Politechnika Śląska
Wydział Matematyk Stosowanej
Kierunek Informatyka

Gliwice, 01.10.2020

Programowanie I

projekt zaliczeniowy

"***Kompresja Huffmana***"

Jakub Cibor gr.2 lab. 3

1. Opis projektu.

Krótki zwięzły opis tematyki projektu.

2. Wymagania

-pobieranie ścieżki pliku tekstowego wprowadzonej przez użytkownika

-odczyt znaków z pliku oraz częstotliwości ich występowania

-utworzenie „złożeń" znaków i ułożenie ich w postaci drzewa binarnego

-odczytanie nowego kodowania dla znaków ze „złożenia" przy użyciu rekurencji

-zapis nowego kodowania znaków do pliku tekstowego

3. Przebieg realizacji

Opis algorytmu

Kompresja (kodowanie) Huffmana to algorytm kompresji bezstratnej, zaliczający się do tzw. „greedy algorithms" czyli paradygmatu który tworzy rozwiązanie kawałek po kawałku, przy tym wybierając najbardziej optymalne rozwiązania na daną chwilę.

<u>Kompresja Huffmana w skrócie:</u>

-odczytujemy każdy rodzaju znaku występujący w pliku oraz sortujemy je według ich częstotliwości występowania (od najrzadszych do najczęstszych)

-bierzemy dwa pierwsze znaki i łączymy je, otrzymujemy „złożenie", jego wartość to suma wartości (ilości wystąpień) znaków z których się składa, następnie odkładamy je na odpowiednie miejsce na naszej liście/tablicy znaków.

-bierzemy dwa kolejne pierwsze znaki lub też złożenia, i powtarzamy nasz powszedni proces, robimy to tak długo aż nie zostanie nam jedno złożenie składające się ze wszystkich znaków.

-gdy zostanie nam jedno złożenie możemy użyć go aby odczytać z niego nowe kodowania dla znaków

<u>Szerzej o składaniu:</u>

Ze wszystkich kroków najmniej jasny może wydawać się ten ze składaniem, w końcu jak to działa? Tak naprawdę polega to na ułożeniu drzewa binarnego z elementów. Weźmy za przykład słowo: Huffman

Znak:	*H*	*u*	*f*	*m*	*a*	*n*
Ilość wystąpień:	*1*	*1*	*2*	*1*	*1*	*1*

po sortowaniu otrzymamy: H u m a n f

rozpoczynamy składanie, składamy dwa pierwsze elementy czyli H i u, bardzo istotnym elementem jest rozdzielnie małych i wielkich liter, gdyż ich kodowania ASCII różnią się.

Tak więc składamy H i u, pierwsze złożenie oznaczmy jako n1

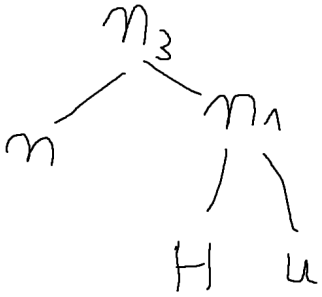
nasza lista po pierwszym złożeniu: m a n n1 f

powtarzamy proces:

nasza lista po drugim złożeniu n n1 n2 f

po trzecim: n2 f n3

w tym momencie warto się zatrzymać, gdyż można się zastanowić czym jest n3, otóż jest to złożenie litery n oraz złożenia n1, w przybliżeniu wygląda tak:



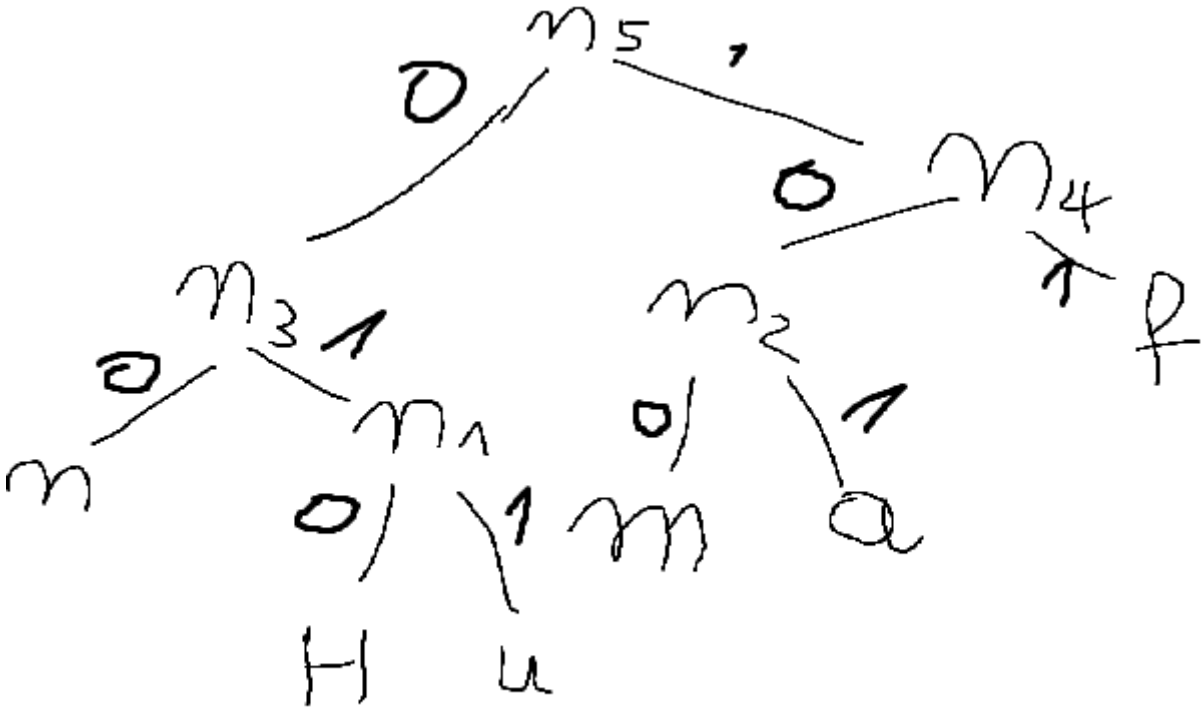
warto zaznaczyć że pierwsze elementy z listy układane są na lewą stronę, a późniejsze na prawą

znowu składamy, po czwartym złożeniu: n3 n4

i po piątym i w naszym przypadku ostatnim n5

Odczyt nowego kodowania:

ten proces najprościej będzie opisać na podstawie tego co otrzymamy na poprzednim przykładzie:



jest to przybliżony wygląd takiego złożenia, ale jak z tego odczytać nowe kodowania, otóż zaczynamy na wierzchu tego złożenia i poruszamy się po połączeniach aż nie dojdziemy do znaku, przejście w lewo dodaje do kodowania 0 a w prawo 1.

tak więc dla f kodowanie to: 11

dla a: 101

dla u: 011

Można się teraz zastanawiać a skąd wiadomo że jakiś ciąg binarny jest kodowaniem dla znaku a nie częścią kodowania innego? Otóż sposób w jaki otrzymujemy kodowanie wyklucza taką możliwość, pełne kodowanie danego znaku nigdy nie będzie częścią innego kodowania i tu dochodzimy do problemów tego algorytmu.

Problemy algorytmu:

Ze względu na niepowtarzalność kodowania znaków, możemy uzyskać kodowanie znaków które jest dłuższe niż ASCII, co powoduje że zapis tych znaków będzie wymagał większej ilości pamięci, jednak będzie to dotyczyło tylko najrzadziej występujących znaków. Jednak to pozostawia nas z oczywistym wnioskiem: kompresja Huffmana jest kompletnie nieefektywna przy zapisie danych z bardzo wyrównaną częstotliwością występowania znaków. Ponadto unikalne kodowanie dla różnych danych powoduje konieczność zapisu kodowania przy pomocy którego skompresowaliśmy plik, tak abyśmy byli go w stanie później zdekompresować. Powoduje to nieefektywność algorytmu dla bardzo małych plików.

Przebieg tworzenia programu:

Tworzenie rozpocząłem od stworzenia części która pobiera dane z pliku oraz sortuje je według częstotliwości występowania, ta część jest prosta i myślę że nie ma sensu jej omawiać.

Kolejną częścią było stworzenie części programu odpowiedzialnej za składanie . Była to najbardziej problematyczna część programu, wymyślenie metody która by się tym dobrze zajmowała zajęła mi sporo czasu, a próby które z początku wydawały się być udane, powodowały problemy przy pisaniu kolejnych części programu. Po wielu próbach oraz całkowitym przepisaniu programu uzyskałem coś takiego:

```
while (*ln > 1)
	{
		node n;
		n.combineNodes(nodeArray[i], nodeArray[i + 1]);
		last_in_array++;
		if (nodeArray[last_in_array - 1].value > n.value)
		{
			int j;
			for (j = i+1; j <= last_in_array-1; j++)
			{
				if (nodeArray[j].value > n.value)
					break;
			}
```

```
                for (int k = last_in_array - 1; k >= j; k--)
                {
                        nodeArray[k + 1] = nodeArray[k];
                }
                nodeArray[j] = n;
        }
        else
        {
                nodeArray[last_in_array] = n;
        }



        i += 2;
        *ln -= 1;
}
```

Powyższy kod działa w następujący sposób:

Dopóki zwartość int wskazywana przez zmienną ln jest większa niż 1 bierze 2 pierwsze elementy i je łączy.

Następnie zwiększana jest zmienna last_in_array która wskazuje na jakiej pozycji w tablicy jest ostatni element, później sprawdzamy czy ostatni element w tablicy jest większy niż nowe złożenie, jeśli jest to:

szukamy pierwszego elementu który jest takiej samej wielkości lub większy niż nowe złożenie, kiedy go znajdziemy to ten element i wszystkie kolejne przenosimy na pozycje dalej, a na miejsce w którym był ten element wpisujemy nowe złożenie.

W przeciwnym wypadku na ostatnie miejsce wpisujemy nowe złożenie.

Zwiększamy iterator o 2 oraz odejmujemy 1 od *ln (jako że 2 elementy zamieniliśmy na 1)

Dalsze części programu to szukanie nowego kodowania dla znaków poprzez rekurencje oraz zapis do pliku.

4. Instrukcja użytkownika

Przygotować plik tekstowy dla którego chcielibyśmy znaleźć nowe kodowanie, następnie uruchomić program i podać ścieżkę do pliku. Program zwróci plik z nowym kodowaniem dla znaków (kodowanie.txt) w folderze wyżej niż ten w którym znajduje się program.

5. Podsumowanie i wnioski.

Udało mi się wykonać algorytm kompresji Huffmana, do pełnej kompresji brakuje tylko zapisu pliku źródłowego w nowej postaci. Największym problemem było napisanie odpowiedniego kodu aby otrzymywać poprawne złożenia. Było z nimi wiele problemu, miedzy-innymi: złożenia w nieodpowiedniej kolejności, utrata wartości na które wskazywały wskaźniki, czy też problemy z przechowywanymi znakami. Naturalnym kolejnym krokiem w rozwoju programu wydaje się być dodanie funkcjonalności samej kompresji, tak aby program wykorzystywał otrzymane kodowanie, ponadto można przetestować i ewentualnie wprowadzić wsparcie dla polskich znaków.