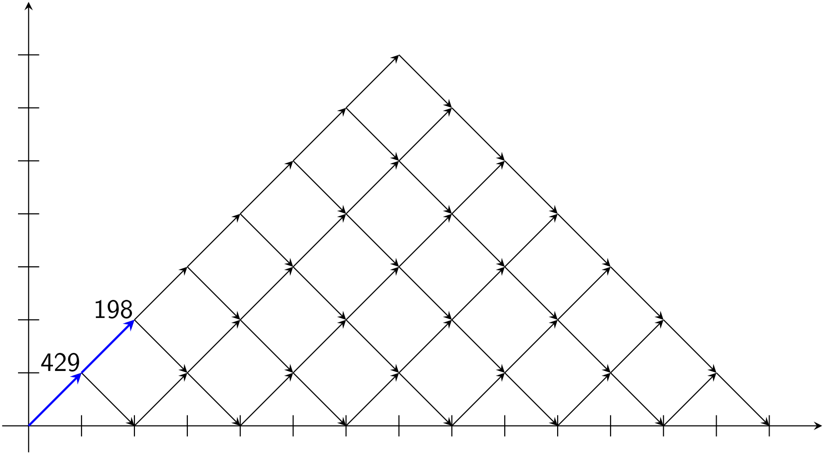
# Random Join Tree Selection



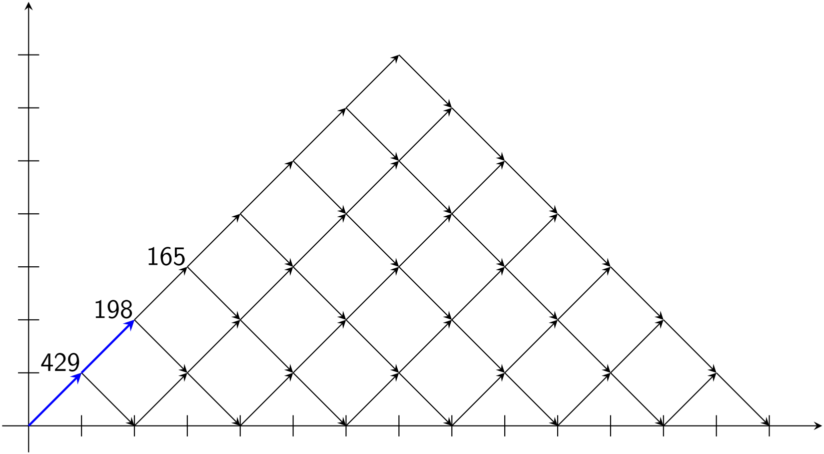
# Random Join Tree Selection



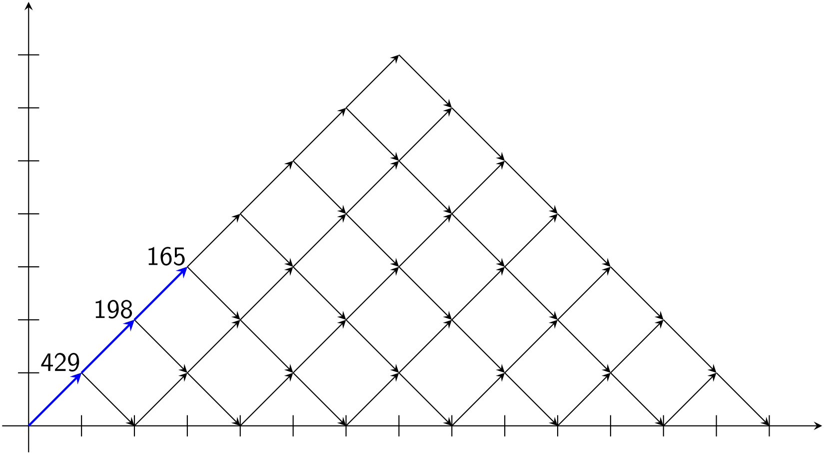
# Random Join Tree Selection



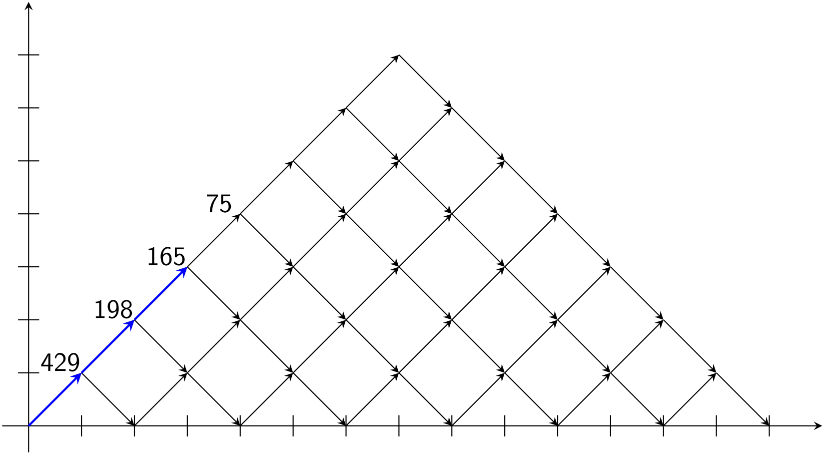
# Random Join Tree Selection



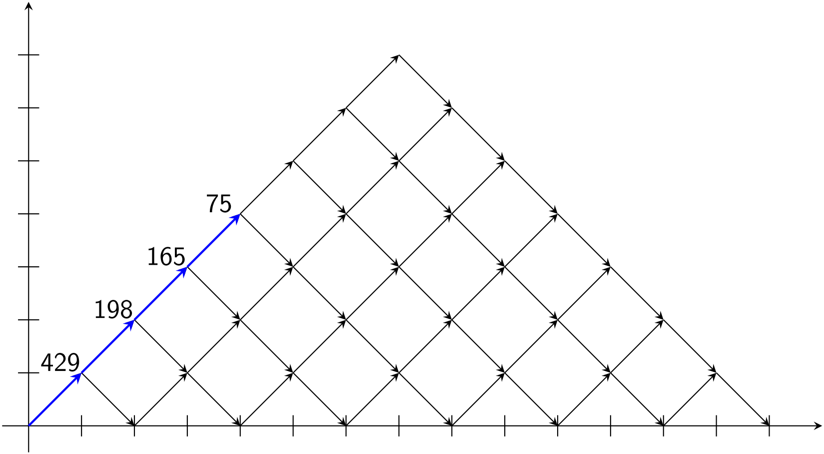
# Random Join Tree Selection



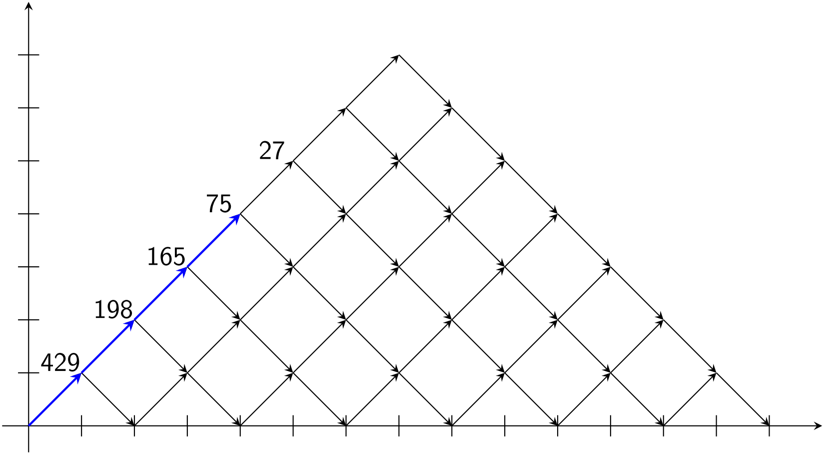
# Random Join Tree Selection



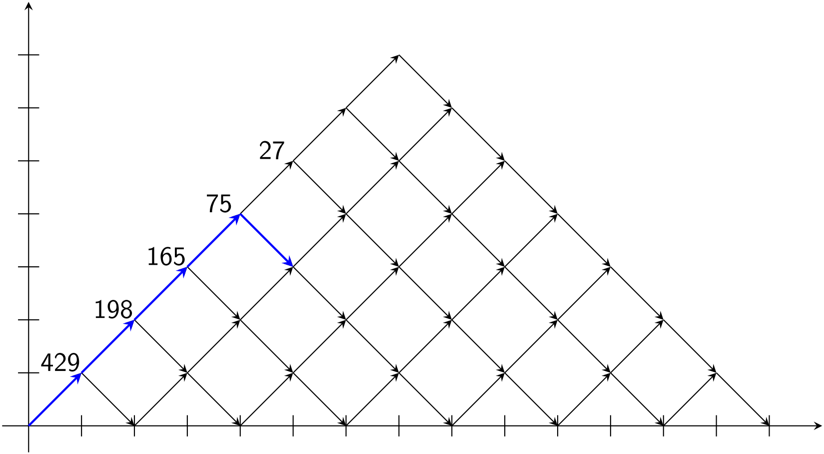
# Random Join Tree Selection



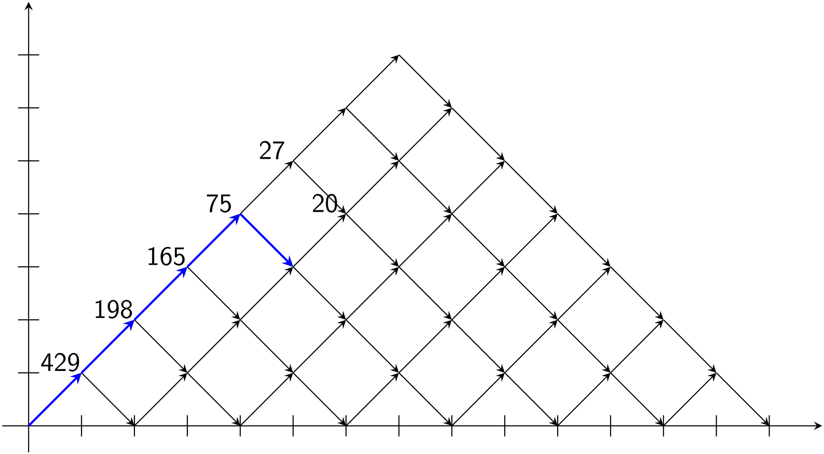
# Random Join Tree Selection



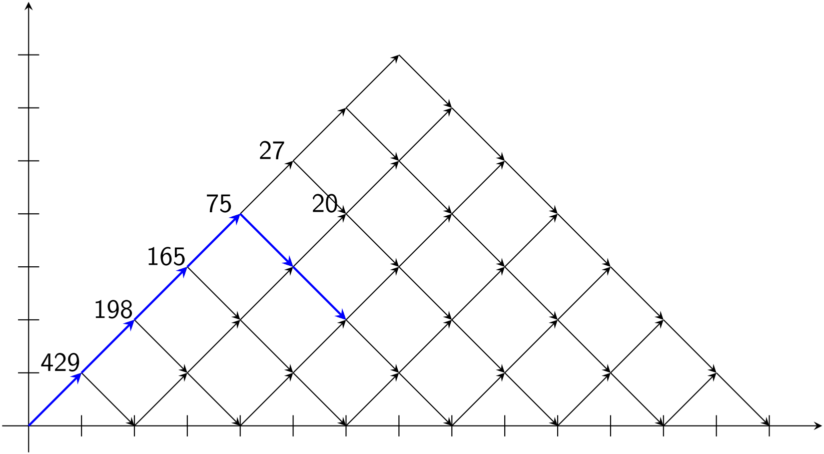
# Random Join Tree Selection



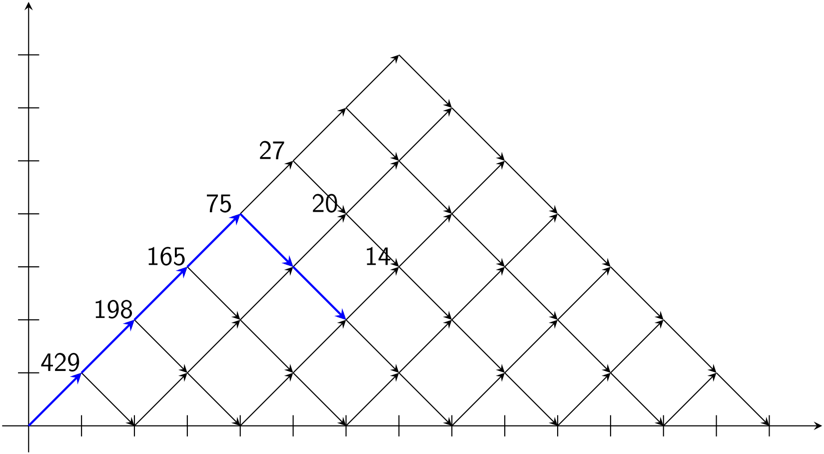
# Random Join Tree Selection



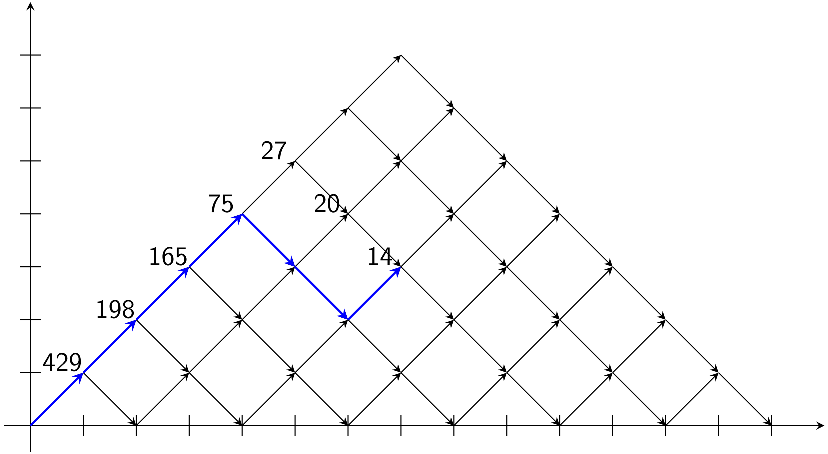
# Random Join Tree Selection



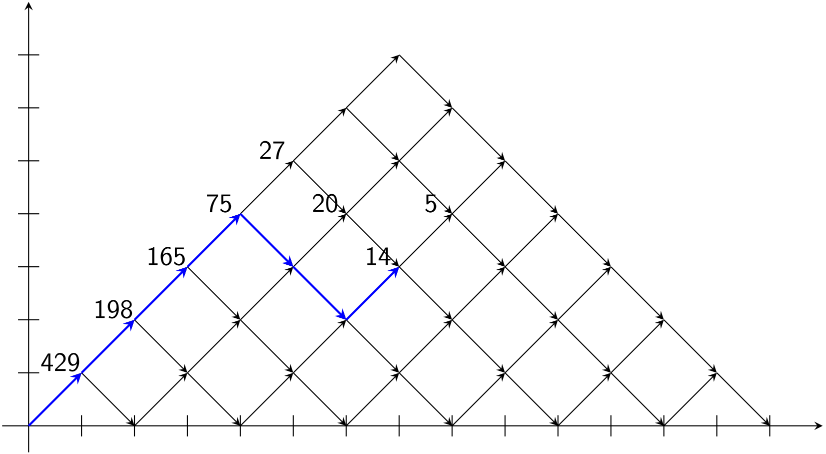
# Random Join Tree Selection



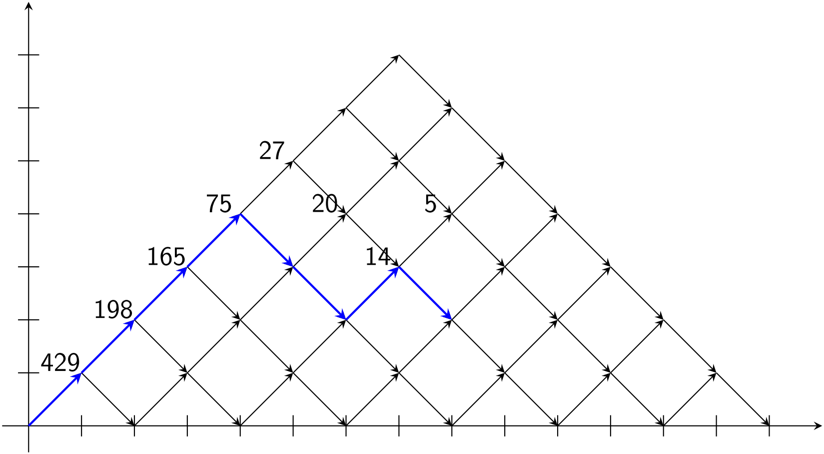
# Random Join Tree Selection



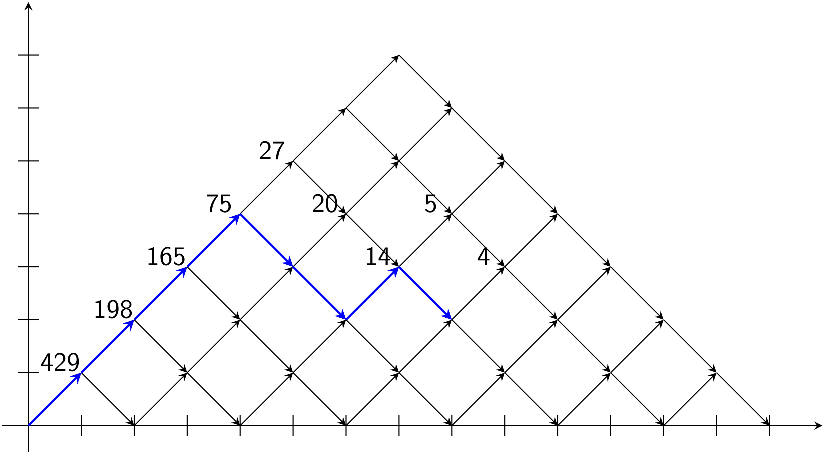
# Random Join Tree Selection



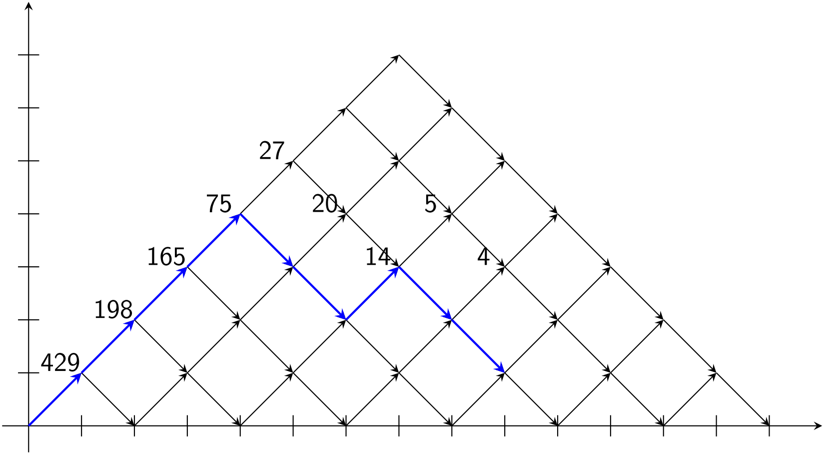
# Random Join Tree Selection



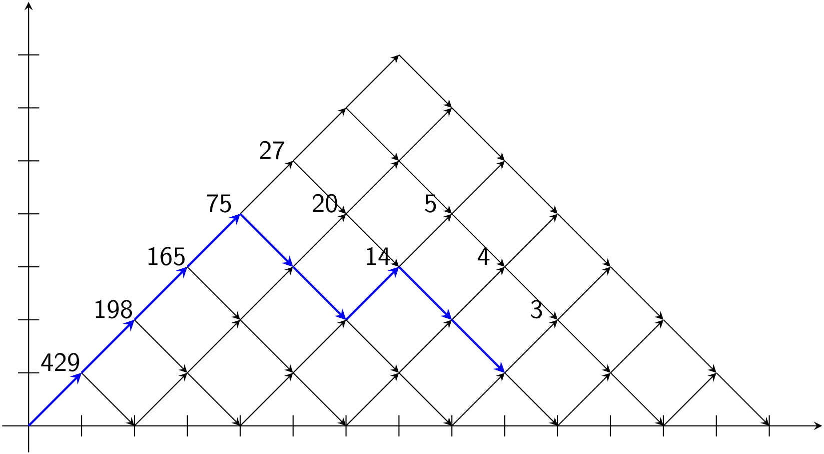
# Random Join Tree Selection



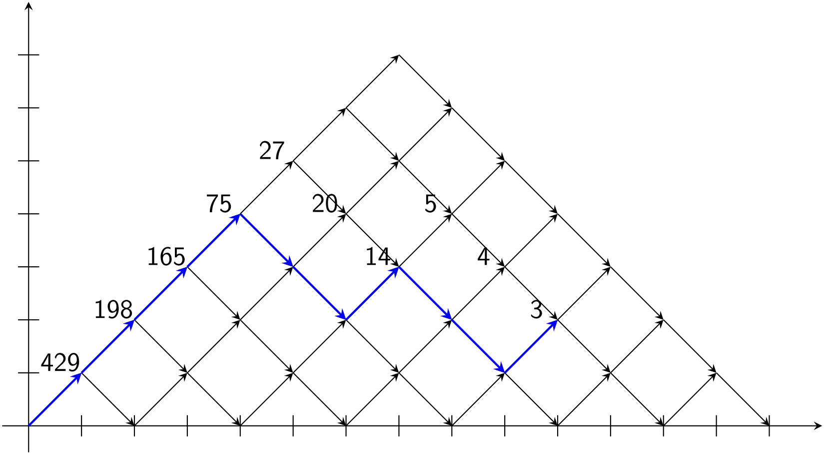
# Random Join Tree Selection



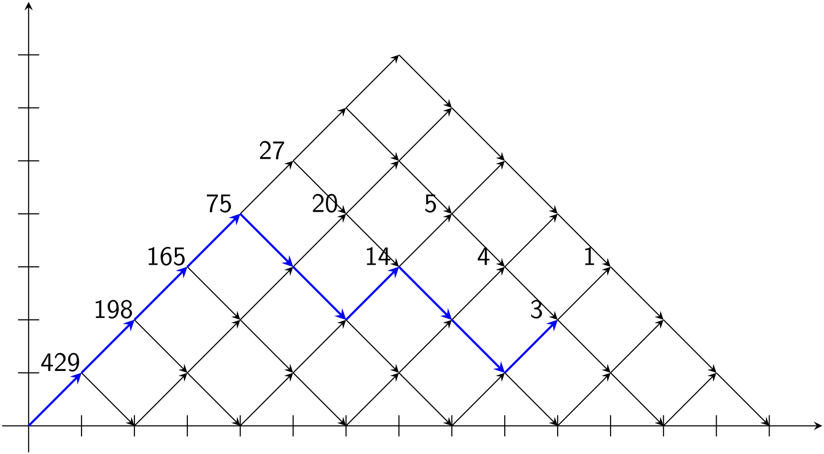
# Random Join Tree Selection



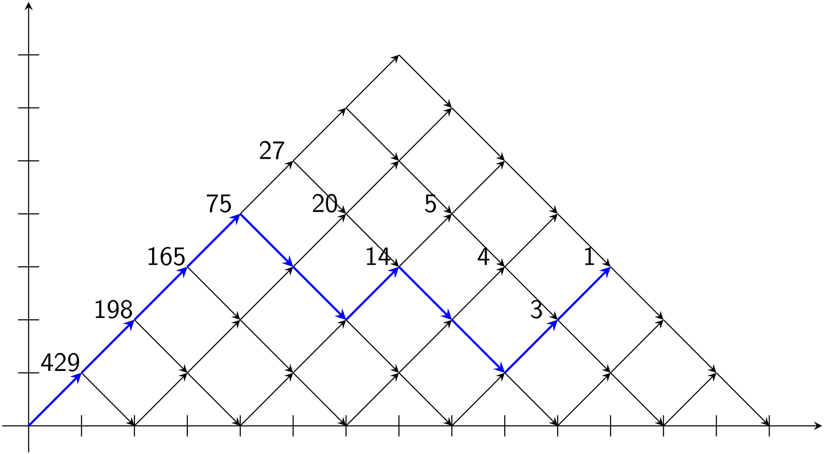
# Random Join Tree Selection



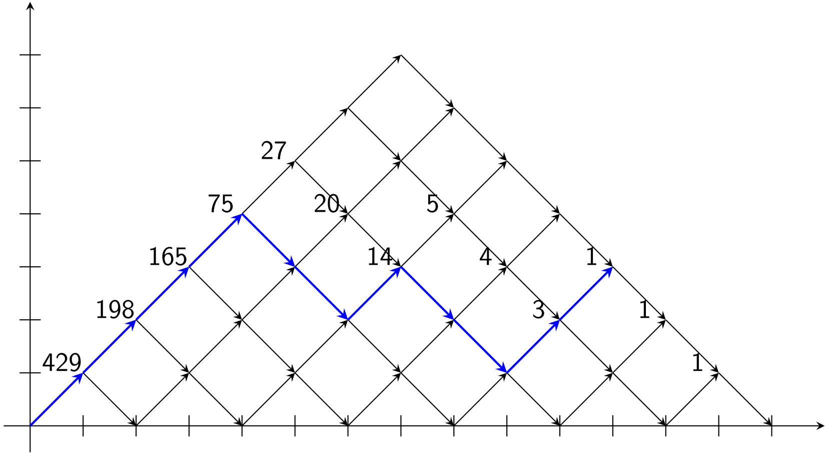
# Random Join Tree Selection



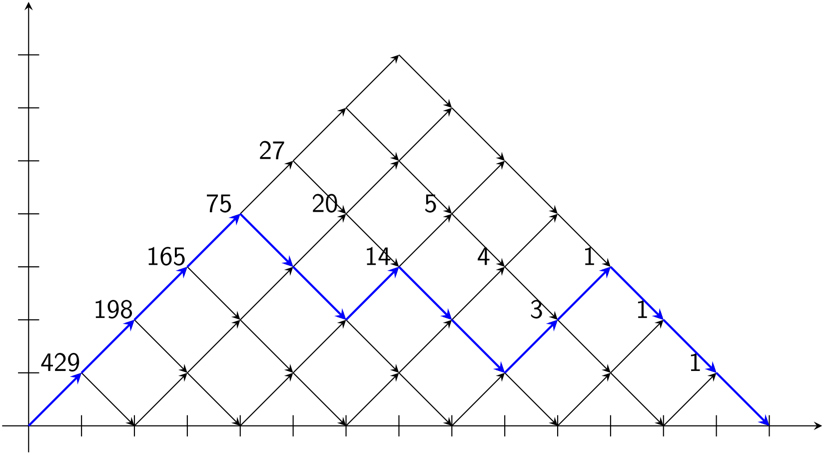
# Random Join Tree Selection



# Random Join Tree Selection



# Random Join Tree Selection



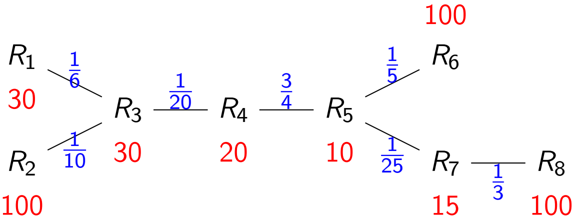
# Plan for today

- Two heuristics: Iterative DP, Quick Pick
- Meta-heuristics

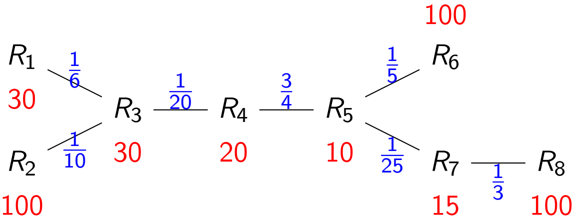
# Iterative DP

- Create all join trees with size up to $k$, get the cheapest one
- Replace the cheapest tree with the compound relation, start all over again

# Iterative Dynamic Programming



# Iterative Dynamic Programming




| $R_1 R_3$ | $R_2 R_3$ | $R_3 R_4$ | $R_4 R_5$ | $R_5 R_6$ | $R_5 R_7$ | $R_7 R_8$ |
|---|---|---|---|---|---|---|
| 150 | 300 | 30 | 150 | 200 | 6 | 500 |

# Quick Pick

- $Trees = \{R_1, \ldots, R_n\}$, $Edges$ = list of edges
- pick a random edge $e \in Edges$ that connects two trees in $Trees$
- exclude two selected trees from $Trees$, add the new tree to $Trees$, $Edges = Edges \setminus \{e\}$
- repeat until the complete join tree is constructed

Question for the homework: How to check that an edge connects two trees? what data structures to use?

# Metaheuristics

# II & SA

Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You'll do a homework exercise on this.

- Rules for left-deep trees: *swap* and *3cycle*
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an "equilibrium":

# II & SA

Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You'll do a homework exercise on this.

- Rules for left-deep trees: *swap* and *3cycle*
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an "equilibrium":
  - # iterations = # relations

# II & SA

Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

$\Rightarrow$ You'll do a homework exercise on this.

- Rules for left-deep trees: *swap* and *3cycle*
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an "equilibrium":
  - $\#$ iterations $= \#$ relations
  - $\#$ iterations $= 16 \times \#$ relations

# II & SA

Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You'll do a homework exercise on this.

- Rules for left-deep trees: *swap* and *3cycle*
- Rules for bushy trees: commutativity, associativity, left/right join exchange

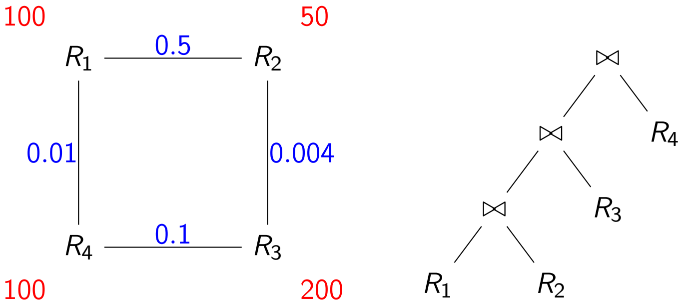
Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an "equilibrium":
  - # iterations = # relations
  - # iterations = 16 × # relations
  - "Would you bet your business on these numbers?"

# Possible transformations

- *Swap* $A \bowtie B \to B \bowtie A$
- *3Cycle* $A \bowtie (B \bowtie C) \to C \bowtie (A \bowtie B)$ (if possible)
- *Associativity* $(A \bowtie B) \bowtie C \to A \bowtie (B \bowtie C)$
- *Left Join exchange* $(A \bowtie B) \bowtie C \to (A \bowtie C) \bowtie B$
- *Right Join exchange* $A \bowtie (B \bowtie C) \to B \bowtie (A \bowtie C)$

# Iterative Improvement




- left deep trees only
  (commutativity for base relations, 3Cycle)
- cost function: $C_{out}$

# Tabu Search

- In each step, take cheapest neighbor[1] (even if more expensive than current)
- Avoid cycles by keeping visited trees in a tabu-set

[1] i.e. join tree that can be produced with a single transformation

# Genetic Algorithms

Big picture

- Create a "population", i.e. create $p$ random join trees
- Encode them using ordered list or ordinal number encoding
- Create the next generation
  - Randomly mutate some members (e.g. exchange two relations)
  - Pairs members of the population and create "crossovers"
- Select the best, kill the rest

Details

- Encodings
- Crossovers

# Encoding

Ordered lists

- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers

- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
- Left-deep trees: $(((R_1 \bowtie R_4) \bowtie R_3) \bowtie R_2) \bowtie R_5$
- Bushy trees: $(R_3 \bowtie (R_1 \bowtie R_2)) \bowtie (R_4 \bowtie R_5)$

# Encoding

Ordered lists

- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers

- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
- Left-deep trees: $(((R_1 \bowtie R_4) \bowtie R_3) \bowtie R_2) \bowtie R_5 \mapsto 13211$
- Bushy trees: $(R_3 \bowtie (R_1 \bowtie R_2)) \bowtie (R_4 \bowtie R_5)$

# Encoding

Ordered lists

- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers

- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
- Left-deep trees: $(((R_1 \bowtie R_4) \bowtie R_3) \bowtie R_2) \bowtie R_5 \ \mapsto\ 13211$
- Bushy trees: $(R_3 \bowtie (R_1 \bowtie R_2)) \bowtie (R_4 \bowtie R_5) \ \mapsto\ 12\,21\,23\,12$

# Crossover

Subsequence exchange for ordered list encoding

- Select subsequence in parent 1, e.g. $abc\underline{def}gh$
- Reorder subsequence according to the order in parent 2

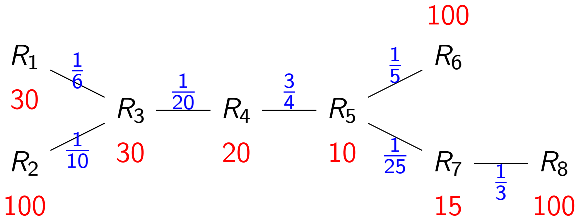
Subsequence exchange for ordinal number encoding

- Swap two sequcences of same length
- What if we get duplicates?

Subset exchange for ordered list encoding

- Find random subsequeces in both parents that have the same length and contain the same relations
- Exchange them to create two children

# Quick Pick, Genetic Algorithm



# Info

- Submit exercises to Andrey.Gubichev@in.tum.de
- Due July 7, 2014.