Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Prof. Dr. Peter Thiemann
Dr. Philipp Klaus Krause

Sommersemester 2019

5. Programmieren in C

Abgabe bis 4. Juni, 10:15 GMT+2.

Aufgabe 0:

Die Dateien `stree.c` und `stree.h` auf der Website zur Vorlesung enthalten eine Implementierung eines Binärbaums von Strings mit 4 Fehlern. Übernimm diese und schreibe Tests in `stree0test.c`, `stree1test.c` und `stree3test.c`, die aufgrund 3 verschiedener in der Implementierung enthaltener Fehler fehlschlagen! Füge für diese Tests dem Makefile eine Variable INVTESTBINARIES hinzu, für Tests, die fehlschlagen sollen, und passe die Variable BINARIES entsprechend an! Das Target `test` im Makefile soll so aussehen:

```
# Führe Tests aus
test: $(TESTBINARIES) $(INVTESTBINARIES)
    $(foreach test,$(TESTBINARIES), ./$(test) &&) echo "Passed all that should pass."
    $(foreach test,$(INVTESTBINARIES), ! ./$(test) &&) echo "Failed all that should fail."
```

Aufgabe 1:

Lege eine Kopie `stree-fixed.c` von `stree.c` an, und behebe darin die 4 Fehler. Erstelle dazu eine `stree-fixedtest.c`, die unter anderem die Tests aus Aufgabe 0 enthält.

Aufgabe 2:

Implementiere eine dynamische Adjazenzmatrix für gerichtete Graphen (mittels der Quadratschalenmethode aus der Vorlesung) in `adjm.c` und `adjm.h`! Das Interface besteht aus dem Typen `adjm_t` und den Funktionen

```
adjm_t *adjm_init(adjm_t *a, size_t num_nodes) // Neue Adjanzenzmatrix für Graph mit
                                               // num_nodes Knoten ohne Kanten. Gibt
                                               // im Fehlerfall 0 zurück, sonst die Matrix.
size_t adjm_size(const adjm *a); // Anzahl der Knoten im Graph.
void adjm_destroy(adjm_t *a) // Gibt Speicher frei.
```

Beispiel:

```
adjm_t a1;
adjm_t a2;
adjm_init(&a1, 0); // Leerer Graph
adjm_init(&a2, 8); // Graph mit 8 Knoten
assert(adjm_size(&a1) == 0);
assert(adjm_size(&a2) == 8);
adjm_destroy(&a1);
adjm_destroy(&a2);
```

Aufgabe 3:

Erweitere die Implementierung aus der vorigen Aufgabe wie folgt! Das Interface erhält die zusätzlichen Funktionen:

```
bool adjm_get_edge(const adjm_t *a, size_t n, size_t m)
adjm_t *adjm_resize(adjm_t *a, size_t num_nodes) // Ändert Graphgröße. Gibt im Fehlerfall 0
                                                 // zurück, sonst die Matrix.
void adjm_set_edge(adjm_t *a, size_t n, size_t m, bool e) // Fügt Kante ein oder löscht sie.
```

Bei einer Änderung der Graphgröße entfallen alle Kanten zwischen nicht mehr existierenden Knoten. Zwischen den neuen Knoten gibt es keine Kanten. Zwischen beibehaltenen Knoten bleiben die Kanten wie sie sind. Beispiel:

```
adjm_t a;
adjm_init(&a, 0); // Leerer Graph
adjm_resize(&a, 4); // Graph mit 4 Knoten
adjm_set_edge(&a, 0, 1, true);
adjm_set_edge(&a, 1, 2, true);
assert(adjm_get_edge(&a, 0, 1));
assert(!adjm_get_edge(&a, 1, 0));
assert(adjm_get_edge(&a, 1, 2));
assert(!adjm_get_edge(&a, 2, 1));
adjm_resize(&a, 2); // Graph mit 2 Knoten
adjm_resize(&a, 3); // Graph mit 3 Knoten
assert(adjm_get_edge(&a, 0, 1));
assert(!adjm_get_edge(&a, 1, 2));
adjm_destroy(&a);
```

Hinweise (gelten für dieses wie auch für alle weiteren Blätter):

- Die Abgabe erfolgt über Subversion in ein Unterverzeichnis Blatt5 (entsprechend bei den folgenden Blättern dann Blatt6, etc).
- Um uns mitzuteilen, wie du bisher mit Vorlesung und Übungsblatt zurechtkommst, für Verbesserungsvorschläge und dergleichen, kannst du eine Datei `Erfahrungen.txt` mit hochladen. Am besten schon bis Montagmittag.
- Für jede nichttriviale Funktion soll es einen Test geben. Auch das Verhalten bei eventuellen Grenzfällen (leere Eingabe, etc) soll getestet werden. Der Beispielcode aus der Aufgabenstellung ist im Allgemeinen als Test nicht ausreichend!