UNIVERSITEIT GENT

Huffman compressie

Project Algoritmen en Datastructuren III
Academiejaar 2017-2018

1 Opgave

De bedoeling van het project is om voor de volgende varianten van Huffmann compressie het comprimeer en decomprimeer algoritme te implementeren en de verschillende varianten te vergelijken met elkaar. Zorg ervoor dat de algoritmes op alle bestanden werken, dus niet enkel op ASCII files. Voor het gemak werk je in de prefixboom best met tekens van 8 bit. De nummering die hieronder gebruikt wordt, zal ook gebruikt worden om mee te geven als argument aan het programma dat je zult schrijven.

1. Standaard Huffman
2. Adaptive Huffman
3. Adaptive Huffman met sliding window
4. Two pass adaptive Huffman
5. Bloksgewijs adaptive Huffman

De verschillen tussen de varianten zijn voornamelijk welk deel van de tekst gebruikt wordt om de boom op te bouwen en aan te passen. Hieronder beschrijven we enkele varianten in meer detail. Het standaard algoritme en het adaptive Huffman algoritme staan voldoende beschreven in de cursus, op die twee varianten zal dus niet verder ingegaan worden.

1.1 Adaptive Huffman met sliding window

Bij adaptive Huffman is de boom op ieder moment in het coderen/decoderen de optimale prefixboom voor het stuk tekst dat al verwerkt is. In 1983 heeft Donald E. Knuth een aanpassing voorgesteld die met een "sliding window" werkt. Als je met een sliding window van grootte l werkt, zal de boom altijd de optimale prefixboom voor de laatste l tekens voorstellen. Dit betekent dat bij het inlezen van een nieuw teken de frequentie van dat teken in de boom verhoogd wordt met 1, maar ook dat de frequentie van het

teken dat l tekens voor het nieuwe teken staat, met 1 verlaagd wordt in de boom.

Of meer formeel, stel dat jouw tekst $t = a_0, a_1, \ldots, a_n$ is en dat je met een sliding window van grootte l werkt. Na het inlezen van $i + 1$ tekens moet de boom de optimale prefixboom van de tekst $a_{max(0,i-l+1)}, \ldots, a_i$ zijn. Als het teken op positie $i + 2$ dan wordt ingelezen zal de frequentie van het teken a_{i+2} met één verhoogd worden en als $i - l + 1 \geq 0$, zal de frequentie van het teken a_{i-l+1} met één verlaagd worden in de boom. Over hoe de verlaging precies gedaan kan worden, kun je zelf nadenken of het opzoeken.

1.2 Two pass Huffman

Bij two pass Huffman wordt de tekst bij het comprimeren twee maal doorlopen. Tijdens de eerste keer zal, net zoals bij het standaard Huffman algoritme, de optimale prefixboom voor de volledige tekst worden opgesteld. Maar in de tweede stap, die dan ook werkelijk de tekst comprimeert, zal na het lezen van ieder teken de frequentie van dat teken met één verlaagd worden. Op die manier is de boom de optimale prefixboom van het stuk tekst dat nog moet komen, en niet zoals bij adaptive Huffman van het stuk tekst dat al geweest is.

1.3 Bloksgewijs adaptive Huffman

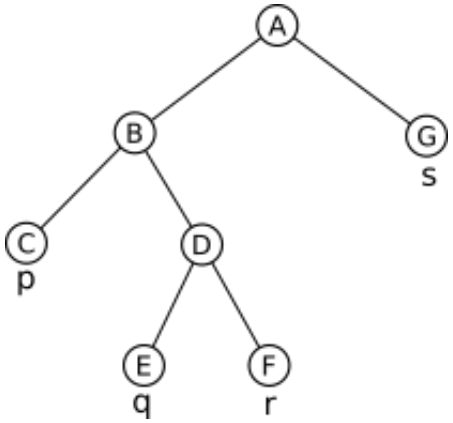
Bij bloksgewijs adaptive Huffman wordt de tekst opgesplitst in verschillende stukken, die dan afzonderlijk gecomprimeerd worden met het standaard adaptive Huffman algoritme. Je tekst wordt dus eerst in n blokken verdeeld, vervolgens wordt het eerste deel gecomprimeerd, dan het tweede met een volledig nieuwe boom, enz.

1.4 Voorstelling boom

Bij sommige varianten zal het nodig zijn dat het gecomprimeerde bericht een prefixboom bevat die nodig is tijdens het decoderen. Dit is onder andere het geval bij het standaard Huffman algoritme. Er zijn veel manieren waarop je de boom kunt coderen. Voor dit project leggen we het formaat van die prefixboom vast.

Het formaat bestaat uit twee delen c_1 en c_2. c_1 zal de structuur van de boom coderen, terwijl c_2 de symbolen die opgeslagen zijn in de bladeren bevat. Voor een gegeven binaire boom T is c_1 als volgt gedefinieerd. Als T bestaat uit één enkele top dan is

$$c_1(T) = 1.$$



Figuur 1: Eenvoudige Huffman boom

In het andere geval heeft T een linkerdeelboom T_1 en een rechterdeelboom T_2 en is

$$c_1(T) = 0c_1(T_1)c_1(T_2).$$

Dus een 0 gevolgd door de codes van zijn deelbomen. Intuïtief komt dit overeen met de boom diepte eerst te doorlopen en iedere keer je een blad tegenkomt een 1 uit te schrijven en anders een 0.

Voor de boom in Figuur 1.4 kan de structuur gecodeerd worden als 0010111. Dit resultaat bekom je als volgt. Stel dat T_X de deelboom is die de top met label X als wortel heeft. De volledige boom is dus gelijk aan T_A. Dan geldt voor de bladeren:

$$c_1(T_C) = c_1(T_E) = c_1(T_F) = c_1(T_G) = 1$$

Voor de andere toppen geldt:

$$c(T_D) = 0c(T_E)c(T_F) = 011$$
$$c(T_B) = 0c(T_C)c(T_D) = 01011$$
$$c(T_A) = 0c(T_B)c(T_G) = 0010111$$

Of intuïtiever, als de boom diepte eerst wordt doorlopen is de volgorde van de toppen A, B, C, D, E, F, G. Als je de bladeren voorstelt door 1 en de andere toppen door 0 krijg je ook $0, 0, 1, 0, 1, 1, 1$. Deze code legt de boom uniek vast.

De code die de structuur van de boom voorstelt, wordt best aangevuld met 0-bits tot een veelvoud van acht zodat de structuur kan voorgesteld worden door een exact aantal

bytes. $c_2(T)$ kan dan de reeks van bytes zijn die de symbolen in de bladeren voorstellen als deze diepte eerst wordt overlopen. De volledige boom uit Figuur 1.4 kan dus gecodeerd worden als.

$$00101110|code(p)|code(q)|code(r)|code(s)$$

Hierbij dienden de verticale strepen enkel als visueel hulpmiddel die de scheidingen tussen de bytes aangeven, deze zijn geen deel van de codering van de boom. In totaal wordt de boom in Figuur 1.4 dus voorgesteld door 5 bytes. Als de frequenties van de tekens ook mee doorgestuurd zouden moeten worden, kan er nog een derde deel toegevoegd worden aan de codering. Het derde deel bestaat dan uit een lijst van integers, waarbij de int op plaats i de frequentie van het teken op plaats i in het tweede deel van de codering aangeeft.

2 Verslag

Test en vergelijk de geïmplementeerde algoritmen uitvoerig in je verslag. Test verschillende bestandsgroottes en bedenk *best* en *worst* case scenario's voor de verschillende varianten. Denk na over wanneer je welke variant het best gebruikt. 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 kwadratisch toe terwijl we hier een lineair verband verwacht hadden. ...'

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, 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!

3.2 Input/Output en implementatiedetails

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

Aangezien de varianten wat functionaliteit zullen delen, is het de bedoeling dat je één programma schrijft en aan de hand van een optie `-t` meegeeft welke variant moet uitgevoerd worden. Deze optie moet een waarde tussen 1 en 5 meekrijgen (inclusief de grenzen). Welk getal overeenkomt met welke variant kun je vinden in de sectie Opgave. Als de optie `-c` wordt meegegeven, dan moet je de inhoud die binnenkomt via `stdin` comprimeren. Als de optie `-d` wordt meegegeven, moet het wat binnenkomt uit stdin decomprimeren (volgens de meegegeven variant). Het resultaat moet telkens weggeschreven worden naar `stdout`.

Je programma moet dus bijvoorbeeld als volgt gebruikt kunnen worden:

```
$ comprimeer -t 3 -c < data.txt > compressed
$ comprimeer -t 3 -d < compressed > data.txt
```

of

```
$ echo "DA3" | comprimeer -t 2 -c | comprimeer -t 2 -d
```

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/
|   `-- je broncode
|-- 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 bestand met de naam `sources` te bevatten waar je de dependencies voor het compileren kan aangeven. Dit bestand bevat slechts 1 regel waarop na `huffman:` alle nodige `*.c`-bestanden voor het compileren opgelijst worden. Een voorbeeld hiervan is:

```
huffman: main.c compress.c common.c
```

4.3 Belangrijke data

Tegen *zondag 5 november* 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 1 december* 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 7 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 40.09.110.032[1] of tijdens de oefeningenles de dag voordien.

[1] Eerste verdieping tweede deur links aan de zijde van de auditoria

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.