

Journal - Komprese stromových struktur

V tomto deníku dokumentuji postup a experimenty, které provádím při vývoji metody pro kompresi stromových struktur v rámci mého projektu. Cílem je optimalizovat způsob, jakým jsou syntaktické stromy reprezentovány, aby bylo možné efektivněji ukládat a zpracovávat jazyková data.





Cíl projektu

Projekt se zaměřuje na vývoj a testování různých metod pro kompresi syntaktických stromů, které jsou výsledkem analýzy textu. Věnuji se zkoumání různých nástrojů pro **dependency parsing**, zpracování textu a tvorbě kompresních algoritmů, jako je gramatická komprese nebo metoda **RePair**.

Jak deník pomáhá

Každý zápis v deníku se zaměřuje na konkrétní denní pokrok, nové výzvy, řešení a testování nových metod. Je to pro mě způsob, jak sledovat vývoj projektu krok za krokem a zároveň poskytnout ostatním náhled na to, jak se projekt vyvíjí.



Zápisky obsahují:

-  **Experimenty** s různými nástroji pro syntaktickou analýzu.
-  **Testování metod komprese**, jako je gramatická komprese a RePair.
-  **Problémy**, na které jsem narazil při implementaci, a způsoby, jak je řešit.
-  **Plány do budoucna**, jak projekt posunout dál.

Každý zápis je podrobný a dokumentuje jak technické kroky, tak teoretické úvahy a rozhodnutí.

Prozkoumejte jednotlivé záznamy

Pokud máte zájem o podrobnosti, můžete si prohlédnout jednotlivé zápisky z deníku podle data. Každý zápis je zaměřen na konkrétní fázi vývoje a řešení, která jsem v daný den řešil.

-  [2025-02-19](#)
-  [2025-02-20](#)
-  [2025-02-23](#)

Tento deník je nejen záznamem pro mě, ale také způsobem, jak sdílet moje pokroky s ostatními a získat zpětnou vazbu na implementované metody.

Journal Entry - 2025-02-19

Na začátku jsem začal zkoumat různé způsoby, jak strojově sestavit stromové struktury vět nebo jednotlivých slov. Jako první jsem se zaměřil na projekt **MorphoDiTa**, který slouží k **taggingu** (tj. přiřazování slovním tvarům jejich gramatických kategorií, jako je slovní druh, pád nebo osoba) a k **lemmatizaci** (tj. převodu slov na jejich základní tvar – lemma).

Pro experimenty jsem vytvořil jednoduchý framework v jazyce **C#**, ve kterém pomocí základních příkazů testuji **tagger** a **lemmatizaci**. Zkoumám také, zda by části **lemmatizačního derivativního stromu** mohly být využity ke kompresi textu.

U menších souborů se může stát, že se velikost spíše zvýší, ale u větších textů by naopak mohla klesnout – to však musím experimentálně ověřit. V této fázi projektu hledám vhodný způsob, jak efektivně sestavit stromové struktury na základě získaných analytických dat. **Tagging se zatím jeví jako nejlepší přístup, ale potřebuji najít optimální kritéria pro jejich konstrukci.**

Našel jsem Dependency Parser [Parsito](#), který by mohl dokázat vytvořit strom. Musím se ještě více naučit jak funguje NLP a tyto algoritmy.

Parsito bohužel nefunguje na architekturách ARM64, zároveň je dlouhou dobu nepoužívaný. Release co se mi podařilo stáhnout je z roku 2016.

Dále jsem narazil na StanfordNLP, který je momentálně označený za deprecated, tudíž pro mě nejspíš nepoužitelný???

~~Zároveň mě napadlo, pokud se mi nepodaří rozjet model pro dependency parsing lokálně, mohl bych využít alespoň nějaké REST API. REST API není podporované.~~

Našel jsem knihovnu UDPipe, která má wrapper přímo pro C#, takže by mohla být jednoduše použitelná.

Journal Entry - 2025-02-20

Pomocí **UDPipe 1** a modelu pro angličtinu se mi podařilo úspěšně sestavit **syntaktický strom** z běžného anglického textu. Projekt byl poměrně náročný, ale nakonec se mi podařilo dosáhnout správných výsledků.

Během práce jsem narazil na **problém s tokenizací**, který ovlivňoval kvalitu **dependency parsingu** a vedl k nedostatečně přesným výstupům. Po několika experimentech se mi však podařilo problém vyřešit a získat správnou strukturu stromu.

Nyní se zaměřím na:

1. **Skládání vět do stromových struktur.**
2. **Testování různých metod komprese** pro syntaktické stromy.
3. **Porovnání a analýzu výsledků** – například ve srovnání s jinými metodami komprese.

Cílem je optimalizovat **reprezentaci syntaktických stromů**, aby bylo možné jazyková data efektivně ukládat a zpracovávat.

Použil jsem **UDPipe 1**, protože:

- **UDPipe 2** je stále experimentální a napsaný v Pythonu.
- **UDPipe 3**, který by měl opět nabídnout přívětivé API a bindingy, je teprve ve vývoji.

Journal Entry - 2025-02-23

1. Sestavení stromové struktury

Po úspěšném testování se mi podařilo sestavit stromovou strukturu z textových dat. Tento proces zahrnoval:

- **Parsování** vstupního textu.
- **Vytvoření stromové struktury**, kde každý uzel obsahuje určitou hodnotu z textu.
- **Levý a pravý podstrom** byly generovány na základě specifických pravidel vycházejících z textového formátu.

2. Komprese stromu pomocí gramatiky

Po sestavení stromové struktury jsem provedl kompresi pomocí **gramatiky**. Tento postup spočíval v:

- **Nahradit opakující se podstromy** jedinečnými pravidly.
- **Redukce velikosti stromu**, kde každý unikátní podstrom získal vlastní pravidlo (např. R1, R2 atd.), což vedlo k výrazné kompresi struktury.

3. Další možný krok: Použití metody RePair

Vzhledem k úspěšnosti gramatické komprese bych rád pokračoval v testování **metody RePair**. Tato metoda je známá svou efektivitou při:

- Hledání **opakujících se vzorců** v textových datech.
- Nahrazení těchto vzorců **symboly**, což vede k další redukci velikosti dat.

Plánuji implementovat metodu RePair a testovat její vliv na kompresi stromu, abych zjistil, zda poskytne lepší výsledky než aktuální metoda gramatické komprese.

4. Problém se seřazením dat v levé a pravé větvi

Jedním z problémů, na které jsem narazil, je potřeba **seřazení dat** v obou větvích stromu (levé i pravé). Tento problém může ovlivnit výsledky komprese, protože:

- **Neoptimalizované seřazení** dat může vést k **ztrátě kompresní účinnosti**.
- Momentálně **nejsem úplně jistý**, jak správně data uspořádat, aby komprese probíhla co nejefektivněji.

5. Další kroky

Pro řešení výše uvedeného problému plánuju:

- **Analyzovat možné přístupy** k seřazení dat, které by mohly optimalizovat kompresi.

- **Experimentovat** s různými metodami uspořádání dat ve stromě pro dosažení co nejlepšího výsledku.

2025-02-25 - Výzkum a implementace algoritmů pro kompresi stromových struktur

Dnes jsem se zaměřil na implementaci a výzkum různých existujících algoritmů pro kompresi stromových struktur. Prozkoumal jsem tři hlavní algoritmy: **DictionaryTreeCompression**, **FrequentSubtreeCompression** a **RePairTreeCompressor**. Každý z těchto algoritmů má své výhody a specifické využití v závislosti na druhu dat, která se komprimují. Tento výzkum mi pomohl lépe pochopit, jak každý z těchto přístupů funguje a jak je možné je využít pro optimalizaci komprese stromů ve svém projektu.



Algoritmy pro kompresi stromových struktur

1. DictionaryTreeCompression

Tento algoritmus je známý svou efektivitou při kompresi dat ve stromových strukturách, přičemž je běžně používán pro **kompresi XML dat** a dalších hierarchických datových formátů. Funguje tak, že vytváří slovník, který obsahuje opakující se podstromy. Každý podstrom je reprezentován klíčem ve slovníku, což umožňuje kompresi tím, že místo opakovaných podstromů se používá pouze jejich klíč. Tento přístup výrazně snižuje velikost dat, zejména pokud existují rekurentní vzory.

Možnosti využití:

- Tento algoritmus bych mohl použít pro kompresi stromových struktur, které vykazují vysokou míru opakování v jejich podstrukturních.
- V budoucnu by se mohl ukázat jako efektivní pro kompresi syntaktických stromů, pokud se budou vyskytovat opakující se vzory, například v dlouhých větách nebo textových blocích.

2. FrequentSubtreeCompression

Tento algoritmus se zaměřuje na **kompresi častých podstromů**, což znamená, že hledá podstromy, které se vyskytují často v celém stromu, a nahrazuje je jedinečnými identifikátory. Tento přístup je často používán v **bioinformatice** pro kompresi dat, jako jsou filogenetické stromy, a v dalších oblastech, kde se často opakují určité struktury.

Možnosti využití:

- Tento algoritmus by mohl být užitečný, pokud budu pracovat s rozsáhlými daty, kde některé podstromy nebo vzory struktury stromu mohou být velmi časté.
- V budoucnu se ukáže jako vhodný pro kompresi složitějších stromových struktur s vysokou mírou opakování, což je typické pro texty s mnoha podobnými větami.

3. RePairTreeCompressor

RePair je algoritmus, který se zaměřuje na **nalezení opakujících se vzorců** v datech a jejich nahrazení symboly, což vede k výrazné redukci velikosti. Tento algoritmus je oblíbený pro kompresi textů a **XML dat** a je známý svou efektivitou při hledání a nahrazování opakujících se podstruktur.

Možnosti využití:

- RePair je vhodný pro situace, kdy je potřeba efektivně komprimovat velké množství dat, a to zejména když se v datech nachází podobné podstruktury.
- Tento algoritmus by mohl být jedním z klíčových nástrojů pro mojí implementaci kompresního algoritmu, zejména pokud budu mít problém s velkým množstvím opakujících se vzorců v syntaktických stromech.

Implementace a experimenty

V rámci implementace jsem se nejprve zaměřil na základní verzi každého algoritmu:

- **DictionaryTreeCompression**: Začal jsem implementací jednoduchého slovníku pro ukládání opakujících se podstromů. Testoval jsem ho na několika příkladech textu, kde jsem hledal opakující se fráze.
- **FrequentSubtreeCompression**: Tento algoritmus jsem implementoval tak, že jsem prohledával strom a identifikoval podstromy, které se vyskytovaly častěji než ostatní. Tyto podstromy jsem nahradil identifikátory.
- **RePairTreeCompressor**: Tento algoritmus jsem implementoval s využitím principu iterativní komprese, kde se v každé iteraci hledají a nahrazují opakující se vzory. Implementace vyžadovala dostatečně efektivní způsob, jak zpracovávat a ukládat nalezené vzory.

Výsledky a zhodnocení

Během implementace jsem se hodně naučil o těchto algoritmech a jejich výhodách:

- **DictionaryTreeCompression** se osvědčil jako efektivní pro kompresi textových stromů, ale je méně efektivní při zpracování stromů s nižšími mírami opakování.
- **FrequentSubtreeCompression** je velmi silný při kompresi dat, kde se vyskytují časté vzory. Může být užitečný pro struktury s výrazným opakováním.
- **RePairTreeCompressor** se ukázal jako nejlepší pro moji aplikaci, protože je schopný najít opakující se vzory a komprimovat je velmi efektivně, zejména u velkých stromů.

Co dál?

V budoucnu bych chtěl:

- **Porovnat výkon** těchto algoritmů na reálných datech.
- **Vytvořit hybridní metodu**, která by kombinovala výhody jednotlivých algoritmů.

- **Testovat na větších datech**, abych zjistil, jak se chovají při vyšší složitosti a větší velikosti stromu.

Celkově jsem se naučil hodně o tom, jak algoritmy pro kompresi stromů fungují a jak je lze aplikovat na různé typy dat. Zatím se mi nejvíce osvědčil **RePair**, ale stále je prostor pro optimalizace a zlepšení.