Projekt Fortgeschrittene Datenstrukturen im SS 2022

In diesem Dokument wird das Programmierprojekt für das Sommersemester 2022 beschreiben. Ziel des Projektes ist es, mehrere Datenstrukturen, die in der **Vorlesung Fortgeschrittene Datenstrukturen vorgestellt** wurden, zu implementieren. Neben der Implementierung ist eine Dokumentation, eine Evaluation und eine Abschlusspräsentation Bestandteil des Projektes.

Dynamische Bitvektoren und BP

In diesem Projekt müssen folgende Datenstrukturen implementiert werden:

1. Ein dynamischer Bitvektore, welcher neben den Operationen *access*, *insert* , *delete* und *flip* unterstützt auch *rank*- und *select*-Anfragen unterstützt.
2. Eine Dynamische BP-Datenstruktur, die in der Vorlesung beschriebenen Operationen *i-th child*, *parent*, *suptree size*, *deletenode* und *insertchild* unterstützt.

Ein- und Ausgabe

Das Programm muss per Kommandozeile steuerbar sein. Die Eingabe hierfür hat das folgende Format.

```
ads_programm_a [bv|bp] eingabe_datei ausgabe_datei
```

Der Parameter *bv* sorgt dafür, dass ein dynamischer Bitvektor entsprechend der Eingabedatei erstellt wird und der Parameter *bp* sorgt dafür, dass ein dynamischer succincter Baum in BP Repräsentation entsprechend der Eingabedatei erstellt wird.

bv. Hier enthält die erste Zeile eine Zahl $n \in \mathbb{N}_0$ und wird gefolgt von n Zeilen bestehend aus entweder `0` oder `1`. Diese folge aus 0en und 1en entspricht der initialen Konfiguration des Bitvektors. Anschließend folgt eine beliebige Anzahl an Zeilen der Form:

- `insert` i [0|1] (füge an die i-te Stelle im Bitvektor 0 oder 1 ein)
- `delete` i (lösche i-tes Bit)
- `flip` i (flippe i-tes Bit)
- `rank` [0|1] i (schreibe rank 0 oder rank 1 bis zur Position i in die Ausgabedatei)
- `select` [0|1] i (schreibe die Position der i-te 0 oder 1 in die Ausgabedatei)

Alle Anfragen sind zulässig, es wird beispielsweise keine Select-Anfrage gestellt, die nicht beantwortet werden kann. Beispiel:

```
4
0
1
1
1
insert 3 1
insert 0 0
select 0 2
```

Neben der Ausgabedatei soll noch eine Ausgabe in der Konsole der Form
`RESULT algo=bv name<first_last_name> time=<running_time_without_output_in_ms> space=<required_space_in_bits>`
erzeugt werden. Beispiel:
`RESULT algo=bv name<florian_kurpicz> time=<512>`

bp. Hier enthält die Eingabedatei eine beliebige Anzahl an Zeilen der Form:

- `deletenode` v (löscht Knoten v)
- `insertchild` v i k (definiert wie in der Vorlesung)
- `child` v i (schreibe$i$-tes Kind von v in die Ausgabedatei)
- `subtre_size` v (schreibe Größe des Subbaumes mit v als Wurzel (inklusive v) in die Ausgabedatei)
- `parent` v (schreibe Elter von v in die Ausgabedatei)

Alle Anfragen sind wieder zulässig. So wird beispielsweise nicht versucht die Wurzel zu löschen. Initial besteht der Baum lediglich aus der Wurzel.

Im Anschluss soll das Programm in die Ausgabedatei den Knotengrad[1] jedes Knotens während einer Tiefensuche (preorder) in jeweils eine Zeile schreiben.
Beispiel:

```
insertchild 0 1 0
insertchild 0 1 0
insertchild 0 1 0
insertchild 0 2 2
deletenode 1
```

Neben der Ausgabedatei soll noch eine Ausgabe in der Konsole der Form
`RESULT algo=bv name<first_last_name> time=<running_time_without_output_in_ms> space=<required_space_in_bits>`
erzeugt werden. Beispiel:
`RESULT algo=bp name<florian_kurpicz> time=<512>`

Minimalanforderungen

Das Projekt darf in einer beliebigen Programmiersprache umgesetzt werden. Wichtig ist aber, dass das Projekt auf einem Linux-System (Ubuntu 20.04.3 LTS) lauffähig ist! Dem Projekt sollte eine detaillierte Anleitung beiliegen, die beschreibt, was zum kompilieren/ausführen benötigt wird und wie genau das Programm ausgeführt werden kann. Das Programm muss das oben beschrieben Ein- und Ausgabeformat unterstützen.

Wichtig: **Es dürfen keine externen Bibliotheken verwendet werden**, die nicht von Florian Kurpicz genehmigt wurden. Bitte per Mail (`kurpicz@kit.edu`) nachfragen, bevor externe Bibliotheken eingebunden werden. Die Standardbibliothek der jeweiligen Programmiersprache kann allerdings ohne Fragen verwendet werden.

Dokumentation, Evaluation und Präsentation

Der Code muss so dokumentiert sein, dass dem Leser klar wird, was an welcher Stelle was genau passiert. Hierfür sollte darauf geachtet werden, dass die Dokumentation auch erklärt *warum* etwas gemacht wird und nicht nur *was* gemacht wird.

Die Evaluation ist Teil der Präsentation. Hier können unter anderem die Laufzeiten des eigenen Ansatzes für unterschiedliche Eingabe gezeigt werden. Ein Vergleich mit anderen Implementierungen ist auch möglich. (Hinweis: Das Ausgabeformat kann direkt zum Erstellen von Diagrammen genutzt werden, siehe `https://github.com/bingmann/sqlplot-tools/`.)

Die Ergebnisse sollen dann in einer **genau** fünfminütigen Präsentation vorgestellt werden. In der Präsentation soll neben der Evaluation der Implementierung darauf eingegangen werden, was implementiert wurde, wie es implementiert wurde und was eventuell besonders an der Implementierung ist.

Wettbewerb

Des Weiteren gibt es noch einen kleinen Wettbewerb, an dem jedes eingereichte Projekt automatisch teilnimmt. Das Abschneiden im Wettbewerb hat keinen Einfluss auf die Note des Projektes! Bei dem Wettbewerb werden alle Laufzeiten der von mir durchgeführten Tests (gewichtet) addiert. Die geringste Laufzeit gewinnt. Die Gewichtung ist: 45 % Laufzeit und 55 % Platzbedarf.

[1] Nur der Ausgrad.

Deadline

Das Projekt muss bis zum 15.07.2022 um 23:59 Uhr deutscher Zeit per Mail an `kurpicz@kit.edu` gesendet werden (gerne als Link zu einem Repository). Spätere Abgaben können leider nicht berücksichtigt werden.