UNIVERSITEIT GENT

Lineaire compressie met LZ77 en deelstring bomen

Project Algoritmen en Datastructuren III
Academiejaar 2017-2018

1 Opgave

De bedoeling van het project is om een lineair algoritme voor LZ77 te implementeren door gebruik te maken van het algoritme van Ukkonen voor deelstring bomen. Het LZ77 algoritme met deelstring bomen staat beschreven in hoofdstuk 4.3.1 van de cursus. Het algoritme van Ukkonen staat uitgelegd in hoofdstuk 3.2.1.

Het project bestaat uit 2 delen met een afzonderlijke deadline. Het eerste deel bestaat uit het implementeren van het algoritme van Ukkonen. Voor het tweede deel moeten jullie LZ77 met deelstring bomen implementeren. Na de deadline van het eerste deel zullen jullie een implementatie van het algoritme van Ukkonen krijgen zodat het tweede deel onafhankelijk van het eerste geschreven kan worden. Het zal wel nodig zijn om deze implementatie lichtjes aan te passen zodat de waarden $beg(t)$ uit het LZ77 algoritme ook worden bijgehouden. Het staat jullie natuurlijk vrij om jullie eigen implementatie uit deel 1 te gebruiken als die goed werkt.

Deel 1: het algoritme van Ukkonen

Voor het eerste deel is het dus de bedoeling dat jullie een implementatie schrijven van het algoritme van Ukkonen. Let er hierbij op dat het algoritme moet werken in lineaire tijd en dat het geheugengebruik ook lineair moet zijn. Je mag ervan uitgaan dat alle tekens uit de invoer ASCII tekens zullen zijn.

Om jullie code eenvoudig testbaar te maken, wordt er gevraagd om een functie te schrijven die de deelboom uitschrijft naar stdout. Hierbij moet de suffix boom uitgeschreven worden in het formaat dat hieronder beschreven staat.

Begin met iedere top diepte eerst oplopende IDs te geven startend vanuit de wortel die ID 0 krijgt. De kinderen worden hierbij lexicografisch alfabetisch volgens het label van

de bogen die er naartoe leiden overlopen.

Schrijf vervolgens per top met n kinderen een regel uit die er als volgt uit ziet:

niet-blad: $\texttt{i} \ @ \ \texttt{start}_\texttt{i} - \texttt{eind}_\texttt{i} = \texttt{c}_1 : \texttt{k}_1, \texttt{b}_1, \texttt{e}_1 \mid \texttt{c}_2 : \texttt{k}_2, \texttt{b}_2, \texttt{e}_2 \mid \cdots \mid \texttt{c}_\texttt{n} : \texttt{k}_\texttt{n}, \texttt{b}_\texttt{n}, \texttt{e}_\texttt{n}$

blad: $\texttt{i} \ @ \ \texttt{start}_\texttt{i} - \texttt{eind}_\texttt{i}$

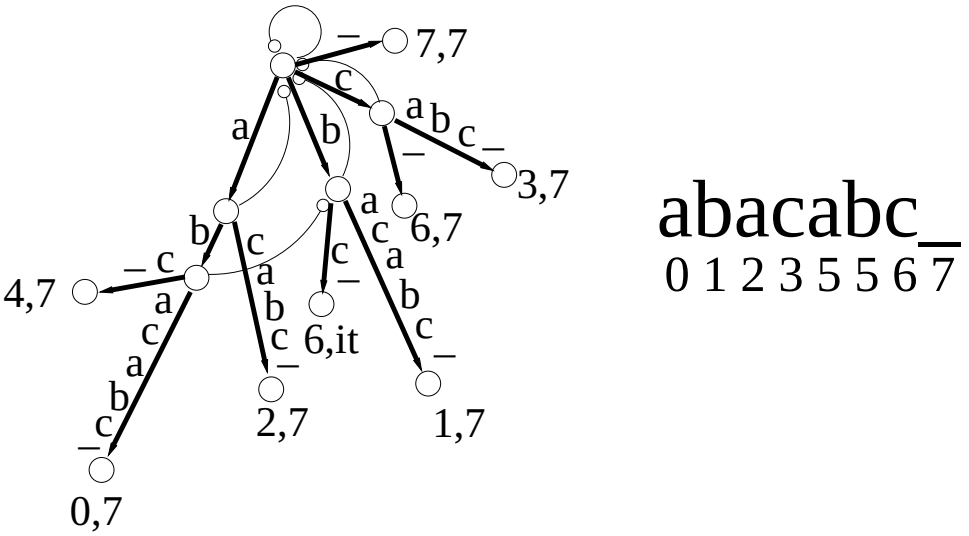
waarbij:

- `i` de diepte-eerst ID van de top is;
- @ het letterlijke teken "@" is;
- $\texttt{start}_\texttt{i}$ de beginindex is van de deelstring die de top voorstelt
- $\texttt{eind}_\texttt{i}$ de eindindex[1] is van de deelstring die de top voorstelt
- = het letterlijke teken "=" is;
- Na het = teken komt voor niet-bladeren een door "|" gescheiden lijst van de niet-NULL kinderen van de top gesorteerd naar stijgende $\texttt{c}_\texttt{j}$. Voor het `j`-de kind is:
  - $\texttt{c}_\texttt{j}$ de numerieke ASCII waarde van het eerste karakter van de boog naar het `j`-de kind
  - $\texttt{k}_\texttt{j}$ de diepte-eerst ID van dit kind
  - $\texttt{b}_\texttt{j}$ de beginindex van de deelstring die de *boog* voorstelt
  - $\texttt{e}_\texttt{j}$ de eindindex van de deelstring die de *boog* voorstelt

Indexen zijn steeds te tellen vanaf 0. Gezien de wortel de lege string voorstelt, kun je geen begin en eindindex nemen. Gebruik spatie (" ") als $\texttt{start}_0$ en $\texttt{eind}_0$. De volgorde van de lijnen doet er niet toe. Suffix links moeten niet in de output zitten.

Merk op dat de bladeren steeds suffixen voorstellen, hun $\texttt{eind}_\texttt{i}$ moet dus steeds de laatste index van de string zijn.

[1] Grens inbegrepen



Figuur 1: Suffixboom voor tekst “abacabc_”

Als je weet dat de ASCII waarden van _, a, b en c respectievelijk 95,97,98 en 99 zijn, wordt de deelboom uit Figuur 1 voorgesteld door:

```
 0 @  -  = 95:1,7-7 | 97:2,0-0 | 98:7,1-1 | 99:10,3-3
 1 @ 7-7
 2 @ 0-0 = 98:3,1-1 | 99:6,3-7
 3 @ 0-1 = 97:4,2-7 | 99:5,6-7
 4 @ 0-7
 5 @ 4-7
 6 @ 2-7
 7 @ 1-1 = 97:8,2-7 | 99:9,6-7
 8 @ 1-7
 9 @ 5-7
10 @ 3-3 = 95:11,7-7 | 97:12,4-7
11 @ 6-7
12 @ 3-7
```

Deel 2: LZ77 met deelstring bomen

Hiervoor zullen jullie dus een werkende implementatie van het algoritme van Ukkonen krijgen. Deze zal wel aangepast moeten worden zodat de waarden $beg(t)$ ook bijgehouden worden. Het LZ77 algoritme moet in staat zijn om alle tekst die enkel uit ASCII tekens bestaat te comprimeren en terug te decomprimeren in lineaire tijd door gebruik te maken van deelstring bomen. Je mag ervan uitgaan dat alle tekens uit de invoer ASCII tekens zullen zijn. Bij het decoderen zul je er rekening mee moeten houden dat je binaire data binnenkrijgt.

De output van de LZ77 compressie is een drietal (p, l, x). Hierbij is

- p de index van het eerste teken in de langste match die je wilt versturen (`uint32_t`),
- l is de de lengte van deze match (`uint32_t`),
- x het teken dat volgt op de match (`uint8_t`)

Dit 3-tal stuur je in volgorde door als een opeenvolging van een `uint32_t`, een `uint32_t` en een een `uint8_t`.

De datatypes `uint32_t` en `uint8_t` kan je halen door `<stdint.h>` te includen.

Het drietal $(1337, 4294967295, \texttt{A})$ schrijf je dus hexadecimaal uit als:

$$\underbrace{\texttt{39 05 00 00}}_{1337}\ \underbrace{\texttt{ff ff ff ff}}_{4294967295}\ \underbrace{\texttt{41}}_{A}$$

De input die je mag verwachten voor de decoder is dezelfde als de output van de encoder. In de praktijk zou je de output van het LZ77 nog eens comprimeren met Huffman. Maar voor dit project doen we dit niet.

Het geheugengebruik van een suffixboom voor een string van lengte n die je opbouwt met het algoritme van Ukkonen is $\mathcal{O}(n)$. Als je aan het comprimeren bent, werk je typisch met grote bestanden die soms niet volledig in jouw werkgeheugen passen. Experimenteer daarom met verschillende varianten van het algoritme. Je kunt bijvoorbeeld eens nagaan wat het effect is als je suffixboom niet meer de volledige geziene tekst bevat. Zo kun je bijvoorbeeld stoppen met het updaten van de suffixboom na het inlezen van x karakters. Of wat gebeurt er als je om de x karakters met een nieuwe suffixboom start? Eigen ideeen zijn zeker en vast ook welkom.

2 Verslag

Op het einde zul je ook een verslag moeten indienen dat zowel deel 1 als deel 2 beschrijft. Vermeld hierin zeker de verschillende implementatiekeuzes die je hebt gemaakt, welke testen je hebt uitgevoerd en voer ook enkele nuttige tijdsmetingen uit.

Bespreek ook de resultaten die je bekwam bij de verschillende varianten die je op het einde van deel 2 uitgeprobeerd hebt. Geef tijdsmetingen en verklaar waar de verschillen vandaan kwamen. Denk na over op welk soort tekst welke variant goed zal presteren in termen van compressie en uitvoeringstijd.

3 Specificaties

3.1 Programmeertaal

In de opleidingscommissie informatica (OCI) werd beslist dat, om meer ervaring in het programmeren in C te verwerven, het project horende bij het opleidingsonderdeel Algoritmen en Datastructuren 3 in C geïmplementeerd dient te worden. Het is met andere woorden de bedoeling je implementatie in C uit te voeren. Je implementatie dient te voldoen aan de ANSI-standaard. Je mag hiervoor gebruikmaken van de laatste features in C11, voor zover die ondersteund worden door `gcc` op `helios`.

Voor het project kan je de standaard libraries gebruiken; externe libraries zijn echter niet toegelaten. Het spreekt voor zich dat je normale, procedurale C-code schrijft en geen platformspecifieke APIs (zoals bv. de Win32 API) of features uit C++ gebruikt. Op Windows bestaat van een aantal functies zoals `qsort` een "safe" versie (in dit geval `qsort_s`), maar om je programma te kunnen compileren op een unix-systeem kan je die versie dus niet gebruiken.

Wat je ontwikkelingsplatform ook mag zijn, test zeker in het begin altijd eens of je op Helios wel kan compileren, om bij het indienen onaangename verrassingen te vermijden! Gebruik ook de scripts op Minerva!

3.2 Input/Output en implementatiedetails

Voor de in- en uitvoer gebruiken we de standaard `stdin` en `stdout` streams.

Voor het eerste deel van het project schrijf je een algoritme dat via stdin de tekst binnenkrijgt en vervolgens naar stdout de beschrijving van de boom op het einde uitschrijft.

Je programma moet dus bijvoorbeeld als volgt gebruikt kunnen worden:

```
$ cat inputtekst.txt
abacabc_
$ ukkonen < inputtekst.txt
  0 @  -  = 95:1,7-7 | 97:2,0-0 | 98:7,1-1 | 99:10,3-3
  1 @ 7-7
  2 @ 0-0 = 98:3,1-1 | 99:6,3-7
  3 @ 0-1 = 97:4,2-7 | 99:5,6-7
  4 @ 0-7
  5 @ 4-7
  6 @ 2-7
  7 @ 1-1 = 97:8,2-7 | 99:9,6-7
  8 @ 1-7
  9 @ 5-7
 10 @ 3-3 = 95:11,7-7 | 97:12,4-7
 11 @ 6-7
 12 @ 3-7
```

Voor het tweede deel van het project schrijf je een programma[2] dat een optie mee kan krijgen. Als de optie `-c` wordt meegegeven is het de bedoeling dat de invoer die meegegeven wordt via stdin gecomprimeerd wordt en het resultaat van de compressie uitgeschreven wordt naar stdout. Als de optie `-d` wordt meegegeven moet de invoer die meegegeven wordt via stdin gedecomprimeerd worden en de originele tekst uitgeschreven worden naar stdout. Als de `-o` vlag wordt meegegegven moet jouw programma zijn geheugengebruik optimaliseren.

```
$ lz77 -c < data.txt > compressed
$ lz77 -d < compressed > data.txt

$ lz77 -o -c < data.txt > compressed
$ lz77 -o -d < compressed > data.txt
```

4 Indienen

4.1 Directorystructuur

Je dient één zipfile in via `http://indianio.ugent.be` met de volgende inhoud:

- `src/` bevat alle broncode (inclusief de makefiles).

[2] een tweede executable

- `tests/` alle testcode.
- `extra/verslag.pdf` bevat de elektronische versie van je verslag. In deze map kan je ook eventueel extra bijlagen plaatsen.

Je directory structuur ziet er dus ongeveer zo uit:

```
example/
|-- extra/
|   `-- verslag.pdf
|-- src/
|   `-- je broncode
|-- tests/
`-- sources
```

4.2 Compileren

De code zal door ons gecompileerd worden op `helios` met behulp van de opdracht `gcc -std=c11 -lm`. Test zeker dat jouw code compileert en werkt op helios voor het indienen. Tijdens het ontwikkelen mag je gebruik maken van een build tool naar keuze (bv. `make` op `helios`).

De ingediende versie dient een bestand met de naam `sources` te bevatten waar je de dependencies voor het compileren kan aangeven. Dit bestand bevat zowel voor het ukkonen als het lz77 algoritme een lijn die de dependencies van het algoritme oplijst. Zoals bijvoorbeeld:

```
ukkonen: ukkonen/main.c ukkonen/tree.c
lz77: lz77/main.c lz77/compress.c lz77/decompress.c ukkonen/tree.c
```

Op Minerva vind je scripts die nagaan of het compileren werkt en testen of een heel erg eenvoudig voorbeeld werkt. Als deze scripts op `helios` niet eindigen met exitcode 0 en de tekst "`OK`" wordt jouw code als onontvankelijk beschouwd.

4.3 Belangrijke data

Ten laatste op *zondag 18 november* 23:59:59 verwachten we dat je via indianio deel 1 van het project indient.

Voor *zondag 2 december* 23:59:59 verwachten we dat je via indianio je code van deel 1 en deel 2 indient.

De uiteindelijke deadline is *donderdag 6 december* om 17u. Dan moet ook jouw verslag af zijn. We verwachten dat je dit verslag zowel elektronisch op indianio indient als op papieren aan ons bezorgt. Het papieren verslag kan ingediend worden in lokaal 40.09.110.032 of tijdens de oefeningenles.

4.4 Algemene richtlijnen

- Schrijf efficiënte code maar ga niet overoptimaliseren: **geef de voorkeur aan elegante, goed leesbare code**. Kies zinvolle namen voor methoden en variabelen en voorzie voldoende commentaar.
- Op Minerva staan er in de map `project` enkele bash scripts. Deze scripts zullen jouw code trachten te compileren en minimale IO testen doen. Als deze op `helios` niet `OK` uitschrijven en eindigen met exitcode 0, dan zal jouw project met 0 op 4 beoordeeld worden. Er is een bash script per deadline.
- Het project wordt gequoteerd op **4** van de 20 te behalen punten voor dit vak, en deze punten worden ongewijzigd overgenomen naar de tweede examenperiode.
- Het is strikt noodzakelijk drie keer in te dienen: het niet indienen van de eerste tussentijdse versie of de code betekent sowieso het verlies van alle punten.
- Projecten die ons niet bereiken voor de deadline worden niet meer verbeterd: dit betekent het verlies van alle te behalen punten voor het project.
- Dit is een individueel project en dient dus door jou persoonlijk gemaakt te worden. Het is uiteraard toegestaan om andere studenten te helpen of om ideeën uit te wisselen, maar **het is ten strengste verboden code uit te wisselen**, op welke manier dan ook. Het overnemen van code beschouwen we als fraude (van **beide** betrokken partijen) en zal in overeenstemming met het examenreglement behandeld worden. Op het internet zullen ongetwijfeld ook (delen van) implementaties te vinden zijn. Het overnemen of aanpassen van dergelijke code is echter **niet toegelaten** en wordt gezien als fraude.
- Essentiële vragen worden **niet** meer beantwoord tijdens de laatste week voor de deadline.