Dokumentace k zápočtovému programu

Hledání nefunkčních odkazů do zadané hloubky

Anotace

Program slouží k prohledání zadané webové stránky a odkazů z ní vedoucích do určité hloubky. Průběžně vypisuje každý odkaz, který prohledá, a zda je funkční nebo ne v podobě stavového http kódu. Nakonec program vypíše seznam nefunkčních odkazů a jejich stavový http kód .Vzhledem k povaze programu je nejefektivnější prohledávat odkazy do relativně nízké hloubky, jelikož z každé stránky může vést i přes 100 odkazů na jiné stránky. Program je ovládán po spuštění souboru main.py, kde uživatel zadá webovou stránku, kterou chce prohledávat, a hloubku, do které bude program prohledávat

Algoritmus

Algoritmus použitý v tomto programu je BFS (breath-first-search), neboli prohledávání do šířky. Na realizaci BFS se využívá fronta. Fronta je FIFO datová struktura, což znamená, že prvky jsou odebrávány z fronty v pořadí, v jakém byli přidány, funguje tedy jako reálná fronta. Každý prvek fronty je zpracován a jeho případní potomci jsou přidáni do fronty. V našem případě si můžeme představit, že budeme prohledávat jakýsi strom webových stránek, kde prvek bude webová stránka a jeho děti budou odkazy, které z ní vedou. Pomocí BFS budeme tento strom procházet po vrstvách, tedy nejdříve projdeme zadanou webovou stránku, a poté stránky na které se dá dostat na 1 prokliknutí, poté na 2 prokliknutí a tak dále až do zadané hloubky. Algoritmus bychom mohli popsat takto:

1) vytvoříme frontu, do které prvek ve tvaru [url, cesta, hloubka]

2) dokud není fronta prázdná:

3) vyber z fronty nejstarší prvek a z něj vezmi url, cestu a hloubku.

4) vytvoř html dokument pomocí url+path a v něm najdi všechny tagy <a>

5) z <a> tagů vyber pouze url adresu

6) pokud jsme adresu ještě neviděli, vytiskneme ji a zkontrolujeme, zda funguje

7) pokud nefunguje, zaznamenáme si její url a http kód odpovědi

8) přidáme do fronty její potomky

Diskuse výběru algoritmu

Vzhledem k povaze zadání vyhledávat nefunkční odkazy po hladinách byla volba algoritmu vcelku jednoznačná. Pokud bych nechtěl implementovat přímo BFS, pak by řešení pravděpodobně stejně směřovalo podobným směrem. Mohl bych například mít 2 pole, ve kterých bych si držel aktuální hladinu a následující hladinu a po každé iteraci bych za současnou hladinu nastavil následující a následující bych nastavil na prázdné pole. Řešení by to bylo tedy velmi podobné, pouze by mohlo být hůře čitelné či méně výkonné. Další, dle mého názoru ještě horší, varianta je Prohledávat každý odkaz pomocí DFS. Tento postup by určitě také fungoval, ale hůře by šlo například tisknout odkazy postupně po vrstvách jako dělá současný program.

Program

Program je rozložen do dvou souborů main.py a broken_links.py. Je zde importováno několik knihoven, jmenovitě: requests, BeautifulSoup, urllib.request ,urljoin z knihovny urllib.parse a pprint. Jejich funkci vysvětlím při průchodu programem.

Začněme pomocným souborem broken_links.py. V tomto souboru jsou dvě funkce. Funkce check_link() slouží v hlavním programu pro výpis http kódu odpovědi webové stránky na náš request + zpráva, která pochází buď přímo z requestu, nebo je předepsaná, jelikož například odpovědi kódu 400 bývají velmi dlouhé.

Poté se zde nachází funkce return_status_code(), která pouze vrátí kód http odpovědi. Používá se na určení, zda má být odkaz přiřazen do seznamu nefunkčních odkazů.

V hlavním souboru main.py se nejprve vyřizuje input tak, aby vše fungovalo, tedy aby například uživatel nezapomněl napsat celou adresu atd. Poté přidáme zadanou webovou stránku do fronty a začneme s DFS. DFS již bylo vysvětleno v sekci Algoritmus, zde tedy hlavně popíšu, jak nám výše zmíněné knihovny pomáhají při tvorbě programu. Nejprve využijeme knihovnu urllib.request, pomocí které získáme html webu, který chceme prohledat. Toto html poté předáme do objektu BeautifulSoup z knihovny beautifulsoup a dostaneme objekt, který má přesně vlastnosti, které potřebujeme, například umí najít v html všechny tagy typu <a>. Z Nalezených <a> tagů vyjmeme href parametr a pomocí knihovny urljoin spojíme s url a dostaneme náš chtěný odkaz na stránku, kterou chceme zrovna prohledat. Poté už pouze zkontrolujeme funkčnost odkazu a pokud zjistíme že nefunguje, přidáme ho do slovníku, který nakonec vrátíme. Nakonec slovník vypíšeme pomocí funkce pprint, která zajistí hezké formátování.

Alternativní programová řešení

Alternativně bychom mohli brát input přímo z terminálu, ale riskovali bychom tím, že uživatel spíš udělá chybu při zadání odkazu. Myšleno tak, že při zadávání inputu z terminálu má uživatel pouze jeden pokus na správné zadání adresy webové stránky. Mohli bychom problém také řešit rekurzivně. Program by také mohl vylepšit asynchronní přístup k prohledávání webových stránek. Stylem, jakým je to nyní naprogramované, procházíme vždy jednu stránku po druhé. Pokud bychom jich dokázali procházet více najednou, určitě by se čas běhu aplikace značně zlepšil.

Reprezentace vstupních dat

Vstupní data jsou pouze dva inputy od uživatele. První je adresa stránky, kterou chceme prohledat a druhý je hloubka, do které chceme zadanou adresu prohledat. Chyba je zde možná pravděpodobně pouze u zadání adresy stránky. Ta musí být zadána celá a bez chyb, tedy včetně http/https a www, nic se nesmí vynechat. Příklad správně zadané adresy je : https://www.youtube.com. U zadání čísla stačí dbát pouze na to, že opravdu zadáme číslo, ne žádný text nebo cokoliv jiného.

Reprezentace výstupních dat

Výstupní data jsou vrácena po ukončení běhu programu. V terminálu se objeví seznam stránek v závorkách, u kterých byla zjištěna chyba. Jednotlivé body výstupu mají tvar *adresa webové stránky : kód chybové odpovědi.*

Průběh práce

Práce probíhala velmi plynule a jednoznačně nad má očekávání. Vzhledem k hezkému zadání jsem měl nápad na vypracování prakticky ihned hotový. Jediné, co mě děsilo, bylo, že nedokážu správně procházet stránku po stránce a předávat si mezi vrstvami odkazy. To nakonec zabralo necelou hodinu hledání správných knihoven, které napomáhaly přes ve funkcionalitě, kterou jsem potřeboval. Nakonec vše zapadlo do sebe, jak jsem si představoval i s tím, že program je poměrně krátký.

Závěr

Líbilo se mi vyzkoušet i praktičtější stránku pythonu, která bude pravděpodobně pro můj budoucí život čím dál důležitější než krásné algoritmické úlohy. Také mi přišla hezká náročnost tématu. Nešlo o nic složitého na nastudování, avšak pokud si s tím člověk chce vyhrát, možnosti jsou zde prakticky neomezené. Můžete se snažit co nejhezčeji vizualizovat průchod, zlepšit UI, nebo se snažit o co nejrychlejší běh aplikace. V tom mi přijde toto téma tak hezké. Je praktické a může být vhodné jak pro lidi, kteří chtějí pouze udělat věc v zadání, tak pro lidi, kteří chtějí využít svoji představivost naplno.