Einführung in die Programmierung mit C++

Übungsblatt 8

Permutationen von Matrixen und Graphen

Sebastian Christodoulou, Alexander Fleming und Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Kanten mit `std::pair<int, int>` Beschreiben

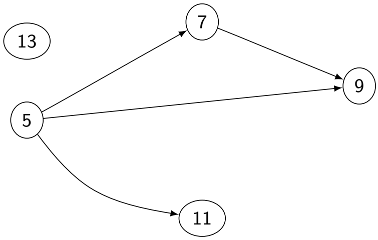
```
5
5 7 9 11 13
4
0 1
0 2
1 2
0 3
```

In dieser Übung kodieren wir eine Graph in zwei Variablen:

- Eine std::vector<**int**> kn für die Knoten-Beschriftungen.
- Eine std::vector<std::pair<**int**, **int**>> ka für die Kanten.

kn[0]	kn[1]	kn[2]	kn[3]	kn[4]
5	7	9	11	13

ka[0]	ka[1]	ka[2]	ka[3]
{0, 1}	{0, 2}	{1, 3}	{0, 3}



Eine Permutation ist eine Operator, die die Reihenfolge der Elemente einer Struktur (z.B. eines Vektors) umordnet. Wir schreiben sie wie folgt auf

$$\pi_{(2,0,1)} = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \end{pmatrix} \tag{1}$$

Beide Zeilen beschreiben Indizes einer Struktur. Auf den Indizes der oberen Zeile werden die Werte der Indizes in der unteren Zeile agebildet. Haben wir z.B. einen Vektor $v = (10, 25, 26)$ vor uns, so bewirkt die Permutation folgendes:

$$\pi_{(2,0,1)}(v) = (26, 10, 25)$$

Wobei $v[2]$ auf index 0 abgebildet, $v[0]$ auf index 1 und $v[2]$ auf index 1.
Die obere Zeile der n-Permutation π ist stets die Folge $[0, \ldots, n]$ (bei obiger 3-permutation $[0, 1, 2]$). Also beschreibt die untere Zeile die Permutation allein komplett.
Permutationen können Elemente auch an ihrem Platz lassen, z.B. in

$$\pi_{(2,1,0)}(v) = (25, 10, 26)$$

bleibt $v[1]$ an ursprünglicher Stelle.

Wir weiten das Konzept der Permutation außerdem auf *quadratische* Matritzen aus. Auf Matritzen wird die Reihenfolge der Zeilen *und* der Spalten umgeordnet.

$$\pi_{(2,0,1)}\left(\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}\right) = \begin{pmatrix}9 & 7 & 8\\ 3 & 1 & 2\\ 6 & 4 & 5\end{pmatrix}$$

Hier wurde Zeile $0 \rightarrow 2$, Zeile $1 \rightarrow 0$, und Zeile $2 \rightarrow 1$ abgebildet. Danach wurde Spalte $0 \rightarrow 2$, Spalte $1 \rightarrow 0$, und Spalte $2 \rightarrow 1$ abgebildet. Visuell:

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix} \overset{\textit{Zeilen}}{\longrightarrow} \begin{pmatrix}7 & 8 & 9\\ 1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} \overset{\textit{Spalten}}{\longrightarrow} \begin{pmatrix}9 & 7 & 8\\ 3 & 1 & 2\\ 6 & 4 & 5\end{pmatrix}$$

Als "Sanity Check" kann man merken, dass Elemente, die auf der Diagonal der ursprünglichen Matrix liegen, liegen auch auf der Diagonal der umgeordneten Matrix.

Zwei Graphen G_1 und G_2 sind isomorph wenn eine Permutation $\pi_{(k)}$ verwendet auf die Adjazenzmatrix von G_1 ergibt die Adjazenzmatrix von G_2.

1. Implementiere kanten_zu_adjazenzmatrix. Diese Funktion erhält als erstes argument einen std::vector<std::pair<**int**, **int**>> **const**& und ergibt einen std::vector<std::vector<**int**>>, der die Adjazenzmatrix entspricht.
   - Siehe Übung 5.
2. implementiere permute_vector. Diese Funktion erhält als erstes Argument einen std::vector, der permutiert zurückgegeben werden soll. Das zweite Arguent beschreibt die Permutation π.
3. implementiere permute_matrix. Diese erhält als erstes Argument eine Matrix, und als zweites die Permutation, π. Diese soll auf die Zeilenreihenfolge, dann auf die Spaltenreihenfolge der Matrix angewandt werden.

Bei allen Eingaben nehmen wir natürlich an, dass eine n-Permutation durch einen std::vector der Länge n beschrieben wird. Dessen Einträge sind die untere Zeile in 1.

4. Die Menge aller möglichen 3-Permutationen erhält man dadurch, dass man im Tupel (0,1,2) alle verschiedenen Reihenfolgen bildet, also $\{(0,1,2),(0,2,1),(1,0,2),\ldots\}$.
   - Wie viele 3-Permutationen gibt es?
   - Wie viele 4-Permutationen gibt es?

Permutationen von Matritzen und Graphen

Aufgaben II

5. Ergänze die Funktionen all_permutations(std::vector<T> a) und recursive_permutation_helper(std::vector<T> &a_active, **int** active_length, std::vector< std::vector<T>> &permutation_list).
   - all_permutations initializiert permutation_list und active_length, dann ruft recursive_permutation_helper(a, active_length, permutation_list) auf.
   - **Base Case:** permutation_length <= 1 und eine Kopie von a_active wird zum permutation_list hinzugefügt.
   - **Recursive Case:**
     recursive_permutation_helper(a, active_length−1, permutation_list) wird aufgerufen. Dann, für jeder i = 0 bis active_length−1:
     - Wenn active_length *gerade* ist, wird a[active_length−1] mit a[i] getauscht.
     - Wenn active_length *ungerade* ist, wird a[active_length−1] mit a[0] getauscht.
     - recursive_permutation_helper(a, active_length−1, permutation_list) aufgerufen.
   - Dieser Algorithmus heißt *Heap's Algorithm*.
6. Lesen Sie die Graphen G_1, G_2, G_3 aus den Dateien `graph_1.txt`, `graph_2.txt`, und `graph_3.txt` aus. Welche Paaren von Graphen sind isomorph? Unter welcher Permutation $\pi_{(k)}$? Die Funktionen vector_gleichheit und matrix_gleichheit werden hilfreich sein.

Aufgaben III

Abgabe:

- `8.cpp`