UNIVERSITEIT GENT

Datacompressie

Project Algoritmen en Datastructuren III
Academiejaar 2016-2017

1 Inleiding

Compressie-algoritmen zijn alomtegenwoordig. Allemaal "zippen" we wel eens een bestandje om het door te sturen, of downloaden een archief met software of source code in. Ze worden vaak ook gebruikt zonder dat we het ons misschien direct realiseren: in Word- of ODF-bestanden, om webpagina's on the fly te comprimeren, in de PNG (lossless) en JPEG (lossy) bestandsformaten, enz.

In dit project zullen we proberen om een bestand met een vastgelegd formaat zo goed mogelijk te comprimeren. Hiervoor zullen we enerzijds een standaard algoritme uit de cursus toepassen en anderzijds een eigen algoritme ontwikkelen dat gebruik zal maken van de specifieke structuur van het bestand.

2 Opgave

2.1 Bestandsformaat

De bestanden die we willen comprimeren hebben een specifiek formaat. Het zijn **tekst**-bestanden die bestaan uit 1 enkele regel. Op deze regel vinden we een lijst van oplopende natuurlijk getallen die gecodeerd worden in het JSON formaat. Dit wil zeggen dat de regel begint met een open vierkant haakje (`[`) en eindigt met een gesloten vierkant haakje (`]`). Tussenin vinden we de getallen (als tekst) gescheiden door komma's. Een voorbeeld van een dergelijk formaat is bijvoorbeeld

```
[0,12,34,567,890]
```

Dit formaat zou bijvoorbeeld de output kunnen zijn van een API die een lijst van object id's teruggeeft van elementen die aan een bepaalde filter voldoen. Je mag er van uit gaan dat de waarde van het laatste getal steeds kleiner is dan 9,223,372,036,854,775,807, de maximale waarde van een *signed* 64-bit getal.[1]

[1] Tip: moest je in je zelf te ontwikkelen algoritme de binaire voorstelling van dergelijk groot getal nodig hebben, kijk dan eens naar het `long long` datatype

2.2 Standaard algoritme

In de cursus staan vier algemene algoritmen voor het comprimeren van gegevens (Huffman, LZ77, LZW en Burrows-Wheeler). Denk na welk van de 4 algoritmen waarschijnlijk de beste compressie zal opleveren voor het opgegeven bestandsformaat en implementeer dit. Bespreek in je verslag minstens de volgende zaken:

- Waarom je voor dat algoritme gekozen hebt en niet voor een van de andere.
- Of het algoritme voldeed aan je verwachtingen.
- De performantie (snelheid) van het algoritme
- Specifieke ontwerpsbeslissingen en eventuele afwijkingen van het algoritme zoals beschreven in de cursus.

2.3 Specifiek algoritme

Aangezien het bestandsformaat zeer specifiek is, kan de structuur waarschijnlijk uitgebuit worden door een specifiek algoritme dat enkel werkt op dergelijke bestanden om zo een betere compressie te bekomen. Bedenk en implementeer een dergelijk compressiealgoritme waarbij je rekening houdt met de compressiefactor en de tijd nodig voor het comprimeren en decomprimeren van de bestanden.

Je kan hierbij een combinatie maken van verschillende technieken of algoritmen, aangevuld met je eigen ideeën. Een niet limitatieve lijst van algoritmen die van pas zouden kunnen komen zijn: huffman encoding, delta encoding, run length encoding, binaire datavoorstelling, linear predictive coding, ...

Bespreek in je verslag uitvoerig de gekozen algoritmen en waarom je ze gekozen hebt.

2.4 Vergelijking algoritmen

Test en vergelijk de twee geïmplementeerde algoritmen uitvoerig in je verslag. Test verschillende bestandgroottes en bedenk *best-* en *worst-case* scenarios voor de twee algoritmes. Heb aandacht voor het rapporteren van je testresultaten. Zet niet gewoon een aantal grafieken op een blad, maar denk eerst na wat je wil aantonen met je resultaten. Maak je grafieken volgens de regels van de kunst en interpreteer deze ook. Zeg dus **niet** 'op de grafiek zien we de uitvoeringstijd voor stijgende invoergrootte' maar bijvoorbeeld eerder iets als '... voor invoer kleiner dan de cachegrootte zien we een constante uitvoeringstijd, vervolgens neemt de uitvoeringstijd lineair toe terwijl we ook hier een constant verband verwacht hadden. ...'

2.5 Versturen over internet

Stel dat we dergelijke bestanden zo snel mogelijk over het internet willen versturen, welke strategie raad je dan aan? Comprimeren zal de bestanden uiteraard kleiner en dus sneller te versturen maken, maar de compressie (en decompressie) kosten uiteraard ook tijd. Je mag er hier van uit gaan dat de nodige software om te coderen en te decoderen aanwezig is bij zowel ontvanger als verzender. Onderzoek voor beide algoritmen tot welke internetsnelheid (bitrate) het voordelig is om eerst compressie toe te passen.

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 III 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 C99, voorzover die ondersteund worden door `gcc` op `helios`.

Voor het project kan je de standaardlibraries 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!

3.2 Input/Output en implementatiedetails

Je moet dus twee programma's schrijven. Een programma dat je standaard compressie algoritme implementeert, en een ander dat je eigen algoritme implementeert. Beide tools hebben als eerste argument `-c` of `-d`, om het programma in respectievelijk de codeer- of decodeermodus te starten. De tweede parameter is de naam van het invoerbestand, de derde parameter de naam van het uitvoerbestand.

Je programma's moeten dus bijvoorbeeld als volgt gebruikt kunnen worden:

```
$ comprimeer1 -c data.txt compressed
$ comprimeer1 -d compressed data.txt
$ comprimeer2 -c data.txt compressed2
$ comprimeer2 -d compressed2 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).
- `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/
|   `-- standaard
|    `-- je code voor het standaard algoritme
|   `-- specifiek
|    `-- je code voor het specifiek algoritme
|   `-- eventuele gedeelde code'
|-- tests/
`-- sources
```

4.2 Compileren

De code zal door ons gecompileerd worden op `helios` met behulp van de opdracht `gcc -std=c99 -lm`. Test zeker dat je 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 specifieke file te bevatten waar je voor elk algoritme de dependencies kan aangeven. Zet deze file in de mappen die bij het algoritme horen. Deze file heeft de naam "sources" en bevat een lijn per algoritme,

gescheiden met een dubbelpunt van de dependencies (hierbij hoef je enkel de `*.c` files op te geven). Zoals bijvoorbeeld:

```
comprimeer1: standaard/main.c standaard/compress.c common.c
comprimeer2: specifiek/main.c specifiek/compress.c common.c
```

4.3 Belangrijke data

Tegen *zondag 30 oktober* verwachten we dat je via indianio een tussentijdse versie indient. Bij deze versie moet de code nog niet compleet zijn, maar je moet kunnen aantonen dat je er voldoende werk hebt aan besteed.

Tegen *vrijdag 25 november* verwachten we dat je via indianio je code indient. Zorg er zeker voor dat deze af is, want na deze deadline mag je code niet meer gewijzigd worden.

De uiteindelijke deadline is *donderdag 1 december* om 17u. Dan moet het verslag af zijn. We verwachten dat je dit verslag elektronisch indient op indianio en we verwachten ook een papieren versie van je verslag. Dit verslag kan ingediend worden in lokaal 110.016 op S9 (eerste verdieping, halfweg de gang, bureau Bart Mesuere) 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.
- 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.
- Het eerste deel is niet finaal. Je mag gerust na de eerste indiendatum nog veranderingen aanbrengen.
- 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.