Resources:

- Recitation slides.
- Reference F1 scores for different levels of markovization.
- Remember, you don't have to match these, it all depends on your annotation choices. Some choice points matter more for smaller grammars like v=1, h=0 and don't matter as much for v=h=2. So if your F1 is lower than reference for smaller grammars but similar for v=h=2, don't worry about it.
- Notes for coarse-to-fine implementation. The notation might be confusing: in this note alpha is inside score and beta is outside, reversed compared to the lectures this year.

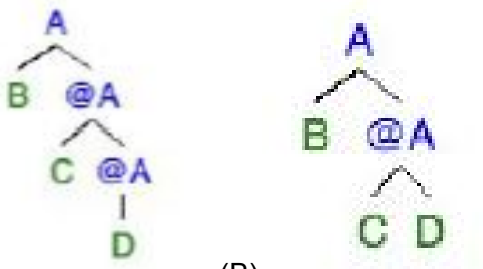
Tips to improve F1:

- When looping over unary rules, use `UnaryClosure.getClosedUnaryRulesBy{Child,Parent}` and not `Grammar.getUnaryRulesBy{Child,Parent}`. Unary closure would give you the most probable path between a pair of nonterminals (A, B), while the Grammar would only give A -> B.
- Don't forget to unroll the closed unary rules when you reconstruct your tree (by calling UnaryClosure.getPath). Note that unary closure includes reflexives too (A -> A), those rules need to be skipped in reconstructed tree.
- Try other annotation choices: doing parent annotation for pre-terminals vs. not, keeping around the ... in the horizontal annotation (@A^B->..._C_D) vs. not (@A^B->_C_D).

Tips for speeding up:

- You should loop over rules by child rather than parent. The reason is that CKY is bottom-up: when you consider a particular span [i, j) and a split point k, you already know have scores for all nonterminals over subspans [i, k) and [k, j), and you should use that information.
- Often certain nonterminals would be impossible over a particular span (have probability 0) -- you can skip over them when looping, or store a list of only the possible non-terminals as suggested in recitation. Accessing the grammar is the slowest part, you want to avoid doing that when possible.
- Same idea applies to tag score: check if they return negative infinity or NaN.
- Given the suggestions above, think what the optimal loop order is to allow for skipping impossible child nonterminals.

- For binary rules, would it make more sense to loop by right or left child? (Think of which of them would have a more sparse set of possible nonterminals, depending on whether your binarization is left- or right-branching. The one in the lectures is right-branching).
- Grammar size influences the speed a great deal. One choice point that reduces grammar size without reducing order of markovization is whether you avoid binarizing the rule further if it's already binary (this was mentioned in the recitation). Example: suppose your tree is A -> B C D. Binarization we showed in the lecture and the baseline code would produce (A) , but there is another way (B):



(A) (B)

This seems simple but has a great effect on speed (why?)

Tips for memory usage:

- For charts and backpointer arrays, use arrays of primitives, not objects. For most integer values, short would be enough instead of int (think of the number of symbols, rules, possible sentence lengths, etc.). For scores, float is enough.
- Keep the grammar symbols dimension innermost, otherwise you will end up with too many objects.
- You are only using half of your charts (a triangle). You can initialize the unused portion to null and then initialize the innermost arrays for the part you use (although preallocating might be faster).