Aufgabe 1: Schmucknachrichten

Teilnahme-ID: 73795

Bearbeiter/-in dieser Aufgabe:
Timon Retzlaff

24. April 2025

Inhaltsverzeichnis

Lösungsidee

Erste Ansätze

Zuerst habe ich Huffman-Codierung implementiert, die aber für verschiedene Zeichenkosten nicht optimal war.
Danach habe ich den Algorithmus aus diesem Paper [1] umgesetzt. Dieser war zwar optimal, aber zu langsam, trotz der quasi polynomiellen Laufzeit $O(n^{C+2})$, wobei n die Anzahl der zu codierenden Zeichen (Z) ist und C die höchsten Kosten eines Codezeichens.

Endgültige Lösung

Ich habe mir einen Greedy-Algorithmus mit $O(r^3n^5 \cdot log(n))$ ausgedacht und umgesetzt. Dabei ist r die Anzahl der Codezeichen und n die Anzahl von Z. Der Algorithmus findet zwar nicht die optimalen, aber brauchbare Lösungen.

$O(r^3n^5 \cdot log(n))$ ist für große Beispiele generell besser als $O(n^{C+2})$. Bei Schmuck 9 zum Beispiel ist $C = 4$. Dadurch potenziert man n mit 6 und nicht 5. Dies hat einen großen Einfluss, da n = 674 ist. Im Gegensatz dazu ist $r = 4$ und somit $r^3 = 4^3 = 64$, wobei $log(n) = 2,8$ ist. Dadurch ist mein Algorithmus um beinahe Faktor 4 schneller. Zudem ist der konstante Faktor deutlich kleiner.

Voraussetzungen

Jeder präfixfreie Codezeichensatz kann durch einen Baum dargestellt werden, in dem die Blätter die einzelnen Z sind und die Kanten jeweils das Codezeichen darstellen. Um den Code eines Blattes zu konstruieren, startet man bei der Wurzel und geht den Weg zu diesem. Währenddessen merkt man sich die Kanten über die man geht und hängt die korrespondierenden Codezeichen aneinander. Dies wird auch in oben genanntem Paper [1] beschrieben.
In jenem wird ebenfalls gezeigt, dass man jeden optimalen Code auch durch einen vollen Baum darstellen kann, bei dem alle Z durch die n höchsten Blätter dargestellt werden, während die restlichen ignoriert werden.
Dabei ist das höchste Blatt das, welches am nächsten an der Wurzel ist und die geringste Tiefe hat. Die Z werden so zugeteilt, dass das Z mit der höchsten Wahrscheinlichkeit dem höchsten Blatt entspricht und das mit der zweithöchsten Wahrscheinlichkeit, dem zweithöchsten usw.
Die Kosten eines Baumes sind die Summe der Produkte von Wahrscheinlichkeit jedes Z multipliziert mit der Tiefe des zugeordneten Blattes.
Der optimale volle Baum (Baum mit geringsten Kosten) kann maximal $n(r-1)$ Blätter haben, wobei r die Anzahl der Codezeichen ist (siehe [1]).
Bei einem vollen Baum kann man die Kinder von einem Knoten an einen anderen Knoten, welcher ein Blatt ist, transferieren, ohne dabei die Anzahl der Blätter zu verändern, da dabei der erste Knoten zu einem Blatt wird und der zweite Knoten zu einem inneren Knoten. Dies nenne ich ab jetzt Subbaum-Migration.

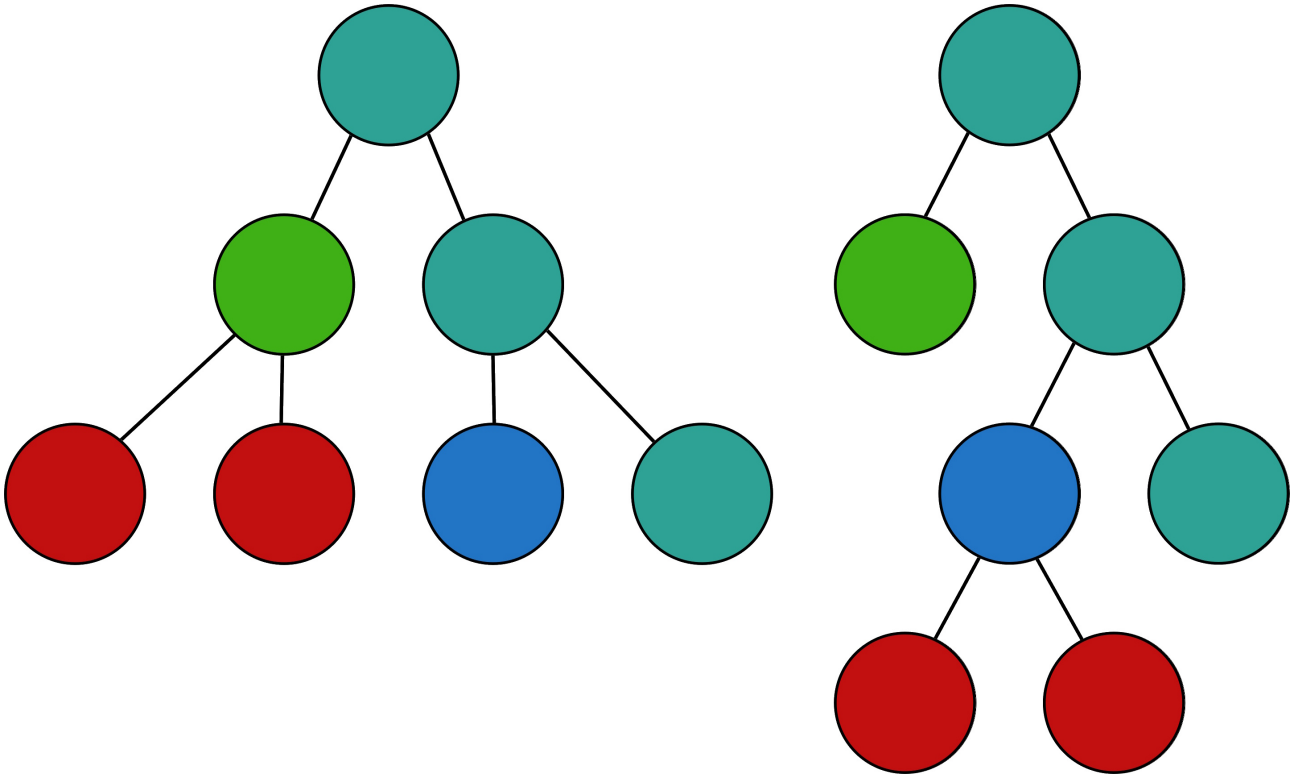

Schaubild 1: Beispiel der Subbaum-Migration

Algorithmus

Man generiert einen vollen Baum mit *n* Blättern. Auf diesem iteriert man über alle inneren Knoten. Der Subbaum von jedem dieser Knoten kann an jedes Blatt des Baums migriert werden, welches nicht in diesem Subbaum enthalten ist. Dabei macht es aber keinen Unterschied welches Blatt man nimmt, solange sie auf der gleichen Tiefe sind. Daher muss man für jede Tiefe nur ein Blatt betrachten.

Für jede unterschiedliche mögliche Migration rechnet man die Kosten des entstehenden Baumes aus, während man sich die geringsten merkt.

Falls die Kosten des Baumes nach der besten Subbaum-Migration geringer sind als davor, führt man sie aus und wiederholt den Vorgang.

Sonst merkt man sich den Baum und wiederholt das ganze mit *n+1* Blättern, danach *n+2*, … bis zu *n(r-1)*. Dabei merkt man sich den Baum mit den niedrigsten Kosten.

Diesen nimmt man letztendlich.

In meiner tatsächlichen Implementierung habe ich die obere Grenze der Blätteranzahl auf *n+3r* begrenzt, um die Performance zu verbessern. Diesen Wert habe ich empirisch ermittelt, wobei der optimale Baum immer weniger als *n+2r* Blätter hatte. Das dritte *r* habe ich als Puffer eingebaut.

Laufzeitanalyse

Es gibt in einem optimalen vollen Baum höchstens $n-1$ innere Knoten und höchstens $n(r-1)$ Blätter (siehe [1]).

Um die Kosten eines Baums zu ermitteln, muss man alle Blätter durchgehen, um die n höchsten zu finden. Da es maximal $n(r-1) = rn-n \leq rn$ Blätter gibt, ist die Laufzeit davon $O(rn \cdot log(n))$.

Diese Kosten werden für höchstens jedes Paar von innerem Knoten und Blatt berechnet.

Daher gibt es höchstens n(r-1)$(n-1) = (rn-n)(n-1) = rn^2-rn-n^2+n \leq rn^2$ Subbaum-Migrationen. Ein Optimierungsschritt hat daher $O(rn \cdot log(n) \cdot (rn^2)) = O(r^2n^3 \cdot log(n))$.

Man kann bei den Migrationen maximal von dem einen Extrem, dass alle Knoten immer am x.-kürzesten Kind eines Knoten sind, zu einem anderen Extrem dieser Art gehen. Dafür benötigt man höchstens so viele Schritte, wie es innere Knoten gibt, also maximal $n-1 < n$. Diese Annahme ist eine extreme Überschätzung der realen Laufzeit, da dieser Worst-Case ziemlich einfach bei der Baum-Generierung vermieden werden kann, indem man zum Beispiel immer das höchste Blatt erweitert.

Das lokale Minimum mit einer festen Anzahl Blättern zu erreichen hat also eine Laufzeit von $O(n(r^2n^3 \cdot log(n))) = O(r^2n^4 \cdot log(n))$.

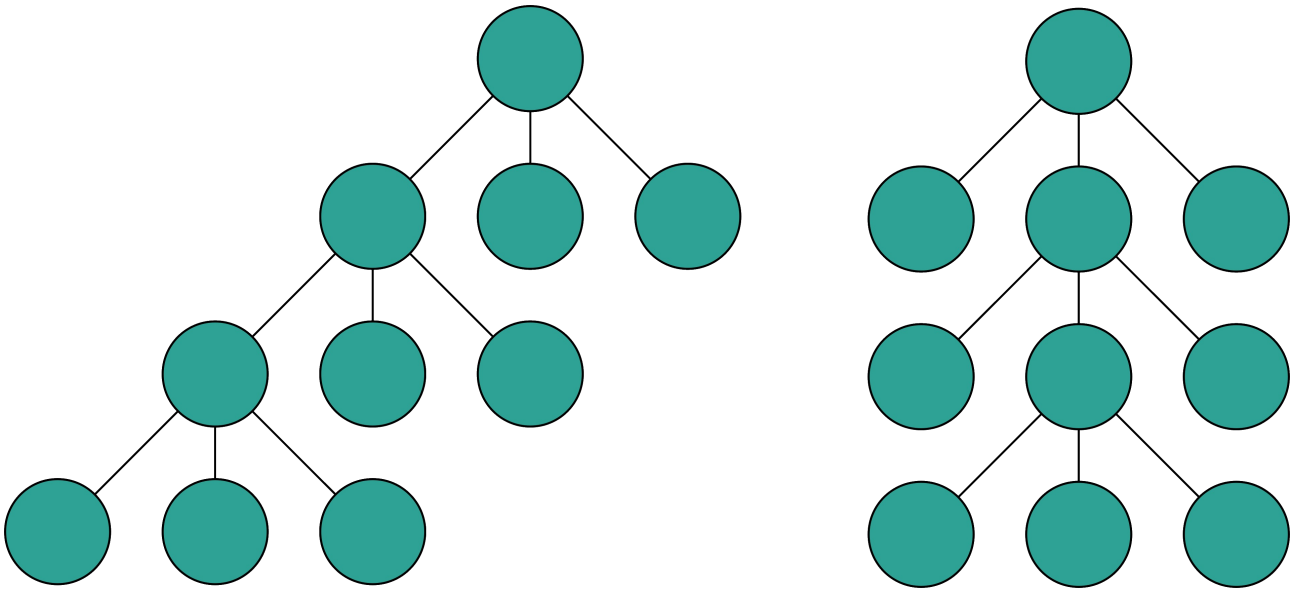

Schaubild 2: Beispiel von Extremen bei der Migration

Der Algorithmus geht maximal alle Blätteranzahlen von n bis $n(r-1)$ durch. Daher muss man maximal $n(r-1)-n+1 = rn-n-n+1 = rn-2n+1 < rn$ Bäume durchgehen.

Den besten Baum finden hat also eine Laufzeit von $O(rn(r^2n^4 \cdot log(n))) = O(r^3n^5 \cdot log(n))$.

Man könnte die Laufzeit noch verbessern, indem man ein *int*-Array verwendet, in dem die Anzahl der Blätter in jeder Tiefe gespeichert wird. Damit könnte man die Kosten in $O(n)$ ausrechnen, aber die Subbaum-Migration würde $O(rn)$ benötigen.

Dadurch wäre die insgesamte Laufzeit nur noch $O(r^3n^5)$.

Umsetzung

Übersicht

Es gibt 5 Klassen: *Main*, *Encoder*, *Tree*, *Node* und *Decoder*. Die *Main*-Klasse kümmert sich um das Setup und die Aufrufe der Methoden. Die *Encoder*-Klasse bereitet das Optimieren vor und generiert die Bäume. Die *Tree*-Klasse repräsentiert die Bäume und enthält die Methoden, um einen Baum zu optimieren. Die *Node*-Klasse repräsentiert die Knoten der Bäume und enthält Verweise auf Eltern und Kinder der Knoten, sowie Methoden zum rekursiven Zuweisen der Tiefe und der Codes.

Details

Main-Klasse

In der *Main-Klasse* lese ich die Input-Datei ein und sortiere die Kosten. Dann rufe ich die *generateTable*-Methode der *Encoder-Klasse* auf. Danach codiere ich die Nachricht mithilfe der Tabelle und decodiere sie daraufhin mithilfe der *decode*-Methode der *Decoder*-Klasse.

Encoder-Klasse

In der *generateTable*-Methode werden zuerst die Zeichen gezählt und die Wahrscheinlichkeiten ausgerechnet. Dann werden sie sortiert und die *getBest*-Methode wird aufgerufen. Diese findet den besten Baum.

Dafür erstelle ich zuerst einen Baum, der nur die Wurzel enthält. Dann bereite ich zwei Variablen vor, um den besten Baum und seine Kosten zu speichern. Außerdem berechne ich die maximale Blattanzahl. Abweichend von dem eigentlichen Algorithmus nehme ich dafür das Minimum von $n+3r$ und $n(r-1)$, um das Programm zu beschleunigen.

Es gibt dann eine Schleife, die läuft bis die maximale Blätteranzahl überschritten wurde. In dieser wird der Baum geklont und durch die *optimize*-Methode der *Tree*-Klasse optimiert.

Dann wird er gespeichert, wenn seine Kosten besser sind.

Danach wird der Baum, der als Start benutzt wird, erweitert durch die *expand*-Methode, um einen Baum mit mehr Blättern zu erreichen. In dieser wird das höchste Blatt zu einem internen Knoten.

Tree-Klasse

In der *optimize*-Methode der *Tree*-Klasse wird *optimizationStep* wiederholt ausgeführt, bis es keine Verbesserungen mehr gibt.

Jeder *optimizationStep* besteht aus zwei Methodenaufrufen. Der erste ist *getBestAction* und der zweite *applyAction*.

GetBestAction geht alle Knoten durch, die in einer Liste gespeichert sind und iteriert für jeden über alle Blätter, die ebenfalls in einer Liste gespeichert sind. Dadurch findet die Methode die beste Migration. Abweichend von dem Algorithmus sortiere ich beim Berechnen der Kosten jedoch alle Blätter und brauche deswegen dafür $O(rn \cdot log(rn))$.

ApplyAction führt diese Migration durch.

Beispiele

Schmuck 0

n=12 r=2

Max steps: 2 avg steps: 2.0

Best tree leaf count: 12/24 with cost: 3.424242424242424

: 101, E: 110, I: 111, N: 100, S: 000, R: 0110, D: 0011, L: 0101, M: 0010, O: 0111, C: 01000, H: 01001

Encoded in: 113 pearls (42.803032%) = 113mm

Decoded as: DIE SONNE SOLL DIR IMMER SCHEINEN

Schmuck 00

n=28 r=3

Max steps: 5 avg steps: 4.4

Best tree leaf count: 37/56 with cost: 2.6382978723404253

: 11, e: 00, i: 02, t: 12, n: 10, h: 212, s: 222, c: 220, l: 200, a: 210, r: 201, D: 202, d: 211, u: 012, G: 0100, g: 0112, m: 0111, w: 0101, A: 22110, E: 22122, P: 22100, W: 22120, Z: 0102, b: 0110, f: 22101, k: 22102, o: 22121, ?: 22112

Encoded in: 372 pearls (32.97872%) = 372mm

Decoded as: Es war mal ein kleines Gedicht Das machte ein finstres Gesicht Die Welt wird nicht heilen Durch lustige Zeilen Pointe? Auch die gibt es nicht

Schmuck 01

n=45 r=5

Max steps: 6 avg steps: 4.75

Best tree leaf count: 45/90 with cost: 2.0318021201413425

: 1, e: 2, n: 00, r: 43, i: 42, s: 31, a: 40, l: 30, t: 44, h: 34, c: 41, g: 32, o: 33, m: 030, u: 034, .: 011, d: 041, k: 032, E: 044, ,: 022, D: 012, b: 020, w: 023, ü: 031, I: 010, S: 014, f: 013, p: 024, v: 042, F: 043, O: 0212, V: 0401, ö: 0332, z: 0331, B: 0214, G: 0404, ä: 0210, H: 0211, K: 0333, M: 0403, N: 0330, R: 0334, W: 0400, ß: 0402, …: 0213

Encoded in: 1150 pearls (25.397528%) = 1150mm

Decoded as: Die Bäume im Ofen lodern. Die Vögel locken am Grill. Die Sonnenschirme vermodern. Im übrigen ist es still. Es stecken die Spargel aus Dosen Die zarten Köpfchen hervor. Bunt ranken sich künstliche Rosen In Faschingsgirlanden empor. Ein Etwas, wie Glockenklingen, Den Oberkellner bewegt, Mir tausend Eier zu bringen, Von Osterstören gelegt. Ein süßer Duft von Havanna Verweht in ringelnder Spur. Ich fühle an meiner Susanna Erwachende

neue Natur. Es lohnt sich manchmal, zu lieben, Was kommt, nicht ist oder war. Ein Frühlingsgedicht, geschrieben Im kältesten Februar…

Schmuck 1

n=25 r=3

Max steps: 4 avg steps: 3.0

Best tree leaf count: 33/50 with cost: 3.4107142857142865

e: 2, t: 101, : 010, i: 011, n: 100, r: 110, w: 111, B: 012, I: 0011, b: 021, ": 0000, f: 112, s: 102, D: 0001, F: 022, N: 121, W: 020, a: 122, d: 1201, h: 1200, k: 0002, m: 0012, o: 0020, u: 0021, ü: 0010

Encoded in: 162 pearls (36.160713%) = 191mm

Decoded as: BWINF steht für "Die Bundesweiten Informatikwettbewerbe"

Schmuck 2

n=9 r=2

Max steps: 3 avg steps: 3.0

Best tree leaf count: 9/18 with cost: 3.292682926829268

a: 0, : 111, b: 1000001, c: 100001, d: 10001, e: 1001, f: 110, g: 101, h: 1000000

Encoded in: 71 pearls (21.646341%) = 135mm

Decoded as: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bcdefgh

Schmuck 3

n=9 r=3

Max steps: 5 avg steps: 3.8

Best tree leaf count: 9/18 with cost: 2.536363636363636

a: 1, b: 01, c: 00, : 20, d: 020, e: 21, f: 022, g: 021, h: 22

Encoded in: 189 pearls (21.477274%) = 279mm

Decoded as: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccccccccccccccc defgh

Schmuck 4

n=14 r=2

Max steps: 4 avg steps: 4.0

Best tree leaf count: 14/28 with cost: 9.785714285714286

a: 0000000, b: 1001, c: 0101, d: 0011, e: 011, f: 0000001, g: 101, h: 11, i: 000001, j: 00001, k: 0010, l: 0001, m: 1000, n: 0100

Encoded in: 61 pearls (54.464287%) = 137mm

Decoded as: abcdefghijklmn

Schmuck 5

n=41 r=7

Max steps: 4 avg steps: 3.5

Best tree leaf count: 61/82 with cost: 3.1245059288537536

 : 10, e: 11, t: 3, i: 001, o: 20, s: 010, a: 12, n: 000, r: 21, c: 02, d: 011, l: 22, m: 13, h: 4, u: 012, f: 002, p: 03, b: 003, y: 14, g: 23, .: 04, ,: 50, q: 51, v: 013, x: 05, w: 15, F: 014, T: 004, A: 61, G: 240, H: 06, S: 016, ’: 25, j: 52, -: 005, 1: 62, 2: 16, 5: 242, 9: 015, z: 60, ;: 241

Encoded in: 2226 pearls (27.49506%) = 3162mm

Decoded as: Summary This paper gives a method for constructing minimum-redundancy prefix codes for the general discrete noiseless channel without constraints. The costs of code letters need not be equal, and the symbols encoded are not assumed to be equally probable. A solution had previously been given by Huffman in 1952 for the special case in which all code letters are of equal cost. The present development is algebraic. First, structure functions are defined, in terms of which necessary and sufficient conditions for the existence of prefix codes may be stated. From these conditions, linear inequalities are derived which may be used to characterize prefix codes. Gomory’s integer programming algorithm is then used to construct optimum codes subject to these inequalities; an elegant combinatorial approach to obtain a strictly computational experience is presented to demonstrate the practicability of the method. Finally, some additional coding problems are discussed and a problem of classification is treated.

Schmuck 6

n=34 r=3

Max steps: 4 avg steps: 3.0

Best tree leaf count: 43/68 with cost: 5.849999999999997

，: 0000, 。: 110, 文: 0100, 有: 101, 馬: 1000, 、: 0021, 時: 121, 雄: 0201, 遇: 1001, 晉: 211, 齊: 0102, 希: 022, 不: 0012, 志: 22, 望: 201, 非: 0002, 也: 021, 無: 0020, 止: 0011, 督: 0200, 古: 210, 武: 111, 大: 0101, 鄧: 120, 未: 0110, 公: 102, 肯: 012, 英: 0010, 郵: 0001, 學: 200, 禹: 112, 去: 1002, 妻: 0111, 都: 202

Encoded in: 141 pearls (44.0625%) = 234mm

Decoded as: 古英雄未遇時，都無大志，非止鄧禹希文學、馬武望督郵也。晉文公有妻有馬，不肯去齊。

Schmuck 7

n=82 r=10

Max steps: 6 avg steps: 5.25

Best tree leaf count: 109/164 with cost: 1.629457852480655

 : 3, e: 2, n: 0, i: 1, r: 41, s: 44, a: 60, d: 55, h: 54, t: 40, u: 62, l: 61, c: 46, g: 65, m: 64, o: 43, b: 45, ,: 63, f: 42, w: 52, .: 53, k: 56, z: 50, S: 76, ä: 660, ü: 57, ö: 74, v: 47, F: 71, p: 663, ß: 75, D: 664, E: 73, V: 72, W: 513, N: 67, K: 511, B: 516, A: 512, M: 661, H: 70, I: 515, L: 665, R: 514, G: 662, T: 666, Z: 667, j: 80, J: 5101, -: 84, ;: 5100, (: 83,): 5106, ::: 5102, _: 68, O: 82, U: 81, [: 85,]: 5105, “: 58, „: 517, P: 5103, 1: 86, 4: 5104, 3: 48, ?: 77, !: 668, 2: 95, 5: 94, 6: 69, 0: 518, 8: 78, Ä: 87, q: 90, ’: 5107, 7: 93, 9: 59, Q: 49, C: 91, Ö: 79, x: 96, y: 92

Encoded in: 130534 pearls (19.758957%) = 134559mm

Decoded as: Aus dem Leben unserer Vögel. Lektion 1. Bekannte Vögel. Ich möchte wissen, wie viele Vögel ihr dem Aussehen nach kennt, und was ihr über ihre Nester und ihre Lebensweise wißt. Es gibt zwischen drei- bis vierhundert verschiedene deutsche Vögel, und sehr wenige Leute kennen sie alle. Aber die an irgend einem Orte gewöhnlich vorkommenden Vögel kann man in einem Jahre ohne große Mühe sämtlich kennen lernen. Später mag man sich dann nach denen umsehen, die seltener vorkommen. Am besten wird so angefangen, daß man alle Vögel aufschreibt, die man sicher kennt, und angibt, woran man sie erkennt….

Schmuck 8

n=321 r=5

Max steps: 14 avg steps: 13.75

Best tree leaf count: 333/642 with cost: 5.206951026856272

，: 21, 。: 30, 「: 0011, 」: 1101, 》: 1011, 《: 0000, ： : 311, 格: 0001, 之: 0111, 不: 1110, 云: 0211, 為: 221, 有: 320, 一: 0113, 時: 01000, 其: 1300, 詩: 1311, 風: 1030, 人: 1130, 相: 0131, 武: 1003, 是: 0112, 也: 2010, 誰: 00100, ； : 10001, 在: 0013, 公: 1210, 知: 220, 言: 11001, 、: 0310, 者: 10011, 而: 0002, 性: 0102, 百: 11111, 皆: 411, 骨: 0120, 花: 1201, 嚴: 3100, 調: 0201, 情: 2000, 以: 203, 都: 10100, 意: 1310, 天: 0210, 未: 2001, 十: 1002, 作: 202, 律: 231, 馬: 10010, 日: 0311, 七: 0040, 將: 1301, 與: 1021, 三: 1121, 上: 0031, 同: 01100, 通: 11000, 君: 401, 吟: 01011, ：0301, 尤: 1113, 中: 01010, 水: 142, 才: 0020, 乎: 230, 繩: 0021, 年: 0200, 談: 2011, 四: 1012, 目: 1102, 曰: 1200, 滿: 1120, 看: 330, 外: 11110, 多: 10101, 無: 1031, 大: 1112, 光: 0121, 李: 313, 問: 312, 何: 331, 非: 0012, 此: 1131, 好: 0033, 來: 1320, 文: 0104, 開: 1400, 見: 0203, 王: 3103, 後: 1103, 林: 1132, 得: 1023, 枝: 0130, 趣: 00101, 禹: 1013, 便: 0300, 篇: 01101, 韓: 10000, 志: 0030, 濟: 3101, 解: 1020, 自: 400, 深: 410, 已: 0003, 出: 1211, 老: 322, 封: 10104, 蘄: 1413, 清: 0204, 分: 0233, 〈: 1331, 須: 422, 專: 01104, 訊: 1203, 逋: 3213, 頌: 1220, 希: 0332, 騎: 01014, 小: 1313, 踏: 1332, 成: 1033, 丑: 403, 怒: 1042, 樓: 01103, 易: 414, 世: 0302, 鐘: 0400, 思: 0323, 丞: 01030, 星: 1330, 訟: 044, 氣: 0042, 春: 01012, 否: 1122, 萬: 333, 般: 1032, 席: 01003, 種: 042, 到: 11004, 許: 0224, 耶: 1133, 樹: 01013, 渾: 0230, 居: 1221, 遇: 413, 則: 143, 屈: 010011, 扈: 1123, 豈: 3102, 晉: 0022, 葉: 134, 桐: 412, 蹕: 10014, 托: 0221, 遙: 1303, 九: 1312, 詞: 1141, 詠: 2012, 象: 10012, 婦: 0320, 恩: 1212, 火: 0032, 陰: 1143, 山: 1222, 色: 1204, 空: 0231, 登: 1411, 命: 0330, 陽: 0114, 皇: 2013, 炊: 10102, 事: 0123, 皋: 043, 和: 0303, 二: 340, 底: 1401, 增: 0333, 誠: 2014, 貧: 01002, 暮: 0041, 肯: 0412, 窺: 334, 押: 1134, 況: 1213, 雄: 424, 雅: 1224, 哉: 0213, 今: 10103, 郎: 2004, 仗: 0331, 雙: 11112, 拙: 10013, 仙: 0024, 招: 1202, 勞: 224, 若: 0122, 士: 0322, 飯: 00103, 英: 11114, 曲: 1322, 拳: 44, 郵: 223, 盼: 421, 能: 1412, 最: 0023, 省: 233, 謂: 2003, 鄂: 1304, 霄: 402, 指: 0202, 鄉: 2002, 國: 341, 圍: 0321, 輕: 1014, 化: 0411, 餘: 01031, 望: 1302, 愛: 0410, 生: 11002, 鄧: 1004, 攬: 124, 甲: 0004, 張: 0312, 猶: 204, 然: 430, 似: 10002, 元: 1104, 奇: 232, 午: 314, 先: 1230, 靈: 0141, 草: 0220, 齊: 222, 楊: 1022, 半: 0313, 齋: 10003, 歌: 11113, 低: 3211, 子: 11003, 卒: 1321, 卓: 0401, 腔: 0133, 絕: 0014, 字: 0140, 煙: 323, 祠: 1140, 止: 0212, 督: 0334, 散: 01034, 入: 1402, 略: 342, 兩: 3212, 浪: 3210, 酬: 0132, 奮: 1124, 異: 0413, 絲: 1044, 蝴: 343, 蝶: 0403, 歸: 0223, 學: 0234, 西: 10004, 門: 431, 如: 0404, 率: 0414, 龍: 1414, 箏: 1142, 憐: 0214, 經: 1314, 首: 0222, 料: 0034, 妙: 0143, 定: 1214, 辛: 1223, 斜: 433, 從: 1333, 熟: 1233, 客: 0304, 辦: 1024, 玩: 1324, 莫: 423, 榮: 01102, 侮: 242, 侯: 0124, 冰: 010012, 家: 1043, 架: 243, 去: 1114, 妻: 01033, 於: 01032, 秀: 0043, 心: 234, 範: 332, 寄: 0134, 毆: 00104, 及: 1231, 里: 1034, 城: 324, 野: 0232, 描: 1323, 金: 432, 近: 1403, 租: 0314, 叟: 0142, 鏡: 144, 口: 0402, 古: 00102, 迥: 010010, 句: 241, 寧: 01004, 臨: 024, 寫: 420, 可: 1041, 華: 240, 翰: 010013, 己: 1040, 柳: 1410, 至: 1232, 寶: 3104, 懷: 034, 韻: 404

Encoded in: 2392 pearls (47.235386%) = 3296mm

Decoded as: 古英雄未遇時，都無大志，非止鄧禹希文學、馬武望督郵也。晉文公有妻有馬，不肯去齊。光武貧時，與李通訟逋租於嚴尤。尤奇而目之。光武歸謂李通曰：「嚴公寧目君耶」窺其意，以得嚴君一盼為榮。韓蘄王為小卒時，相士言其日後封王。韓大怒，以為侮己，奮拳毆之。都是一般見解。鄂西林相公《辛丑元日》云：「攬鏡人將老，開門草未生。」《詠懷》云：「看來四十猶如此，便到百年已可知。」皆作郎中時詩也。玩其詞，若不料此後之出將入相者。及其為七省經略，《在金中丞席上》云：「問心都是酬恩客，屈指誰為濟世才」《登甲秀樓》絕句云：「炊煙卓午散輕絲，十萬人家飯熟時。問訊何年招濟火，斜陽滿樹武鄉祠。」居然以武侯自命，皆與未得志時氣象迥異。張桐城相公則自翰林至作首相，詩皆一格。最清妙者：「柳陰春水曲，花外暮山多。」「葉底花開人不見，一雙蝴蝶已先知。」「臨水種花知有意，一枝化作兩枝看。」《扈蹕》云：「誰憐七十龍鐘叟，騎馬踏冰星滿天」《和皇上〈風箏〉》云：「九霄日近增華色，四野風多仗寶繩。」押「繩」字韻，寄托遙深。 二 楊誠齋曰：「從來天分低拙之人，好談格調，而不解風趣。何也格調是空架子，有腔口易描；風趣專寫性靈，非天才不辦。」餘深愛其言。須知有性情，便有格律；格律不在性情外。《三百篇》半是勞人思婦率意言情之事；誰為之格，誰為之律而今之談格調者，能出其範圍否況皋、禹之歌，不同乎《三百篇》；《國風》之格，不同乎《雅》、《頌》：格豈有一定哉許渾云：「吟詩好似成仙骨，骨里無詩莫浪吟。」詩在骨不在格也。

Schmuck 9

n=674 r=4

Max steps: 54 avg steps: 51.25

Best tree leaf count: 685/1348 with cost: 7.998907581385136

の: 011, た: 21, に: 30, い: 020, し: 01000, と: 00100, を: 1001, て: 00001, は: 10000, 、: 00010, な: 0002, 。: 000000, が: 021, る: 012, こ: 003, で: 030, か: 31, っ: 22, う: 10100, ら: 00200, れ: 01001, す: 10001, り: 00002, ー: 20000, き: 00101, ま: 11000, そ: 00110, ン: 01010, く: 230, だ: 10110, よ: 0103, あ: 11001, さ: 00030, も: 10002, わ: 01002, : 320, : 01020, ・: 000002, お: 00102, 「: 00120, 」: 00210, け: 023, ル: 20100, え: 10101, ち: 11100, ん: 0022, つ: 00003, ス: 0013, ラ: 12000, イ: 12010, カ: 200100, ト: 002010, 見: 231, 人: 11020, ウ: 11101, レ: 00013, ど: 01012, や: 10111, ジ: 000003, 聖: 00103, 大: 20002, 彼: 20011, 者: 10300, め: 03100, エ: 0001100, 上: 330, 思: 00121, せ: 10210, ア: 01300, フ: 10030, 出: 10120, 堂: 12100, ば: 00000100, サ: 001110, 車: 0023, 建: 20200, 道: 102000, み: 11200, む: 01021, タ: 00211, チ: 12001, ョ: 13000, 海: 10201, 言: 02200, 中: 11011, 手: 321, 物: 11110, 事: 00202, げ: 11102, べ: 12200, ク: 12110, 一: 20210, 会: 01013, 地: 110020, 口: 11201, 通: 11300, 水: 20102, ね: 000310, び: 10211, ほ: 11012, ょ: 13001, ろ: 01310, オ: 10103, マ: 0001101, 部: 00203, 入: 11030, 主: 11111, 乗: 010030, 立: 20120, 雨: 00000110, 任: 03101, 築: 00032, 司: 11021, 合: 10031, —: 100030, 気: 11120, 場: 10121, ぶ: 10220, 決: 20300, ァ: 20003, ド: 10202, 員: 03200, 祭: 200200, 馬: 13010, 川: 12020, 自: 10022, 分: 0001200, 家: 02201, 送: 0001110, 行: 02210, ぎ: 0101100, ご: 10310, ざ: 133, ッ: 12011, 奏: 20021, 内: 2000100, 駅: 12101, 重: 101020, 線: 12002, 市: 00212, 不: 11210, 明: 20030, 利: 13100, 港: 201100, 前: 1101000, 降: 03110, 目: 10130, 後: 00000101, 友: 100210, 取: 20111, 屋: 00000102, 運: 13101, ….

Encoded in: 20908 pearls (57.10072%) = 36611mm

Decoded as: 英国陸地測量部制作の地図ではカラン・ウォーフ、地元の人には単にカランと呼ばれている場所は、今でこそ海岸線から二マイルほど内陸部にあるが、かつてはもっと海寄りで、無敵艦隊との戦いに六隻の船を送り、その一世紀後にはオランダの攻撃を迎え撃つため四隻の船を送り出した由緒ある港として歴史に輝かしい名を残している。ところがやがてカル川の河口域は沈泥でふさがって港口には砂州ができ、海上貿易の船は他に港を探さざるを得なくなった。その後、カル川の流れはやせ細り、それまでのようにあちらこちらへ縦横に伸びるかわりに身を縮めておとなしい河川に変貌し、しかも河川としても決して大きいほうの部類ではなかった。市民たちは港で生計が立てられないことを見て取ると、塩沢を埋め立てることでなにがしかの代償が得られるかも知れないと考え、海水を防ぐために石の堤防を築き、その真ん中にカル川の流れを海に放出する水路を造った。こうしてカラン・フラットと呼ばれる低地の牧草地ができあがり、自由市民はここで羊を放牧する権利を持ち、海峡のむこう、フランスのプレサレ羊にも負けない美味なマトンを生産するようになった。….

Quellcode

```
public class Encoder {

   private static Tree getBest(final Double[] probabilities, final int[] costs) {
      Tree tree = new Tree(costs);
      Tree best = null;
      double bestCost = Double.MAX_VALUE;
      int maxLeaves = Math.min(n + costs.length * 3, n * (costs.length - 1));
      while (tree.getLeafCount() <= maxLeaves) {
         if (tree.getLeafCount() >= n) {
            Tree optimized = tree.clone();
            optimized.optimize(probabilities, OPTIMIZATION_STEPS);
            double cost = optimized.getCost(probabilities);
            if (cost <= bestCost) {
               bestCost = cost;
               best = optimized;
            }
```

```
            }
            tree.expand();
        }
        return best;
    }

}

public class Tree {

    public int optimize(final Double[] probabilities, final int optimizationSteps) {
        for (int i = 0; i < optimizationSteps; i++) {
            if (!optimizationStep(probabilities)) {
                return i + 1;
            }
        }
        return optimizationSteps;
    }

    private boolean optimizationStep(final Double[] probabilities) {
        int bestAction = -1;
        double bestCost = Double.MAX_VALUE;

        bestAction = getBestAction(probabilities, bestAction, bestCost);

        return applyAction(bestAction);
    }

    private int getBestAction(final Double[] probabilities, int bestAction, double bestCost) {
        for (int i = 0; i < nodes.size(); i++) {
            final Node node = nodes.get(i);
            if (node.isLeaf) {
                continue;
            }
            testedDepths.clear();
            List<Node> children = new ArrayList<>(node.children);
            Set<Node> descendants = node.getDescendants();
            node.clearChildren();
            leaves.add(node);
            List<Node> leavesClone = new ArrayList<>(leaves);
            for (final Node leaf : leavesClone) {
                if (descendants.contains(leaf) || testedDepths.contains(leaf.depth)) {
                    continue;
                }
                testedDepths.add(leaf.depth);
                leaves.remove(leaf);
                leaf.addChildren(children, depths);
                double cost = getCost(probabilities);
                leaf.clearChildren();
                leaves.add(leaf);
                if (cost < bestCost) {
                    bestCost = cost;
                    bestAction = i + (leaf.depth << 16);
                }
            }
            node.addChildren(children, depths);
            leaves.remove(node);
```

```
        }
        return bestAction;
    }

    private boolean applyAction(final int bestAction) {
        if (bestAction == -1) {
            return false;
        } else {
            int nodeIndex = bestAction & 0xFFFF;
            int targetDepth = bestAction >> 16;
            Node node = nodes.get(nodeIndex);
            Set<Node> descendants = node.getDescendants();
            Node target = null;
            for (Node leaf : leaves) {
                if (leaf.depth == targetDepth && !descendants.contains(leaf)) {
                    target = leaf;
                    break;
                }
            }
            if (target == null) {
                return false;
            }
            if (target.equals(node)) {
                return false;
            }
            leaves.remove(target);
            target.addChildren(node.children, depths);
            node.clearChildren();
            leaves.add(node);
            return true;
        }
    }

}
```

Erweiterung

Die Perlen werden auf eine geschlossene Kette aufgefädelt. Da diese ein Kreis ist, ist nicht klar, wo der Anfang der Nachricht ist.

Lösungsidee

Es wird ein Extrazeichen generiert, welches den Anfang anzeigt und nicht mit anderen Zeichen verwechselt werden kann. Dieses wird Marker genannt.

Algorithmus

Man iteriert über alle Blätter des Baumes, die noch nicht belegt sind und überprüft, ob der damit verbundene Code als Marker geeignet ist. Falls ja nimmt man diesen.

Ein Code ist als Marker geeignet, wenn folgendes sichergestellt ist:
Wenn ich den Code auf der Kette sehe, ist das der Marker und keine Kombination von anderen Zeichen und eventuell dem Code selbst.

Also ist ein Marker als Code geeignet, wenn keine Kombination von einem Zeichenende, beliebig vielen Zeichen und am Ende der Anfang eines Zeichens diesen Code ergibt.

Wenn es keine unbelegten Blätter gibt oder keines von diesen als Marker geeignet ist, nimmt man das höchste unbelegte Blatt. Falls es nur *n* Blätter gibt, das niedrigste Blatt. Dieses erweitert man, sodass es nun *r* Kinder hat. Dann wiederholt man den ganzen Prozess.

Umsetzung

In der *Tree*-Klasse wurden die Methoden *createMarker* und *isMarker* hinzugefügt. *CreateMarker* wird in der *Encoder*-Klasse aufgerufen, nachdem der Baum generiert wurde.

Quellcode

```
public class Encoder {

    public static Map<String, String> generateTable(final String msg, final int[] costs) {
        final MapInt mapInt = getCounts(msg);
        Map<Character, AtomicInteger> counts = mapInt.counts;
        final int n = counts.size();
        int sum = mapInt.sum;
        Double[] probabilities = getProbabilities(counts, n, sum);
        Arrays.sort(costs);

        Tree best = getBest(probabilities, costs);
        Node marker = best.createMarker(n);
        final Map<String, String> map = getMap(counts, best);
        map.put("marker", marker.code);
        return map;
    }
}

public class Tree {
    public Node createMarker(int n) {
        nodes.get(0).startRecursiveCodeAssign();
        Collections.sort(leaves);
        Set<String> codes = getCodeSet(n);
        for (int j = n; j < leaves.size(); j++) {
            final Node potentialMarker = leaves.get(j);
            boolean isMarker = isMarker(codes, potentialMarker, n);
            if (isMarker) {
                return potentialMarker;
            }
        }
        int index = n;
        while (true) {
            boolean nLeaves = leaves.size() == n;
            if (nLeaves) {
                index--;
            }
            final Node node = leaves.get(index);
            expand(node, index);
            node.assignCodesRecursively(node.parent.code, node.index);
            Collections.sort(leaves);
            codes = getCodeSet(n);
```

```
            final List<Node> newNodes = node.children;
            for (int i = nLeaves ? 1 : 0; i < newNodes.size(); i++) {
                final Node potentialMarker = newNodes.get(i);
                if (isMarker(codes, potentialMarker, n)) {
                    return potentialMarker;
                }
            }
            if (nLeaves) {
                index++;
            }
        }
    }

    private boolean isMarker(final Set<String> codes, final Node potentialMarker, int n) {
        boolean isMarker = true;
        leaves.add(n, potentialMarker);
        for (int j = 0; j < n+1; j++) {
            final Node leaf = leaves.get(j);
            if (leaf.equals(potentialMarker)) {
                continue;
            }
            for (int i = 1; i <= Math.min(leaf.code.length(), potentialMarker.code.length()); i++) {
                if (leaf.code.substring(leaf.code.length() - i).equals(potentialMarker.code.substring(0, i))) {
                    int lastIndex = i;
                    for (int k = i; k < potentialMarker.code.length(); k++) {
                        final String substring = potentialMarker.code.substring(lastIndex, k + 1);
                        if (codes.contains(substring)) {
                            lastIndex = k + 1;
                        }
                    }
                    if (lastIndex < potentialMarker.code.length()) {
                        for (int k = 0; k < n; k++) {
                            final Node oLeaf = leaves.get(k);
                            if (oLeaf.code.startsWith(potentialMarker.code.substring(lastIndex))) {
                                isMarker = false;
                                break;
                            }
                        }
                    } else {
                        isMarker = false;
                        break;
                    }
                }
            }
            if (!isMarker) {
                break;
            }
        }
        leaves.remove(n);
        return isMarker;
    }

}
```

Beispiele

Schmuck 1

n=25 r=3

Max steps: 4 avg steps: 3.0

Best tree leaf count: 33/50 with cost: 3.4107142857142865

marker: 00220022, e: 2, t: 100, : 111, i: 011, n: 010, r: 101, w: 110, B: 020, I: 120, b: 021, ": 112, f: 0011, s: 0001, D: 121, F: 122, N: 012, W: 0010, a: 0012, d: 022, h: 0020, k: 0002, m: 0021, o: 00001, u: 00000, ü: 102

Encoded In 171 pearls (38.169643%) = 203mm

Decoded shifted as: BWINF steht für "Die Bundesweiten Informatikwettbewerbe"

Schmuck 5

n=41 r=7

Max steps: 4 avg steps: 3.5

Best tree leaf count: 61/82 with cost: 3.1245059288537536

marker: 626, : 11, e: 10, t: 010, i: 12, o: 21, s: 02, a: 001, n: 000, r: 3, c: 011, d: 20, l: 002, m: 03, h: 4, u: 012, f: 22, p: 13, b: 14, y: 04, g: 23, .: 003, ,: 014, q: 50, v: 013, x: 24, w: 004, F: 51, T: 15, A: 06, G: 61, H: 015, S: 016, ’: 25, j: 16, -: 050, 1: 052, 2: 60, 5: 006, 9: 52, z: 005, ;: 051

Encoded In 2266 pearls (27.989132%) = 3176mm

Decoded shifted as: Summary This paper gives a method for constructing minimum-redundancy prefix codes for the general discrete noiseless channel without constraints. The costs of code letters need not be equal, and the symbols encoded are not assumed to be equally probable. A solution had previously been given by Huffman in 1952 for the special case in which all code letters are of equal cost. The present development is algebraic. First, structure functions are defined, in terms of which necessary and sufficient conditions for the existence of prefix codes may be stated. From these conditions, linear inequalities are derived which may be used to characterize prefix codes. Gomory’s integer programming algorithm is then used to construct optimum codes subject to these inequalities; an elegant combinatorial approach to obtain a strictly computational experience is presented to demonstrate the practicability of the method. Finally, some additional coding problems are discussed and a problem of classification is treated.

Schmuck 9

n=674 r=4

Max steps: 54 avg steps: 51.0

Best tree leaf count: 685/1348 with cost: 7.9989075813851365

marker: 12102320121023, の: 00000, た: 21, に: 020, い: 002, し: 10000, と: 003, を: 00010, て: 111, は: 31, 、: 0110, な: 012, 。: 0002, が: 300, る: 00100, こ: 00001, で: 22, か: 01000, っ: 102, う: 01001, ら: 00101, れ: 00011, す: 11000, り: 00002, ー: 10100, き: 10001, ま: 00110, そ: 10010, ン: 20000, く: 01010, だ: 0103, よ: 320, あ: 00102, さ: 3010, も: 302, わ: 0013, : 00012, : 20001, ・: 11001, お: 10011, 「: 01011, 」: 10110, け: 20100, ル: 20010, え: 01020, ち: 10101, ん: 01110, つ: 230, ス: 00030, ラ: 10002, イ: 01012, カ: 00031, ト: 02200, 見: 20200, 人: 010020, ウ: 10300, レ: 231, ど: 021000, や: 01300, ジ: 00112, 聖: 03100, 大: 3011, 彼: 10012, 者: 120000, め: 00013, エ: 01003, 上: 20002, 思: 10030, せ: 12001, ア: 13000, フ: 321, 出: 01120, 堂: 20011, ば: 12100, サ: 10111, 車: 11011, 建: 02110, 道: 11020, み: 01021, む: 20020, タ: 330, チ: 00103, ョ: 02300, 海: 02101, 言: 10021, 中: 20110, 手: 100030, 物: 03110, 事: 10310, げ: 03002, べ: 20210, ク: 01301, 一: 01121, 会: 030100, 地: 201010, 口: 0300000, 通: 110020, 水: 02111, ね: 0000300, び: 12011, ほ: 02102, ょ: 120100, ろ: 12101, オ: 1101000, マ: 02201, 部: 112000, 入: 10112, 主: 01022, 乗: 10031, 立: 101200, 雨: 01013, 任: 11021, 築: 13100, ….

Encoded In 20601 pearls (56.26229%) = 36645mm

Decoded shifted as: 英国陸地測量部制作の地図ではカラン・ウォーフ、地元の人には単にカランと呼ばれている場所は、今でこそ海岸線から二マイルほど内陸部にあるが、かつてはもっと海寄りで、無敵艦隊との戦いに六隻の船を送り、その一世紀後にはオランダの攻撃を迎え撃つため四隻の船を送り出した由緒ある港として歴史に輝かしい名を残している。ところがやがてカル川の河口域は沈泥でふさがって港口には砂州ができ、海上貿易の船は他に港を探さざるを得なくなった。その後、カル川の流れはやせ細り、それまでのようにあちらこちらへ縦横に伸びるかわりに身を縮めておとなしい河川に変貌し、しかも河川としても決して大きいほうの部類ではなかった。市民たちは港で生計が立てられないことを見て取ると、塩沢を埋め立てることでなにがしかの代償が得られるかも知れないと考え、海水を防ぐために石の堤防を築き、その真ん中にカル川の流れを海に放出する水路を造った。こうしてカラン・….