HTMLPrettyPrinter

Cocu Matei-Iulian și
Tinică Alexandru Ștefan

UNIVERSITATEA DIN BUCURESTI
DEPARTAMENTUL DE INFORMATICA
INSTRUMENTE SI TEHNICI DE BAZA IN INFORMATICA
2024-2025

Outline

1) Descrierea problemei
2) Design soluție
3) Implementare
4) Experimente
5) Concluzii

Descrierea problemei

Problema implică **prelucrarea și afișarea structurii unui fișier HTML sub forma unui arbore ierarhic.** Scriptul trebuie să proceseze un fișier HTML dat ca input printr-o cale absolută, să elimine elementele redundante (cum ar fi comentariile sau spațiile inutile) și să creeze o reprezentare structurată a relațiilor dintre tagurile HTML. Rezultatul va fi un fișier text (format_tree.txt) care reflectă clar ierarhia și relațiile de cuibărire dintre elementele HTML.

Această problemă face parte din categoria **parsing-ului** și procesării documentelor ierarhice. În mod specific, abordează:

- Analiza sintactică a limbajului HTML.
- Reprezentarea structurii fișierelor marcate în format text.
- Optimizarea lizibilității fișierelor complexe pentru utilizatori sau alte sisteme software care interpretează arbori ierarhici.

Ce se urmărește?

Generarea unui fișier text care reflectă **structura** HTML într-un **format clar**, utilizând indentări și simboluri specifice ierarhiei:

- ├ pentru elementele din același nivel.
- └ pentru ultimul element dintr-un nivel.
- │ pentru indicarea continuității unui subarbore.

Scopul principal este îmbunătățirea lizibilității structurii HTML pentru utilizatori sau alte sisteme care analizează documente marcate.

De ce e nevoie de soluție?

Navigabilitate: Documentele HTML complexe pot deveni greu de urmărit fără o reprezentare structurată.
Depanare: O reprezentare clară a structurii poate ajuta programatorii să identifice erori de validare sau probleme de "nesting".
Automatizare: Soluția poate fi folosită ca bază pentru alte aplicații, cum ar fi generatoare de diagrame, validatoare de cod HTML, sau alte unelte de analiză a documentelor marcate.

Specificația soluției

Plecând de la cuvântul cheie din numele proiectului ales de noi, anume **“pretty”**, ideea este de a genera o reprezentare vizuală a structurii unui fișier HTML sub formă de arbore ierarhic într-o manieră simplistă, pentru a facilita înțelegerea relațiilor dintre elementele HTML. Scriptul prelucrează un fișier de intrare pentru:

- Eliminarea redundanțelor (spații inutile, comentarii, taguri de tip <!DOCTYPE>)
- Simplificarea tagurilor pentru a păstra doar denumirea lor esențială
- Crearea unei reprezentări ierarhice cu simboluri și indentări pentru fiecare nivel al structurii HTML

Această soluție abordează două provocări:

a) Prelucrarea liniară a documentelor HTML fără un parser dedicat.

b) Reprezentarea clară și lizibilă a structurii ierarhice într-un format text.

Caracteristici de utilizare (?)

Interfață simplă (un singur argument pentru script – calea fișierului HTML).

- Mesaje clare de eroare pentru fișiere inexistente sau corupte.
- Tehnice:
- Scriptul trebuie să ruleze pe orice sistem Unix/Linux cu utilitarele sed, grep, tr disponibile.
- Compatibilitate cu fișiere HTML de dimensiuni mari.
- Integrare: Poate fi folosit ca un pas intermediar pentru validarea structurii HTML înainte de parsare.

Use-case-uri minimale:

- Use-case 1: Utilizatorul specifică un fișier HTML existent, iar scriptul generează format_tree.txt cu structura sa ierarhică.
- Use-case 2: Utilizatorul utilizează un fișier HTML invalid (ex.: fără taguri de închidere), iar scriptul gestionează eroarea și indică problema prin mesaje clare de eroare pentru fișiere inexistente sau corupte.

Cerințe de sistem și mediu de dezvoltare

Sistem de operare:

- Linux/Unix (Ubuntu, CentOS, MacOS etc.).

Middleware și unelte (utilitare standard disponibile pe sistemele Unix):

- bash, sed, grep, tr.

Resurse necesare:

- Memorie: Minim 512 MB RAM.

Putere de calcul:

- Orice procesor modern capabil să ruleze un script Bash.

Nu sunt necesare biblioteci externe.

Spațiu minim pe disc pentru fișierele procesate (~10-20 MB).

Presupuneri, constrângeri și dependențe

Presupuneri:

- Fișierele de intrare sunt în format HTML valid.
- Utilizatorul are drepturi de acces pentru fișierul specificat.
- Tagurile HTML sunt bine formate și închise corespunzător.

Constrângeri:

- Scriptul nu validează conformitatea completă a fișierului HTML cu standardele W3C.
- Fișierele de dimensiuni extrem de mari (>500 MB) pot necesita mai mult timp de procesare.

Dependențe:

- Utilitarele grep, sed, tr trebuie să fie disponibile pe sistemul utilizatorului.
- Fișierele HTML trebuie să fie accesibile printr-o cale absolută.

Descrierea arhitecturii software folosite

Preprocesare fișier HTML:

- Eliminarea redundanțelor (spații, comentarii, taguri redundante) folosind un lanț de comenzi Bash (sed, tr, grep) pentru procesare linie cu linie.
- Scopul este de a reduce complexitatea structurală a fișierului HTML la un set minim de taguri relevante.
- Identificarea structurii ierarhice:
- Folosirea expresiilor regulate (grep -oP '<[^>]*?>') pentru extragerea tuturor tagurilor HTML din fișierul preprocesat.
- Algoritm bazat pe un stack logic (simulat cu array-uri în Bash) pentru gestionarea relațiilor de părinte-copil între taguri.

Construirea arborelui:

- Folosirea unei reprezentări incremental ierarhice a arborelui HTML, cu simboluri vizuale (├, └, |) și indentare corespunzătoare.
- Controlul indentării se realizează printr-un algoritm care urmărește nivelul curent al ierarhiei (indent_level) și verifică tipul tagurilor.

Alegerea mecanismelor și algoritmilor

Procesare liniară folosind comenzi Unix standard:

- Alegerea utilitarelor standard (sed, grep, tr) se bazează pe portabilitatea și ușurința lor de utilizare pentru procesarea textului.
- Acest mecanism este preferabil pentru proiecte care nu necesită parsare completă de HTML conform standardelor W3C.

Gestionarea structurii ierarhice cu un stack logic:

- Algoritmul bazat pe stive (stack-uri) este eficient pentru a construi structuri ierarhice, necesitând un spațiu de memorie minim proporțional cu nivelul maxim de imbricare al tagurilor HTML.

Criteriu	Abordare utilizată (Bash)	Alternative (ex.: Python/parsing libray)
Funcționalitate	Simplă, axată pe extragerea și procesarea liniară a tagurilor HTML	Parsare completă conform standardelor HTML (cu librării dedicate)
Performanță	Rapidă, ideală pentru fișiere mici și medii (;100 MB)	Mai lentă, dar mai robustă pentru fișiere complexe
Resurse utilizate	Minime (doar utilitare standard)	Necesită biblioteci externe și runtime Python
Complexitate	Cod simplu,accesibil și modificabil	Cod mai complex, cu funcții integrate, mai avansate

Alegerea abordării Bash este motivată de nevoia de simplitate, portabilitate și de faptul că scriptul nu necesită parsare completă, ci doar o reprezentare structurală a HTML.

Biblioteci software folosite și funcționalitățile lor

Proiectul utilizează "utilitare" standard disponibile în Bash pentru a evita dependențele suplimentare, fiind eficiente și ușor de integrat în script-ul bash:

- sed: Folosit pentru manipularea textului, cum ar fi eliminarea comentariilor, simplificarea tagurilor și eliminarea newline-urilor și spațiilor.
- grep: Utilizat pentru extragerea tagurilor HTML și filtrarea acestora în funcție de reguli definite (ex. identificarea tagurilor self-closing).
- tr: Folosit pentru eliminarea newline-urilor din fișierul de intrare.
- xargs: Normalizează spațiile și asigură manipularea corectă a tagurilor.

Descrierea mecanismelor/algoritmilor folosiți

a) Eliminarea redundanțelor:

Spații inutile:

- Se elimină cu tr -d '\n' (eliminarea newline-urilor) și sed 's/> *</></g' (ștergerea spațiilor dintre taguri).

Comentarii:

- Sunt identificate și eliminate cu sed -E 's/<!--.*?-->//g'.
- Taguri redundante (<!DOCTYPE>):
- Sunt filtrate cu grep -iqE '^<!DOCTYPE[^>]*>' și eliminate cu sed.

b) Extracția tagurilor:

- Tagurile sunt identificate printr-o expresie regulată grep -oP '<[^>]*?>', care asigură extragerea oricărui text închis între paranteze < >.

c) Construirea arborelui:

- Algoritm bazat pe indentare și simboluri:
- Se folosește un contor de nivel de indentare (indent_level) pentru a determina poziția curentă în arbore.
- Indentarea este realizată cu array-uri (indent_stack) care marchează dacă o ramură trebuie extinsă (1) sau încheiată (0).
- Simboluri vizuale:
  - ├: Indică un nod intermediar din arbore.
  - └: Indică un nod final al unei ramuri.
  - │ : Indică o conexiune verticală între niveluri.

Particularitățile adaptării mecanismelor/algoritmilor

a) Diferențe față de abordările similare:

Constrângeri:

- Scriptul nu face o validare completă a sintaxei HTML, ceea ce înseamnă că fișierele incoerente din punct de vedere sintactic aduc cu sine eroarea implementată pentru acest caz.

Simplificare:

- Față de un parser complet, soluția elimină toate atributele și conținutul dintre taguri pentru a reduce complexitatea și a accelera procesarea.

b) Limitările implementării:

- Nu gestionează cazuri speciale, precum taguri cu atribute complexe sau neînchise.
- Performanța poate scădea pentru fișiere foarte mari, deoarece procesarea este secvențială și nu paralelizată.

c) Avantaje ale adaptării:

- Eficiență ridicată pentru fișiere HTML standard și dimensiuni mici-medii.
- Flexibilitate: algoritmii pot fi extinși pentru a include alte tipuri de validări sau prelucrări (ex.: validare W3C).

Probleme tehnice întâmpinate

Dificultatea de a diferenția între tagurile self-closing și celelalte:

- Unele taguri HTML pot apărea ca self-closing sau normale în funcție de context, ceea ce a făcut identificarea lor mai complexă.

Gestionarea tagurilor imbricate:

- Crearea unei structuri arborescente vizuale cu simboluri și indentare corectă a fost provocatoare în cazul fișierelor cu niveluri de nesting ridicate.

Eliminarea corectă a spațiilor și comentariilor:

- Fișierele HTML mai complexe pot include spații neregulate și comentarii întinse pe mai multe linii, ceea ce a necesitat expresii regulate robuste.

Compatibilitatea între medii diferite:

- Comportamentul unor utilitare precum sed poate varia între sistemele Unix (GNU vs BSD).

Soluțiile adoptate pentru problemele tehnice

Pentru diferențierea tagurilor self-closing:

- S-a folosit o listă explicită de taguri self-closing, verificând fiecare tag în raport cu această listă.

Pentru gestionarea tagurilor imbricate:

- Algoritmul bazat pe un stack logic a fost implementat pentru a urmări relațiile de părinte-copil. Array-urile au fost utilizate pentru a simula structura unui stack.

Pentru eliminarea corectă a spațiilor și comentariilor:

- S-au utilizat expresii regulate adaptative pentru diferite cazuri, cum ar fi comentarii extinse (<!--.*?-->) sau spații dintre taguri (> <).

Pentru compatibilitatea între medii:

- Scriptul a fost testat pe multiple sisteme Unix pentru a asigura portabilitatea. Comenzi alternative (ex. awk sau perl) au fost luate în considerare ca fallback-uri.

Experimente

- Pentru a testa performanța codului, pe partea hardware, a fost utilizat un laptop cu un procesor Ryzen 9 6900hs (8 nuclee, 16 fire de executie) care rulează windows 11 23H2, cu o memorie de 16GB RAM DDR5@4800MT și 1TB de stocare ssd nvme. Evident, pentru a rula scriptul bash s-a folosit o mașină virtuală, în cazul nostru WSL (cu Ubuntu), care oferă posibil cea mai performantă și minimalistă interfață.
- Utilizare de biblioteci/comenzi de bază din Linux, scriptul are o funcționalitate valabilă pe o multitudine de distribuții Linux fără probleme, scriptul fiind single-threaded, de asemenea.

Experimente

- Fișierul de intrare este de tip HTML, și contine o structură generică de pagină web. Complexitatea din punct de vedere al timpului ar trebui să fie echivalentă cu complexitatea funcțiilor precum grep, algoritmul folosindu-se intens de expresii regulate.
- Ținând cont că majoritatea fișierelor de acest tip sunt relativ scurte, iar algoritmii folosiți sunt oricum eficienți, am reusit să observăm un runtime suficient de mic cât să poată fi perceput de oameni ca fiind "practic instant". Astfel, o optimizare a algoritmului nu ar aduce schimbări semnificative, ținând cont de timpul liniar de execuție al acestuia.

Concluzii

Librăria din Babel: pagina 158 al volumului 8 din raftul 1 al zidului 1 al hexagonului

063se9o3w51299yxm3urclsyg8q8fdwg07dbiq0s6ie4n3mqexlxe028fb371vcpol5b19awy546cuv9otuxl4ytp5toplfvs5p09vc8uurw8dmzv5vjce4wk8vk9b5nm8h7invpnzuxs00zs0y9fg
ad2iai13h8l4s36hoa0q55nixotk4xva3qzux07vrrvpzi37cuya60sieyjfu1521wxwjs7sghmu1v5szorbpxh0o2f5m1n3eoozmxam5pxf771s3ralk6yja86qqg05o9xhlxb2b4c2kzlfiwgqtfki0alff
2qy0molrpljbs460k8h6j9ikwqk3ps771dpgji6zzqn9hvq9hvnvi2wez4dvjtchxwn6mddtkfib1m80286wn9zh7hnecucwij6a750n5hcc9sm9l7in9zqclo7a0mrrsgxply5m7mi3eujxnggzzvds
apnyzvzv9qho0i2y7d4gl39l57w8uaertwj67yh8flb0a9p2z367a1f0vwhr52htpdngduu4hmvg2wr1kj9qosxnbtubna8o6pttgxh43bt5nz4gwa5d3h5e63goi0yw8r5qkfdrgh4tnlq6dxpds74f
dcucf2iax0k4w0vjy5gd6hnb34myfx4zt1sjiixewzf2uxy53f4vbcz01lhyz9bcmrulixgwmoww4wgqzgqy046jynmxtv1xvzmvlecsf6vn80m8s8tesww36ayypyulov0woc1j3vdtzmxs686q6fk
gqua1gfyyzf2gymsppul8ox8p96xxkx8uzvuq34nl4rgqjg8me80jignldlg3c1ilimnh5ml5j34ws98qjbv7ty4khm6fthwf0ptcjro8d1zkcb4houttry719chqhw7lmmt2djejr8rdz7ctgk5jrdez3w2l
npvf8umf45kp4inppxhkjzn6pwsqb08t3huhxjm9vvvul1dcjdeiqkexwk0oc4ozgqojirepe050tgyjm66zrq29ba1syv6lh8xt30bft4bu1ictkh4d6d25trpcy0erf2px9rc1k4cc8p7r2u7j23k7zatz8
l26do9rxni5d67hgbayhleam2hgd6zwgjasyo6fe09r5dw145092tylylke4l0oe1gyfp8tdk4wyhz0qtmptiurwxkih4fbng8lg5lx0m12cp8zr2e88e8w95hq6doz0nwao17tnaqr32mr8aezsj9yq
w4ou7a9s3jjs80ppyse44491kxgboj1e7gyysl50h1t47aa68x38lczqp2vpenlrbz5ewueh1487btcx2o1qwog37a9p3cok2fz6tkhc0vlpnmmmouvx0ggyrkyy3svjs9xr7i6k4a5n96e0hdav8v
bp5g950vujfeoqlqry87espzd7uiqs0cilaslosmgaklsipbqi64ttqedhd78blcfj607px45ejc13z5nlnavk0qrfptj952dsdsoym03khmt0q595gxdij774xtajzvejvzx2b9btl3ltx9rvy21rnq4kefyawsc
d6dlpnf7fwhdi7bcw26wyuuhff632rlkchh70b55gxaf5u7kie3vp4qc6cziz9uq8emlz6wv6mrrkhu4d69iziqoqrum3x4rp2p08jfe0w33jh8l682qebdv7rvcqlwc25vxdt7qj66aw2dxkpm9188h
0d1b2keerg0sy65f6mlk59fa6ee3djymfj2d3zkdufuo9dumeltjc0pa1qf0wvoz9niqg2whkr0qidbbdcsgma13a814nqip5w8p0b13vpmfbyqw9w5vpaburiv2ppcql2czvlp0mal4ze51xdarvu
889ekyre4owl5dywmh1dvqxb87cvodcp6b0ydh2qo6ph3xhk8j9wu4e4zu8u1xxjcj182hwf5a4o0k290exx4mn2hslpq4uekuc87prcnivil5eytmx5lvfom7cyj8lgbeojn8jxp4qu0900suwf65
vyo4bo9kawuizi1cuaq559c9k4vmgm7xkcxwnk37484249dg8gzs53cxa435lzxvae6bd6k6a7k43ypa8d8mzbn28ucnxfd6400lt3f2kdyyaboryumwsk79yy45nrcew13ey5ceytdae1hlcfut
8jmmlg22uo2xhn7wbkjbsfzy9ynlew5gqwrzs4coch9g5ivs6mhj4dco4hfiz82iis6eby5qvgbzh8ygyh2nn91qdtapyphtiz2rgl3nr7sz07eygo8w5mk311e43gg0gzy9i9ejiad6yxom19g6xfun
8eeryxt8vdvj6do3ldz315izagpaumfkr7ua5b76i0k0eu244s4jpljvtdnysh5e7gk9fvrx6ghg1vk4qe08fqcgpn30b0mqkxxvndk9osb68o3dmi9rsc9b9okmfyhxb5rxsu6lajysj69xlyplulqarbxa
vze1gm07g97en2fj8zfqplchhf32ukav1ge0jzf0p742hx981t6scsaksxs1ndi2gh1kpsjawpm3z6gwsdhn6797tlokaq5h8acagulbh1lg68cism4smozkgnhyk3v626jpd7j96t1m131icr3musp
c1flh8mn9ehh6ignxdb4k6whj5nrsfrufh4km0pdo46b7s6knsknwctudyb5tebbafp8bd038y8x71c1c93gd152pjbd8pgd86aepdh7udvag7fowixcyz559tgyb0i0rvd6n7hkpu13dd7zisgz55
amuqj0p7cwbbu50fz6gatztua2snmbvsjj48x33dyf2o4r6vrx8gqyabm6q5hne4lggy4e5i3u87pfqhrl7e00p1tri427bbzvfyb67vu3ftt75c9vq6ud9la1xteluyiwlgb91iyjdy63xvxajakwn7imcia
nrkxnfqirqionetlelxbk3v4swrwbu0qt08w1lm2z6x1p4ackcbfrh0g7mswuklivdz95xv1mc9h4hu8dcasiw5whno41vwfnsxeuibtslymyp72d5um6w573frkwnxxdvwd0dxt60k1q096nutkua
w6gpmzq18hz8a9hh6o7gdn47sayqcwh3llsmwd0g6h2z09bkdbh32c7f7bgu229bvhslck1nn0s61z16tco2l6so0v1hnrft6fkmr79a7vqj8pe8xvtinh130fgmca79tnearc7t89p5qyyfitww5er
h03hgebtss9ptll7ulew0shmn2jq3nb9m6qvrm6yscxcmd3bx