Übungen zur Vorlesung

"Objektorientiertes Programmieren im Wissenschaftlichen Rechnen"

Dr. Olaf Ippisch, Ole Klein

Abgabe am 02. 07. 2013 in der Vorlesung

ÜBUNG 1 DÜNNBESETZTE MATRIZEN

In Anwendungen im wissenschaftlichen Rechnen haben Matrizen oft sehr viele Einträge die 0 sind. Diese sogenannte dünnbesetzte Matrixstruktur ist eine direkte Folge der Diskretisierung. Für solche dünnbesetzte Matrizen gibt es effiziente Verfahren, um ein Gleichungssystem näherungsweise zu lösen. Der in der vorherigen Übung implementierte Gauss-Seidel-Algorithmus ist ein Beispiel dafür.

Um diese Verfahren effizient zu implementieren (sowohl Rechenzeit, als auch Speicherverbrauch) ist es üblich diese dünnbesetzen Strukturen auch bei der Speicherung der Matrix zu berücksichtigen.

In dieser Übung wollen wir eine einfache Variante einer dünnbesetzten (sparse) Matrix implementieren.

- Wir verwenden hierzu den Compressed-Row-Storage Ansatz.
- Diese soll unter dem generischen Iterator-Interface von Blatt 8 verfügbar sein: `RowIterator` und `ColIterator`, sowie die dazugehörigen `begin` und `end` Methoden.
- Einträge die 0 sind werden nicht gespeichert und vom Iterator übersprungen (da sie bei Matrix-Operationen keinen Beitrag haben).
- Da nur die Non-Zero Einträge gespeichert werden, muss die Anzahl dieser Matrix mitgeteilt werden. Das passiert im Konstruktor:

```
SparseMatrix(unsigned int rows, unsigned int cols, unsigned int nonZeros);
```

- Die Datenstruktur sieht folgendermaßen aus (Pseudocode):

```
template<typename T>
SparseMatrix {
    T data[nonZeros];
    unsigned int column[nonZeros];
    unsigned int rowOffset[rows];
};
```

  - Die Daten werden zeilenweise in `data` gespeichert.
  - Zu jedem Non-Zero Eintrag wird im `column` Vektor gespeichert, in welcher Spalte er steht.
  - Der `rowOffset` Vektor speichert für jede Zeile, wo die Daten dieser Zeile in den `data`- und `column`-Vektoren beginnen.

Beispiel (Matrix vom letzten Übungsblatt):

```
rows: 3
cols: 3
nonZeros: 7
data: 2, 1, 1, 3, 2, 0, -4
column: 0, 1, 0, 1, 2, 1, 2
rowOffset: 0, 2, 5
```

- Testen Sie Ihre Implementierung mit den beiden Matrix-Algorithmen vom letzten Blatt. Verwenden Sie die beschriebenen Testfälle.

12 Punkte

ÜBUNG 2 STL-ALGORITHMEN: VEKTORNORM

Gegeben ein Vektor $x \in \mathbb{R}^n$, kennen wir verschiedene Vektornormen, z.B.:

$$||x||_2 = \sqrt{x \cdot x}, \qquad ||x||_1 = \sum_{i=0}^{n-1} |x_i|, \qquad ||x||_\infty = \max_{i=0}^{n-1} |x_i|$$

Der Vektor x sei als STL-Container gespeichert.

1. Implementieren Sie die obigen Vektornormen unter Verwendung der STL-Algorithmen.
   - Welche Algorithmen können hier eingesetzt werden, um Programmieraufwand zu sparen?
   - Würde sich Ihre Wahl des verwendeten Algorithmus ändern, wenn der Container modifiziert werden darf? Wenn ja wie?
2. Schreiben Sie die Algorithmen so um, dass diese möglichst viel Code teilen.
   - Identifizieren Sie die Gemeinsamkeiten der verschiedenen Normberechnungen.
   - Entwerfen Sie ein Interface um diese Gemeinsamkeiten auszunutzen.
   - Welche Art von Austauschbarkeit würden Sie für die sich unterscheidenden Teile wählen und warum? (Template-Spezialisierung, statischer Polymorphismus, dynamischer Polymorphismus)

8 Punkte

Wie immer gilt: Kommentieren Sie Ihr Programm. Erklären Sie was Sie tuen.