Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2025/26

Abschlussaufgabe 2
Version 1.0

20 Punkte

Ausgabe:	02.03.2026, ca. 13:00 Uhr
Abgabestart:	16.03.2026, 12:00 Uhr
Abgabefrist:	31.03.2026, 06:00 Uhr

Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt jederzeit (auch nachträglich) zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Beachten Sie darüber hinaus die Richtlinien der Fakultät zum Verwenden von Generativer KI [1].

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich jederzeit zum Ausschluss der Erfolgskontrolle führen. Dies bedeutet ausdrücklich, dass auch nachträglich die Punktzahl reduziert werden kann.

Kommunikation und aktuelle Informationen

In unseren *FAQs*[2] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im Wiki[3].

In den *ILIAS-Foren* oder auf *Artemis* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki
[2] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ
[3] https://sdq.kastel.kit.edu/programmieren/

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[4] einsehen.

Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen verpflichtender Tests für eine erfolgreiche Abgabe von Abschlussaufgabe 2 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen außerdem zuerst die verpflichtenden Tests in Artemis bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 21*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io`, `java.util`, `java.util.regex`, `java.nio.file`, `java.nio.charset`, `java.time`, `java.time.format`.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.
- Achten Sie darauf, den bereitgestellten Satz von Checkstyle-Regeln einzuhalten.

Beachten Sie desweiteren die allgemeinen Bewertungsrichtlinien für das Aufgabenblatt. Diese finden Sie unter `https://sdq.kastel.kit.edu/programmieren/Hauptseite`

Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 16.03.2026, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 31.03.2026, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

[4] `https://artemis.cs.kit.edu/`

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

Wiederverwendung von Lösungen

Falls Sie für die Bearbeitung der Abschlussaufgaben oder Übungsblätter Beispiellösungen aus diesem Semester wiederverwenden, *müssen* Sie in die entsprechenden Klassen "Programmieren-Team" ins Autor-Tag eintragen. Dies ist nötig, um die Checkstyle-Kriterien zu erfüllen.

Prüfungsmodus in Artemis

Wenn Sie mit einer Abschlussaufgabe fertig sind, können Sie diese frühzeitig abgeben. Dazu dient Schaltfläche „Vorzeitig abgeben“. Nach der frühzeitigen Abgabe einer Abschlussaufgabe können Sie keine Änderungen an Ihrer Abgabe mehr vornehmen.

Aufgabe A: Skiroutenplaner

A.1 Einführung

In dieser Aufgabe implementieren Sie ein System zum Planen eines Skitages. Basierend auf Liften, Pisten und den Können und Präferenzen des Nutzers soll es den ideale Folge von Liften und Pisten bestimmen, so dass eine optimale und zeitlich machbare Route vorgeschlagen wird.

Im Folgenden werden zunächst die vorkommenden Konzepte kurz erklärt. Im Anschluss werden die geforderten Befehle, die das von Ihnen implementierte System unterstützen soll, mit beispielhaften Ein- und Ausgaben beschrieben.

A.2 Konzepte

Im Folgenden werden alle für diese Aufgabe relevanten Konzepte anhand eines Beispiel Skigebiets, dargestellt in Abb. A.1, erläutert.

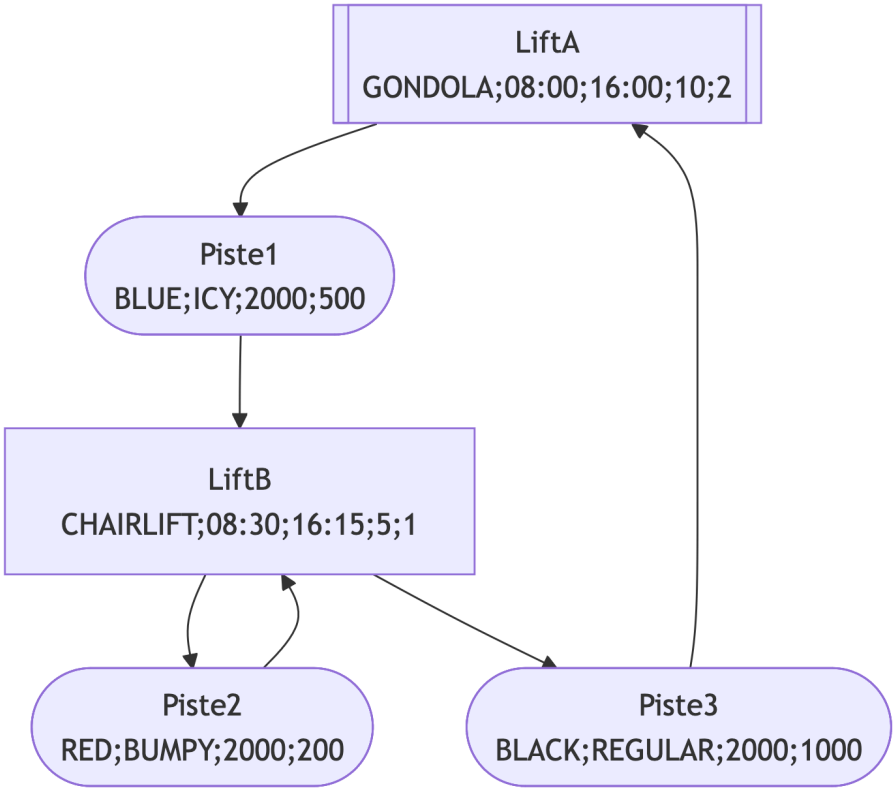

Abbildung A.1: Beispiel Skiegebiet mit 2 Liften und 3 Pisten

A.2.1 Skigebiet

Ein Skigebiet ist ein zusammenhängender gerichteter Graph, der aus Liften und Pisten besteht. Ein Gebiet enthält mindestens einen Lift (Talstation) und mindestens eine Piste. Jeder Lift und

jede Piste ist mit 1..*n* Liften und Pisten verbunden. Es können also Lifte auf Lifte und Pisten und Pisten folgen, sowie jegliche Kombination aus diesen. Es darf keine reflexiven Verbindungen geben, also kann ein Lift, oder eine Piste nicht mit sich selbst verbunden sein. Es darf symmetrische Verbindungen nur zwischen Liften und Pisten geben.

A.2.1.1 Lifte Lifte haben einen eindeutigen Bezeichner und können entweder Gondeln oder Sessellifte sein. Sie haben einen Startzeit und eine Endzeit, zwischen denen diese fahren. Darüber hinaus gibt es für jeden Lift eine Fahrtdauer und eine Anstehzeit, beides angegeben in Minuten. Ein Lift kann nur dann genommen werden, wenn Startzeit ≤ (aktuelle Zeit + Anstehzeit) ≤ Endzeit gilt. Sollte sie also vor Start des Liftes dort sein, müssen sie extra warten, sollten sie nach Ende des Liftes dort sein, können sie ihn **nicht** verwenden. Eine besondere Art von Liften sind Talstationen, an denen das Skigebiet betreten und verlassen werden kann. Jedes Skigebiet verfügt über mindestens eine Talstation. In Abbildung A.1 sind Liften mit kantigen Boxen dargestellt, wobei Talstation eine doppelte Linie haben. Beispielsweise ist `LiftA` eine Talstation mit Gondel, die zwischen 8:00 Uhr und 16:00 Uhr fährt, wobei die Fahrt 10 Minuten dauert und die Anstehzeit 2 Minuten beträgt.

A.2.1.2 Pisten Pisten haben einen eindeutigen Bezeichner und sind in Abbildung A.1 mit abgerundeten Boxen dargestellt. Pisten haben eine Schwierigkeit (Blau, Rot, Schwarz) und eine Oberfläche (Normal, Eisig, Buckelig). Darüber hinaus hat jede Piste eine Länge in Metern und eine Anzahl an Höhenmeter, die zwischen Start und Ende der Piste überwunden werden. Beispielsweise ist `Piste1` eine blaue Piste mit eisiger Fläche, die 2000 Meter lang ist und zwischen Start und Ende 500 Höhenmeter liegen.

A.2.1.3 Grammatik Wir definieren die Skigebiete in einer *Mermaid*[5]-Syntax, so dass diese Online visualisiert werden können. Das folgende Beispiel erzeugt das Skigebiet aus Abb. A.1.

example_graph.txt

```
graph
    LiftA[[LiftA<br/>GONDOLA;08:00;16:00;10;2]]
    Piste1([Piste1<br/>BLUE;ICY;2000;500])
    LiftB[LiftB<br/>CHAIRLIFT;08:30;16:15;5;1]
    Piste2([Piste2<br/>RED;BUMPY;2000;200])
    Piste3([Piste3<br/>BLACK;REGULAR;2000;1000])
    LiftA --> Piste1
    Piste1 --> LiftB
    LiftB --> Piste2
    LiftB --> Piste3
    Piste2 --> LiftB
    Piste3 --> LiftA
```

[5] `https://mermaid.live`

Das Format der Skigebiets-Datei ist durch folgende BNF-Grammatik [6] gegeben. Terminalsymbole sind im `Typewriter`-Stil geschrieben. Abweichend von der BNF sind die Nichtterminalsymbole *time*, *integer* sowie *id* jeweils durch einen regulären Ausdruck spezifiziert, der die Menge der gültigen Terminalsymbole ausdrückt.

BNF-Grammatik des Skigebiets

area ::= `graph` definitions edges
definitions ::= definition (definitions | ϵ)
edges ::= edge (edges | ϵ)
definition ::= regularLiftDefinition | transitLiftDefinition | slopeDefinition
regularLiftDefinition ::= id `[`id`<br/>`liftType `;` time `;` time `;` meter `;` meter`]`
transitLiftDefinition ::= id `[[`id`<br/>`liftType `;` time `;` time `;` meter `;` meter`]]`
slopeDefinition ::= id `(`[id`<br/>`difficulty `;` surface `;` meter `;` meter`)`
edge ::= id `-->` id
liftType ::= `GONDOLA` | `CHAIRLIFT`
difficulty ::= `BLUE` | `RED` | `BLACK`
surface ::= `REGULAR` | `ICY` | `BUMPY`
time ::= `[0-9][0-9]:[0-9][0-9]`
meter ::= `[0-9]+`
id ::= `[a-zA-Z0-9_]+`

A.2.2 Skifahrer

Ein weitere Zentrale Rolle für die Routenplanung spielen der Skifahrer und seine Fertigkeiten, Präferenzen und Tagesziele. Diese werden im Folgenden erläutert.

A.2.2.1 Fertigkeiten Die Fertigkeiten eines Skifahrers lassen sich in die Kategorien Anfänger, Fortgeschritten und Experte unterteilen. Anhand dieser Fertigkeit und den Eigenschaft einer Piste (siehe Absatz A.2.1.2) bestimmt sich die Zeit, die ein Skifahrer für das Bewältigen der Piste braucht.

Sei L die Länge der Piste in Metern, Δh der Unterschied an Höhenmetern, $M_{difficulty}$ der Schwierigkeitsmodifikator, $M_{surface}$ der Oberflächenmodifikator, M_{skill} der Fertigkeitsmodifikator, und r die Steigung.

Hierbei gilt:

[6] Backus-Naur-Form, siehe z.B. `http://de.wikipedia.org/wiki/Backus-Naur-Form`

$$M_{difficulty} = \begin{cases} 1.00, & \text{blue} \\ 1.15, & \text{red} \\ 1.35, & \text{black} \end{cases} \qquad M_{surface} = \begin{cases} 1.00, & \text{regular} \\ 1.20, & \text{bumpy} \\ 1.30, & \text{icy} \end{cases}$$

$$M_{skill} = \begin{cases} 1.35, & \text{beginner} \\ 1.10, & \text{intermediate} \\ 0.90, & \text{expert} \end{cases} \qquad r = \frac{\Delta h}{L}$$

Zusammen ergibt sich die Zeit T in Sekunden, die ein Skifahren für eine Piste braucht wie folgt:

$$T = \left\lceil \frac{L}{8} \cdot M_{\text{difficulty}} \cdot M_{\text{surface}} \cdot (1 + 2r) \cdot M_{\text{skill}} \right\rceil$$

A.2.2.2 Ziele Ein Skifahrer wählt für seinen Skitag genau **ein Ziel** sowie ein Zeitfenster $[t_{\text{start}}, t_{\text{end}}]$. Gesucht ist eine gültige Route durch das Skigebiet, die

- an einer Talstation startet,
- nur zulässige Lifte (unter Berücksichtigung von Öffnungszeiten und Anstehzeit) verwendet,
- alle Fahrzeiten von Liften und Pisten korrekt berücksichtigt,
- und spätestens zur Zeit t_{end} wieder an einer Talstation endet.

Für die Bewertung einer Route werden **ausschließlich** befahrene Pisten berücksichtigt. Je nach gewähltem Ziel ist der jeweilige Nutzenwert $U(R)$ zu maximieren:

Maximale Höhenmeter

$$U_{\Delta h}(R) = \sum_{s \in S(R)} \Delta h(s)$$

Maximale Strecke

$$U_{L}(R) = \sum_{s \in S(R)} L(s)$$

Maximale Anzahl an Pistenfahrten

$$U_{\#}(R) = |S(R)|$$

Maximale Anzahl unterschiedlicher Pisten

$$U_{\text{unique}}(R) = |\{s \mid s \in S(R)\}|$$

Dabei bezeichnet $S(R)$ die Folge aller in der Route R befahrenen Pisten. Mehrfaches Befahren einer Piste erhöht den Nutzen bei $U_{\Delta h}$, U_L und $U_\#$ entsprechend mehrfach. Beim Ziel U_{unique} trägt eine Piste genau dann zum Nutzen bei, wenn sie mindestens einmal befahren wurde; die Anzahl ihrer Befahrungen ist dabei unerheblich.

Hinweis: Die Problemstellung entspricht einer Routenoptimierung in einem gerichteten Graphen mit Zeitrestriktion. Geeignete Lösungsstrategien aus dem Bereich der Pfad- und Graphalgorithmen können hierfür angepasst werden.

A.2.2.3 Präferenzen Ein Skifahrer kann sowohl Vorlieben als auch Abneigungen hinsichtlich der Schwierigkeit und der Oberflächenbeschaffenheit von Pisten haben. Beispielsweise kann ein Skifahrer schwarze Pisten bevorzugen, jedoch eisige Oberflächen meiden.

Diese Präferenzen werden in dieser Aufgabe verwendet, um Gleichstände bei der Routenwahl aufzulösen. Wenn bei der Erstellung einer Route zwei Pisten für das gewählte Ziel gleichermaßen infrage kommen, wird jene Piste ausgewählt, die den Präferenzen des Fahrers stärker entspricht.

Für jede Übereinstimmung mit einer positiven Präferenz wird ein Wert von +1 vergeben, für jede Übereinstimmung mit einer negativen Präferenz ein Wert von −1. Die Summe dieser Werte wird für beide Pisten berechnet und miteinander verglichen. Gewählt wird die Piste mit dem höheren Gesamtwert.

Besteht weiterhin Gleichstand, so wird die Piste ausgewählt, deren Bezeichner lexikografisch vor dem Bezeichner der anderen liegt.

A.3 Hinweise zur Implementierung

Nach dem Start nimmt Ihr Programm über die Kommandozeile mittels der Standardeingabe Eingaben entgegen, die im Folgenden näher spezifiziert werden. Alle Befehle werden auf dem aktuellen Zustand des Programms ausgeführt. Nach Abarbeitung einer Eingabe wartet Ihr Programm auf weitere Eingaben, bis das Programm irgendwann durch die Eingabe der Zeichenfolge `quit` beendet wird.

Im Folgenden werden Eingabezeilen mit einer schließenden spitzen Klammer (`>`) gefolgt von einem Leerzeichen eingeleitet. Diese beiden Zeichen sind kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung von Ein- und Ausgabezeilen.

Achten Sie darauf, dass durch die Ausführung der folgenden Befehle die gegebenen semantischen und syntaktischen Spezifikationen nicht verletzt werden und geben Sie bei Verletzung der Spezifikationen immer eine aussagekräftige Fehlermeldung aus. Wenn die Benutzereingabe nicht dem vorgegebenen Format entspricht, ist auch eine Fehlermeldung auszugeben. Nach der Ausgabe einer Fehlermeldung soll das Programm wie erwartet fortfahren und wieder auf die nächste Eingabe warten. Die Fehlermeldung sollte so geformt sein, dass für den Benutzer erkenntlich ist, warum eine Eingabe abgelehnt wurde.

Da wir automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführen, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Setzen Sie nur die in der

Aufgabenstellung angegebenen Informationen um. Geben Sie auch keine zusätzlichen Informationen aus. Bei Fehlermeldungen dürfen Sie den englischsprachigen Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung muss aber mit `Error,` beginnen und darf keine Sonderzeichen, wie beispielsweise Zeilenumbrüche oder Umlaute, enthalten.

Wenn nicht anders angegeben, ist für Eingabe immer die Standardeingabe `System.in` zu verwenden. Wenn nicht anders angegeben, ist für Ausgaben immer die Standardausgabe `System.out` zu verwenden. Für Fehlermeldungen kann anstelle der Standardausgabe optional die Standardfehlerausgabe `System.err` verwendet werden. Weisen Sie diese Standardeingabe und -ausgabe niemals neu zu.

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen für Platzhalter stehen, die bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine spitzen Klammern. Vergleichen Sie hierzu auch den Beispielablauf.

A.4 Funktionalität

Im Folgenden wird die erforderliche Funktionalität des Systems beschrieben.

A.4.1 Platzhalter

Zusätzlich zu den Nichtterminalsymbolen aus Absatz A.2.1.3 sind folgende Platzhalter definiert:

A.4.1.1 `<goal>` Ziel der Route (`ALTITUDE`,`DISTANCE`,`NUMBER`,`UNIQUE`)

A.4.1.2 `<skill>` Fertigkeit des Fahrers (`BEGINNER`,`INTERMEDIATE`,`EXPERT`)

A.4.2 Befehle

A.4.2.1 Der `quit`-Befehl

Der parameterlose Befehl ermöglicht es das Programm vollständig zu beenden. Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen.

Eingabe `quit`

Beispielinteraktion

```
1 | > quit
```

A.4.2.2 Der `load area`-Befehl

Lädt eine spezifizierte Datei, parsed den Inhalt und validiert das enthaltene Skigebiet. Zu Testzwecken soll der Befehl den Inhalt der Datei Verbatim ausgeben. Wird der Befehl mehrfach ausgeführt, so wird, falls das neue Gebiet gültig ist, das alte überschrieben und die eventuell laufende Route abgebrochen.

Eingabe `load␣area␣<path>`

Ausgabe `Verbatim␣Inhalt␣der␣Datei`

Überprüfen Sie beim Validieren insbesondere ob:

- alle Bezeichner eindeutig sind
- das Skigebiet zusammenhängend ist
- das es mindestens eine Talstation und eine Piste gibt

Beispielinteraktion

```
> load area ./area.txt
graph
    LiftA[[LiftA<br/>GONDOLA;08:00;16:00;10;2]]
    Piste1([Piste1<br/>BLUE;ICY;2000;500])
    LiftB[LiftB<br/>CHAIRLIFT;08:30;16:15;5;1]
    Piste2([Piste2<br/>RED;BUMPY;2000;200])
    Piste3([Piste3<br/>BLACK;REGULAR;2000;1000])
    LiftA --> Piste1
    Piste1 --> LiftB
    LiftB --> Piste2
    LiftB --> Piste3
    Piste2 --> LiftB
    Piste3 --> LiftA
```

A.4.2.3 Der `list lifts`-Befehl

Gibt alle Lifte (und deren Eigenschaften) aufsteigend sortiert nach dem Bezeichner aus. Es wird ein Lift pro Zeile ausgegeben. Falls es sich um eine Talstation handelt, so kommt am Ende der Zeile zusätzlich die Ausgabe `TRANSIT`.

Eingabe `list␣lifts`

Ausgabe `<id>␣<liftType>␣<time>␣<time>␣<meter>␣<meter>␣[TRANSIT]`

Beispielinteraktion

```
> list lifts
LiftA GONDOLA 08:00 16:00 10 2 TRANSIT
LiftB CHAIRLIFT 08:30 16:15 5 1
```

A.4.2.4 Der `list slopes`-Befehl

Gibt alle Pisten (und deren Eigenschaften) aufsteigend sortiert nach dem Bezeichner aus. Es wird eine Piste pro Zeile ausgegeben.

Eingabe `list␣slopes`

Ausgabe `<id>␣<difficulty>␣<surface>␣<meter>␣<meter>`

Beispielinteraktion

```
1 > list slopes
2 Piste1 BLUE ICY 2000 500
3 Piste2 RED BUMPY 2000 200
4 Piste3 BLACK REGULAR 2000 1000
```

A.4.2.5 Der `set skill`-Befehl

Setzt die Fertigkeit des Skifahrers auf den Angegeben Wert.

Eingabe `set␣skill␣<skill>`

Beispielinteraktion

```
1 > set skill EXPERT
```

A.4.2.6 Der `set goal`-Befehl

Setzt das Ziel des Skifahrers auf den Angegeben Wert.

Eingabe `set␣goal␣<goal>`

Beispielinteraktion

```
1 > set goal DISTANCE
```

A.4.2.7 Der `like`-Befehl

Speichert die Vorliebe einer Schwierigkeit oder Oberfläche von Pisten.

Eingabe `like␣<difficulty>|<surface>`

Beispielinteraktion

```
1 > like BLUE
```

A.4.2.8 Der `dislike`-Befehl

Speichert die das Abneigung einer Schwierigkeit oder Oberfläche von Pisten.

Eingabe `dislike␣<difficulty>|<surface>`

Beispielinteraktion

```
1 > dislike BUMPY
```

A.4.2.9 Der `reset preference`-Befehl

Setzt alle Vorlieben des Skifahrers auf neutral zurück.

Eingabe `reset␣preferences`

Beispielinteraktion

```
1 > reset preferences
```

A.4.2.10 Der `plan`-Befehl

Plant eine neue Route im Zeitlichen Rahmen, die an der angegeben Talstation beginnt und endet. Erfordert, dass ein Skigebiet geladen ist, Ziel und Können festgesetzt wurden und momentan keine Route läuft. Ferner muss die Endzeit nach der Startzeit liegen. Sollte keine Route gefunden werden, da alle Lifte in der angegebenen Zeitspanne geschlossen sind, so soll ein Fehler ausgegeben werden.

Eingabe `plan␣<id>␣<time>␣<time>`

Ausgabe `route␣planned`

Beispielinteraktion

```
1 > plan LiftA 08:30 15:45
2 route planned
```

A.4.2.11 Der `abort`-Befehl

Falls gerade eine Route geplant ist oder schon bereits begonnen wurde, so wird diese Abgebrochen. Dies entfernt auch den Skifahrer aus dem Skigebiet.

Eingabe `abort`

Ausgabe `route␣aborted`

Beispielinteraktion

```
1 > abort
2 route aborted
```

A.4.2.12 Der `next`-Befehl

Falls gerade eine Route geplant ist oder schon bereits begonnen wurde, so wird der nächste Lift oder die nächste Piste ausgegeben.

Eingabe `next`

Ausgabe `<id>`

Beispielinteraktion

```
1 > next
2 LiftA
```

Sollte die Route beendet sein, so wird stattdessen `route finished!` ausgegeben.

Beispielinteraktion

```
1 > next
2 route finished!
```

A.4.2.13 Der `take`-Befehl

Darf nur **direkt** nach dem `next`-Befehl ausgeführt werden, wodurch der Fahrer den vorgeschlagenen Lift oder die vorgeschlagene Piste nimmt.

Eingabe `take`

Beispielinteraktion

```
1 > next
2 LiftA
3 > take
```

A.4.2.14 Der `alternative`-Befehl

Darf nur **direkt** nach dem `next`-Befehl ausgeführt werden. Falls der Fahrer mindestens die Talstation befahren, also die Route begonnen hat, wird versucht eine alternative für den nächsten Lift oder Piste zu berechnen. Hierbei wird die optimale neue Route bestimmt, die nicht den ursprünglich vorgeschlagenen nächsten Lift, oder Piste als nächsten Schritt enthält. Falls es eine alternative Route gibt, so erfolgt die Ausgabe mit dem vermiedenem nächsten Halt.

Eingabe `alternative`

Ausgabe `avoided␣<id>`

Beispielinteraktion

```
1 > next
2 Piste2
3 > alternative
4 avoided Piste2
5 > next
6 Piste3
```

Falls es keine alternative Route gibt, so soll stattdessen `no alternative found` ausgegeben werden.

Beispielinteraktion

```
1 > next
2 Piste1
3 > alternative
4 no alternative found
5 > next
6 Piste1
```

A.4.2.15 Der `show route`-Befehl

Falls gerade eine Route geplant ist oder schon bereits begonnen wurde, so wird die (verbleibende) Route ausgegeben. Der erste Bezeichner in dieser Liste entspricht der nächsten Piste oder dem nächsten Lift, siehe `next`-Befehl.

Eingabe `show␣route`

Ausgabe `<id>␣<id>␣...␣<id>`

Beispielinteraktion

```
1 > show route
2 LiftA Piste1 LiftB Piste3
```

A.4.2.16 Dynamisches Planen Die Befehle `set skill`, `set goal`, `like`, `dislike`, `reset preferences` können auch benutzt werden, nachdem eine Route geplant oder begonnen worden ist. In dem Fall wird die Route dynamisch neu berechnet. Sollte sich der Skifahrer schon auf der Route befinden, so ist die Startzeit der Route die aktuelle Zeit und der Startpunkt die aktuelle Position des Skifahrers.

A.5 Beispielinteraktion

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Die Eingabezeilen werden mit einer schließenden spitzen Klammer (`>`) gefolgt von einem Leerzeichen eingeleitet, diese beiden Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung zwischen Ein- und Ausgabezeilen.

Beispielinteraktion

```
> load area ./input/area.txt
graph
    LiftA[[LiftA<br/>GONDOLA;08:00;16:00;10;2]]
    Piste1([Piste1<br/>BLUE;ICY;2000;500])
    LiftB[LiftB<br/>CHAIRLIFT;08:30;16:15;5;1]
    Piste2([Piste2<br/>RED;BUMPY;2000;200])
    Piste3([Piste3<br/>BLACK;REGULAR;2000;1000])
    LiftA --> Piste1
    Piste1 --> LiftB
    LiftB --> Piste2
    LiftB --> Piste3
    Piste2 --> LiftB
    Piste3 --> LiftA
> set skill BEGINNER
> set goal DISTANCE
> like BLUE
> dislike BUMPY
> plan LiftA 11:00 12:30
route planned
> show route
LiftA Piste1 LiftB Piste3 LiftA Piste1 LiftB Piste3
> next
LiftA
> take
> next
Piste1
> take
> next
LiftB
> take
> next
Piste3
> take
> next
LiftA
> take
> next
Piste1
> take
> next
```

Beispielinteraktion

```
LiftB
> take
> next
Piste3
> take
> next
route finished!
> set skill EXPERT
> set goal ALTITUDE
> like BLACK
> like ICY
> plan LiftA 13:00 15:30
route planned
> show route
LiftA Piste1 LiftB Piste3 LiftA Piste1 LiftB Piste3 LiftA Piste1 LiftB
    Piste3 LiftA Piste1 LiftB Piste3
> next
LiftA
> take
> next
Piste1
> take
> next
LiftB
> take
> next
Piste3
> take
> next
LiftA
> take
> next
Piste1
> take
> next
LiftB
> take
> next
Piste3
> alternative
avoided Piste3
```

Beispielinteraktion

```
> show route
Piste2 LiftB Piste2 LiftB Piste3 LiftA Piste1 LiftB Piste3
> next
Piste2
> take
> next
LiftB
> take
> next
Piste2
> take
> next
LiftB
> take
> next
Piste3
> take
> next
LiftA
> take
> next
Piste1
> take
> next
LiftB
> take
> next
Piste3
> take
> next
route finished!
> quit
```