CODIERUNGSTHEORIE - PRAKTIKA 1

Wombacher Sascha

13. Dezember 2015

AGENDA

1. Generierung des Huffman-Baums
2. Zeichencodierung
3. Rekunstruktion des Huffman-Baums
4. Lesen des codierten Strings
5. Entropie

GENERIERUNG DES HUFFMAN-BAUMS

Generierung des Huffman-Baums - Wahrscheinlichkeit eines Zeichen

1. für jedes Zeichen *z* der Eingabezeichenkette wird die Häufigkeit (Anzahl) errechnet
2. Anschließend wird diese Anzahl durch die Eingabelänge Normiert (Intervall zwischen (0, 1])

Generierung des Huffman-Baums - Wahrscheinlichkeit - Programmcode

```
1   float numElements = 0;
2   for (auto& e : containerType){
3       bool elementFound = false;
4       for (auto& iter : elementCounter){
5           if (iter->data.first == e){
6               elementFound = true;
7               ++iter->data.second;
8               break;
9           }
10      }
11      if (!elementFound){
12          elementCounter.reserve(1);
13          elementCounter.push_back(std::make_unique<LEAVE>(LEAVE(e, 1.f)));
14      }
15      ++numElements;
16  }
17  elementCounter.shrink_to_fit();
18  for (auto& e : elementCounter)
19      e->data.second /= numElements;
```

Generierung des Huffman-Baums - Baumgenerierung

1. Erstelle je ein Node pro Zeichen
2. Sortiere alle Nodes aufsteigend anhand ihrer Wahrscheinlichkeit
3. Verbinde die zwei Nodes mit den geriengsten Wahrscheinlichkeiten
4. Füge das entstehende Node in die Zeichenliste ein (Wahrscheinlichkeit = Summe der einzel Nodes)
5. Stelle die Sortierung wieder her (*insertion sort* - Ansatz)
6. Wiederhole Schritt 3-5 bis nur noch ein Node existiert (=> Tree-Root)

Generierung des Huffman-Baums - Baumgenerierung - Programmcode

```
1  // sort elements
2  std::sort(std::begin(elementCounter), std::end(elementCounter), [](const auto&
       element1, const auto& element2) -> bool{
3      // data.second = symbol probability
4      if (element1->data.second < element2->data.second)
5          return true;
6      return false;
7  });
```

Generierung des Huffman-Baums - Baumgenerierung - Programmcode

```
1  // create huffman tree
2  for (std::size_t i = 0; i < elementCounter.size() - 1; ++i){
3      // pick first 2 elements and stick them together
4      // and stick them into the second element -> last element will be the tree's root
5      std::unique_ptr<NODE>& ptr1 = elementCounter[i];
6      std::unique_ptr<NODE>& ptr2 = elementCounter[i + 1];
7
8      std::unique_ptr<NODE> ptr = std::make_unique<NODE>(ptr1->data.first, ptr1->data.
         second + ptr2->data.second);
9      ptr-> leftNode = std::move(ptr1);
10     ptr->rightNode = std::move(ptr2);
11
12     ptr2 = std::move(ptr);
13     ptr2->leftNode ->parentNode = ptr2.get();
14     ptr2->rightNode->parentNode = ptr2.get();
15
16     // sort elements again, using the "insertion sort" appraoch
17     for (int iter = i + 1; iter < elementCounter.size() - 1; ++iter){
18         auto& e1 = elementCounter[iter];
19         auto& e2 = elementCounter[iter + 1];
20
21         if (e1->data.second > e2->data.second)
22             std::swap(e1, e2);
23         else
24             break;
25     }
26 }
```

Generierung des Huffman-Baums - Speicherung

für die Weiterverarbeitung ist das Speichern des Huffman-Baums ein wichtiger Bestandteil

1. Ersten 32Bit beschreiben die Anzahl der Zeichen im Baum (Blätter)
2. Die Folgenden 8Bit beschreiben die Länge eines Zeichens (Bsp.: char = 1, int32 = 4)
3. Nehme ein Zeichen aus Huffman-Baum
4. Schreibe die Gesamtlänge des Zeichen + Codierung in die Folgenden 8Bit
5. Schreibe das Zeichen in die Nächsten Bits
6. Schreibe die Codierung des Zeichen in die folge Bits
7. Wiederhole Schritt 3-6 für jedes Zeichen im Huffman-Baum

Generierung des Huffman-Baums - Speicherung - Programmcode

```
1  BitWriter<> writer(output);
2  const char* count = reinterpret_cast<const char*>(&leaveCount);
3  writer.addByte(count[0]);
4  writer.addByte(count[1]);
5  writer.addByte(count[2]);
6  writer.addByte(count[3]);
7
8  for (const _Leave<T>* ptr = this->m_FirstLeave; ptr; ptr = ptr->nextLeave){
9       writer.addByte(8 * (1 + (char)sizeof(T)) + (char)ptr->m_Coding.size());
10
11      // data: first = symbol
12      const char* tmpPtr = reinterpret_cast<const char*>(&ptr->data.first);
13
14      for (int i = 0; i < sizeof(T); ++i)
15           writer.addByte(tmpPtr[i]);
16
17      writer.addBits(ptr->m_Coding);
18 }
19 writer.flush();
```

ZEICHENCODIERUNG

ZEICHENCODIERUNG

1. Generiere pro Zeichen die jeweilige Codierung
   (von Root gesehen: leftNode = 1, rightNode = 0)
2. Lese ein Zeichen *z* des Input-Strings
3. Finde die Codierung für *z*
4. Füge die Codierung dem BitWriter hinzu
5. Wiederhole Schritte 2-4 für alle Zeichen des Strings
6. Flush für den BitWriter

ZEICHENCODIERUNG - PROGRAMMCODE TEIL 1

```
1  // rucursive call starting at tree root
2  bool isLeave = true;
3  if (this->leftNode){
4      isLeave = false;
5      this->leftNode->generateCodings();
6  }
7  if (this->rightNode){
8      isLeave = false;
9      this->rightNode->generateCodings();
10 }
11 if (isLeave){
12     _Leave<T2>* THIS = static_cast<_Leave<T2>*>(this);
13     std::vector<bool>& coding = THIS->m_Coding;
14     for (const _Node<T2>* currentPtr = this->parentNode, *previousePtr = this;
15          currentPtr;
16          currentPtr  = currentPtr->parentNode)
17     {
18          // 1 coding left, 0 right
19          if (currentPtr->leftNode.get() == previousePtr)
20              coding.push_back(1);
21          else
22              coding.push_back(0);
23          previousePtr = currentPtr;
24     }
25     // vector is generated by bottom up approach -> inverse its order for top down
26     std::reverse(std::begin(coding), std::end(coding));
27 }
```

ZEICHENCODIERUNG - PROGRAMMCODE TEIL 2

```
1  BitWriter<> writer(output);
2  // generate coding table
3  std::array<const _Leave<T>*, 256> table;
4  for (auto& e :table)
5       e = nullptr;
6
7  for (const _Leave<T>* iter = this->m_FirstLeave; iter; iter = iter->nextLeave)
8      table[iter->data.first] = iter;
9  // write symbols
10 for (const auto& e : str){
11     if (!table[e])
12         throw "Error, symbol not found in huffmantree";
13
14     writer.addBits(table[e]->m_Coding);
15 }
16 writer.flush();
```

REKUNSTRUKTION DES HUFFMAN-BAUMS

Rekunstruktion des Huffman-Baums

1. Öffne die erstellte Header-Datei, siehe Folie *9*
2. Lese *invertert* wie zuvor beschrieben

LESEN DES CODIERTEN STRINGS

LESEN DES CODIERTEN STRINGS

1. Setze Pointer auf Root
2. Lese ein Bit des Inputstreams (BitReader)
3. Verfolge Pointer anhand des Bits
   (1: Pointer = Pointer->left, 0: Pointer = Pointer->right)
4. Zeigt der Pointer auf ein Blatt?
   - Wenn Ja : Schreibe Zeichen, setze Pointer auf Root
5. Wiederhole Schritt 2 bis 4 für jedes Bit des Inputstreams

LESEN DES CODIERTEN STRINGS - PROGRAMMCODE

```
1  std::fstream input(filename, std::fstream::in);
2  if (!input)
3      throw "Error, file can't be opened for reading";
4
5  float bitCount = 0.f;
6  float characterCount = 0.f;
7
8  BitReader<> reader(input);
9  while(reader.good()){
10     std::unique_ptr<_Node<T>>* iter = &this->m_Root;
11
12     // find symbol
13     while((*iter)->leftNode || (*iter)->rightNode){
14         bool left = reader.readBit();
15         ++bitCount;
16
17         if (left)
18             iter = &(*iter)->leftNode;
19         else
20             iter = &(*iter)->rightNode;
21     }
22     container.push_back((*iter)->data.first);
23     ++characterCount;
24 }
25 std::cout << "Newly calculated entropie: " << bitCount / characterCount << std::endl;
```

ENTROPIE

ENTROPIE

Die Entropie wird auf zwei wegen errechnet und verglichen

- Wie auf der Folie 19 zu erkennen
  - Zähle die verwendeten bits
  - Dividire diese durch die Anzahl der erhaltenen Zeichen
- Errechnung nach der Huffman-Baum Generierung
  - Berechne die Einzelwahrscheinlichkeiten jedes Zeichens
  - Multipliziere dies mit der Codierlänge des Zeichens
  - Die Aufaddierung dieser entspricht die Entropie

VIELEN DANK FÜR DIE AUFMERKSAMKEIT!