Stefan Kurtz **Universität Hamburg**
Abteilung für Genominformatik **Zentrum für Bioinformatik**

Programmierung für Naturwissenschaften 2
Sommersemester 2020
Übungen zur Vorlesung: Ausgabe am 09.06.2020

Aufgabe 6.1 (7 Punkte) In der Vorlesung wurde die Methode der binären Suche in einem sortierten Array verwendet, um zu entscheiden, ob ein Schlüssel in dem Array vorkommt.

In dieser Aufgabe soll die binäre Suche für eine verallgemeinerte Fragestellung genutzt werden, nämlich für eine Intervallsuche. Dazu betrachten wir ein aufsteigend sortiertes Array a von n Fliesskommazahlen, die z.B. Messwerte repräsentieren können. Das Intervallsuchproblem besteht darin, für gegebene $\ell, h \in \mathbb{R}$, $\ell \leq h$ den Indexbereich i, j, $0 \leq i \leq j \leq n-1$ zu bestimmen, so dass $\ell \leq a[k] \leq h$ für alle k, $i \leq k \leq j$. Falls es kein $a[k]$ mit $\ell \leq a[k] \leq h$ gibt, dann soll das Ergebnis der leere Indexbereich sein. Der Indexbereich von i bis j umfasst also alle Werte in a, die im Intervall $[\ell, h]$ liegen.

Zur Lösung des Intervallsuchproblems bestimmt man i und j wie folgt:

- i ist minimal, so dass $\ell \leq a[i]$ ist.
- j ist maximal, so dass $a[j] \leq h$ ist.

Ihre Aufgabe ist es nun in der Datei `binsearch_interval.c` (die Sie durch Umbenennung aus der Datei `binsearch_interval_template.c` erhalten) zwei Funktionen zu implementieren, die ausschließlich auf binärer Suche basieren, also insbesondere, auch in Teilen, keine lineare Suche verwenden.

- Die Funktion

  ```
  Basetype *binsearch_gt_leq(const Basetype *array,size_t length,Basetype key)
  ```

  liefert einen Zeiger auf das grösste Element in `array`, das $\leq$ `key` ist. Falls dieses Element mehrmals auftaucht, soll der Zeiger auf das Vorkommen des Elements in `array` mit dem maximalen Index geliefert werden. Falls es kein Element in `array` $\leq$ `key` gibt, dann soll der `NULL`-Zeiger zurückgeliefert werden. Hierbei ist `Basetype` ein Synonym für **`double`**. `array` ist ein Verweis auf einen Speicherbereich mit `length` Elementen.

- Die Funktion

  ```
  Basetype *binsearch_sm_geq(const Basetype *array,size_t length,Basetype key)
  ```

  liefert einen Zeiger auf das kleinste Element in `array`, das $\geq$ `key` ist. Falls dieses Element mehrmals auftaucht, soll der Zeiger auf das Vorkommen des Elements in `array` mit dem minimalen Index geliefert werden. Falls es kein Element in `array` $\geq$ `key` gibt, dann soll der `NULL`-Zeiger zurückgeliefert werden.

In der genannten Datei finden Sie eine Implementierung einer Funktion `binsearch` zur exakten Suche nach einem Schlüssel. Hieran können Sie sich bei der Implementierung der beiden Funktionen orientieren. Insbesondere sollen Sie zum Vergleich der Werte die Funktion `basetype_compare` verwenden, um Ungenauigkeiten bei der Darstellung von Fliesskommazahlen zu berücksichtigen.

Hinweis: Es gibt eine einfache Möglichkeit, die beiden Funktionen zu implementieren. Wenn man es geschickt anstellt, benötigt man eine zusätzliche Variable und weniger Anweisungen als die Funktion `binsearch`.

In der genannten Datei finden Sie die Implementierung einer Funktion `binsearch_interval`, die die beiden zu implementierenden Funktionen aufruft und insbesondere auch die Fälle behandelt, in denen das Intervall leer ist. Im Programmcode dieser Funktion finden Sie Fragen 1-8 zu den einzelnen Anweisungen, die Sie bitte in Form eines Kommentars direkt nach der Frage beantworten.

In den Materialien finden Sie außerdem ein Hauptprogramm und ein Makefile. Durch `make` kompilieren Sie Ihr Programm. Im Hauptprogramm finden Sie einige Fragen A-D, die Sie bitte in Form eines Kommentars direkt nach der Frage beantworten. Durch `make test` verifizieren Sie die Korrektheit Ihrer Implementierung für einige Testfälle.

Punkteverteilung:

- Implementierung der beiden Funktionen: 3 Punkte
- bestandene Tests: 1 Punkt
- Beantwortung der Fragen 1-8: 2 Punkte
- Beantwortung der Fragen A-D: 1 Punkte

Zur Bearbeitung dieser Aufgabe sollten Sie die Abschnitte 1-6, 8, 9, 16 (Folien 254-256) (entsprechend der Tabelle in `pfn2_vorlesung_2020.html`) kennen.

Aufgabe 6.2 (6 Punkte)

In der fiktiven Welt Simul lebt die fiktive Bakterienart *Enpe completii*. Das Genom des Bakteriums enthält ein Gen *dolly*, das in zwei Varianten (Allelen) *dolly-0* und *dolly-1* vorkommen kann. Zur Vereinfachung der Beschreibung sprechen wir von Individuen vom Typ 0 (diese habe das Allel *dolly-0*) und Typ 1 (diese haben das Allel *dolly-1*).

Auf Simul läuft die Zeit in diskreten Zeiteinheiten. Wir sprechen daher von Generationen. In jeder Generation hat jedes *Enpe completii* Bakterium eine gewisse Wahrscheinlichkeit sich zu vermehren, d.h. sich in zwei identische Individuen des gleichen Typs zu teilen. Diese Wahrscheinlichkeit p_0 bzw. p_1 ist vom Typ abhängig (p_i für Typ $i \in \{0, 1\}$).

Da die Ressourcen auf Simul stark beschränkt sind, kann die Population nicht wachsen. Falls sich ein Individuum vermehrt, stirbt unmittelbar nach der Teilung ein zufällig ausgewähltes Individuum der Population. Dieses darf auch eines der zwei neu aus der Zellteilung entstandenen Individuen sein.

Die Populationsgröße ist also in allen Generationen konstant. Wenn das sich teilende Individuum vom Typ 0 ist und das sterbende Individuum vom Typ 1, dann erhöht sich die Anzahl der Individuen vom Typ 0 und die Anzahl der Individuen vom Typ 1 verringert sich. Das entsprechende gilt für den umgekehrten Fall.

Implementieren Sie in der Datei `simul_evolution.c` eine Funktion

```
void simulation_run(const size_t *num_individuals, const double *prob_divide,
                    size_t generations, FILE *logfp);
```

zur Simulation der Populationsgrößen entsprechend des obigen Modells und der Parameter

`num_individuals[0]`	initiale Anzahl der Individuen vom Typ 0,
`num_individuals[1]`	initiale Anzahl der Individuen vom Typ 1,
`prob_divide[0]`	Wahrscheinlichkeit, dass sich ein Individuum vom Typ 0 teilt,
`prob_divide[1]`	Wahrscheinlichkeit, dass sich ein Individuum vom Typ 1 teilt,
`generations`	maximale Anzahl der simulierten Generationen,
`logfp`	`FILE`-pointer zur Ausgabe in eine log-Datei.

Siehe hierzu auch die Datei `simul_evolution.h`, die in der C-Datei inkludiert werden muss.

Die Simulation soll beendet werden, sobald ausschliesslich Individuen von einem der beiden Typen vorhanden sind (d.h. die Population ist fixiert auf Individuen eines Typs). In einem solchen Fall wird eine Zeile der folgenden Form ausgegeben.

```
fixed:<typ><TAB>steps:<nsteps>
```

Hier kennzeichnen die spitzen Klammern Platzhalter für den Typ (0 oder 1) und die Anzahl der simulierten Generationen bis zur Fixierung auf den angegebenen Typ. Die spitzen Klammern sind nicht Teil der Ausgabe.

Falls nach `generations` Generationen keine Fixierung auf einen Typ vorliegt, wird eine Zeile der folgenden Form ausgegeben und die Simulation beendet:

```
simulation stopped after <nsteps> steps (0:<num_0>,1:<num_1>)
```

Auch hier sind die Platzhalter in spitzen Klammern angegeben. Falls `logfp` nicht `NULL` ist, werden über `logfp` nach der Kopfzeile `step<TAB>num_0<TAB>num_1` für jede Generation die aktuellen Werte für die Anzahl der Individuen vom Typ 0 bzw. 1 ausgegeben und zwar in folgendem Format:

```
nstep<TAB>num_0<TAB>num_1
```

z.B.:

```
0<TAB>100<TAB>100
1<TAB>99<TAB>101
2<TAB>98<TAB>102 ...
```

Sei `popsize` die Größe der Population. Die Population können Sie als Array der Größe `popsize` mit einem geeigneten Basistyp repräsentieren. Sie müssen in der genannten Funktion für eine Generation und für alle Individuen der Generation folgende Ereignisse simulieren:

1. Teilung des Individuums entsprechend der beiden vorgegebenen Wahrscheinlichkeiten p_0 und p_1. Dazu generiert man eine Zufallszahl r im Interval $[0, 1)$ und falls $p_i \geq r$ ist, teilt sich das Individuum vom Typ i in zwei Individuen des gleichen Typs i.
2. Falls eine Teilung erfolgt, bestimmt man durch Zufallsauswahl den Index des in diesem Schritt sterbenden Individuum unter `popsize+1` möglichen Individuen. Die `+1` ergibt sich aus der Tatsache, dass nach der Teilung zunächst ein zusätzliches Individuum vom gleichen Typ, wie das sich teilende Individuum, entsteht, bevor ein Individuum stirbt. Eine Zufallsauswahl unter n Werten erhält man durch Multiplikation von n mit einer Zufallszahl im Interval $[0, 1)$ und Interpretation des Produktes als ganze Zahl. Je nach zufälliger Auswahl des sterbenden Individuums ergibt sich möglicherweise eine Veränderung der Größen der Einzelpopulationen, über die man Buch führt.

Verwenden Sie zur Erzeugung der Zufallszahlen für die Simulation die Funktion `drand48()`. Der Aufruf von `srand48()` erfolgt bereits im Hauptprogramm.

Zur besseren Strukturierung des Programms implementieren Sie die Simulation einer neuen Generation in einer eigenen Funktion `simulation_step`, die in `simulation_run` wiederholt aufgerufen wird. `simulation_step` benötigt als Parameter einen Verweis auf das genannte Array, die Anzahl der Individuen vom Typ 0, die gesamte Anzahl aller Individuen, sowie einen Verweis auf ein Array mit zwei **`double`**-Werten, das die Wahrscheinlichkeiten p_0 und p_1 enthält. Die Funktion liefert als **`return`**-Wert die aktualisierte Anzahl der Individuen vom Typ 0 entsprechend der Werte im genannten Array zurück. In `simulation_run` erfolgt dann die Initialisierung des Arrays sowie die Erzeugung der Ausgaben entsprechend der oben beschriebenen Fälle.

In den Materialien zu dieser Aufgabe finden Sie ein Hauptprogramm mit einem Optionsparser, ein Makefile zum Kompilieren Ihres Programms sowie ein Shell-Skript mit Testfällen. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms für einige Testfälle.

Zur Bearbeitung dieser Aufgabe sollten Sie die Abschnitte 1-6, 8, 9, 14, 16 (Folien 260-285) (entsprechend der Tabelle in `pfn2_vorlesung_2020.html`) kennen.

Punkteverteilung:

- nachvollziehbare Implementierung von `simulation_step` entsprechend der Vorgaben: 4 Punkte
- nachvollziehbare Implementierung von `simulation_run` entsprechend der Vorgaben: 1 Punkt
- bestandene Tests: 1 Punkt

Bitte die Lösungen zu diesen Aufgaben bis zum 16.06.2020 um 18:00 Uhr an pfn2@zbh.uni-hamburg.de schicken.