Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

Aufgabe 3 (Unendliche Datenstrukturen): (26 Punkte)

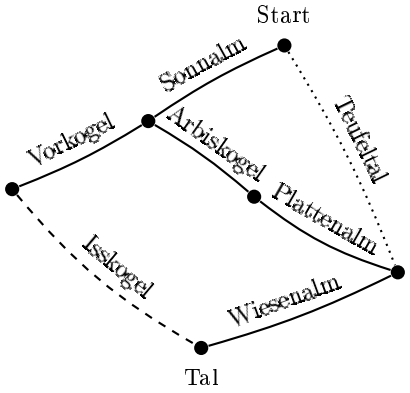
Geben Sie einen Haskell-Ausdruck an, der zu der aufsteigend sortierten Liste aller *guten Primzahlen* ausgewertet wird. Die n-te Primzahl p_n ist nach einer üblichen Definition genau dann eine gute Primzahl, wenn ihr Quadrat größer als das Produkt der benachbarten Primzahlen ist, d.h. $p_n^2 > p_{n-1} \cdot p_{n+1}$. Die erste gute Primzahl ist daher 5, denn es gilt $5^2 > 3 \cdot 7$.

Hinweise:

- Sie dürfen die vordefinierte Funktion `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]` verwenden. Diese berechnet eine Liste, in der an der i-ten Position das Ergebnis der Anwendung der übergebenen Funktion auf die beiden Elemente an der i-ten Position der beiden Eingabelisten steht. Sie ist also analog zur Funktion `zipLists` aus Aufgabe 6(f) von Übungsblatt 9.
- Sie dürfen die Funktion `primes` aus der Vorlesung verwenden.
- Sie können auch die vordefinierte Funktion `drop :: Int -> [a] -> [a]` benutzen. Der Aufruf `drop n xs` gibt die Liste zurück, die man erhält, wenn man die ersten `n` Elemente aus `xs` entfernt.
- Auch die beiden Funktionen `map` und `filter`, die aus der Vorlesung bekannt sind, dürfen hier benutzt werden.

Aufgabe 6 (Programmieren in Prolog): (4.5 + 2 + 5 + 4.5 + 4.5 + 4.5 + 6 = 31 Punkte)

In dieser Aufgabe soll ein Skipisten-Plan in Prolog modelliert und analysiert werden. In der Abbildung sind einfache (blaue) Pisten mit durchgehenden Linien, die einzige mittelschwere (rote) Piste mit einer gestrichelten und die einzige schwere (schwarze) Piste mit einer gepunkteten Linie eingezeichnet.



Nutzen Sie bei Ihrer Implementierung jeweils Prädikate aus den vorangegangenen Aufgabenteilen. Benutzen Sie **keine vordefinierten Prädikate**. Achten Sie auf die **korrekte Schreibweise** der Namen aus der Aufgabenstellung. Achten Sie bei Ihrer Implementierung darauf, dass diese allgemein sein soll und *nicht* nur für das angegebene Beispiel funktionieren soll. Es nutzt also nichts, in weiteren Fakten konkrete Fälle aus dem Beispiel zu sammeln.

a) Übertragen Sie die oben gegebenen Informationen in eine Wissensbasis für Prolog. Geben Sie hierzu Fakten und/oder Regeln für die Prädikatssymbole `blau`, `rot`, `schwarz`, `start` und `endetIn` an. Hierbei gilt `blau(X)`, falls `X` eine einfache Piste ist, `rot(X)`, falls `X` eine mittelschwere Piste ist, `schwarz(X)`, falls `X` eine schwere Piste ist, `start(X)`, falls die Piste `X` am Ausgangspunkt („Start“) beginnt, und `endetIn(X,Y)`, falls die Piste `X` an einem Punkt endet, an dem die Piste `Y` beginnt, oder falls `X` im Tal endet und `Y = tal` gilt.

b) Geben Sie eine Anfrage an das im ersten Aufgabenteil erstellte Programm an, mit der man herausfinden kann, welche Pisten am Startpunkt der Wiesenalm enden.

Hinweise:

- Durch wiederholte Eingabe von „`;`“ nach der ersten Antwort werden alle Antworten ausgegeben.

c) Schreiben Sie ein Prädikat `gleicherStartpunkt`, sodass `gleicherStartpunkt(X,Y)` genau dann gilt, wenn die Pisten `X` und `Y` am gleichen Punkt beginnen. In obiger Abbildung gilt das genau für die Pisten „Sonnalm“ und „Teufeltal“ bzw. „Vorkogel“ und „Arbiskogel“.

d) Schreiben Sie ein Prädikat `erreichbar`, sodass `erreichbar(X,Y)` genau dann gilt, wenn es einen Weg von der Piste `X` in Richtung Tal gibt, der über die Piste `Y` führt.

e) Schreiben Sie ein Prädikat `moeglicheSchlusspiste`, sodass `moeglicheSchlusspiste(X,S)` genau dann gilt, wenn es es einen Weg von der Piste `X` ins Tal gibt, der als Letztes über die Piste `S` führt.

f) Schreiben Sie ein Prädikat `treffpisten`, sodass `treffpisten(X,Y,T)` genau dann gilt, wenn die Piste `T` sowohl auf einem Weg von der Piste `X` in Richtung Tal als auch auf einem Weg von der Piste `Y` in Richtung Tal liegt.

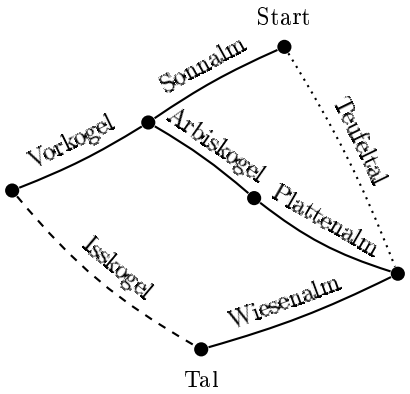
Hinweise:

- Das Tal ist keine Piste. Um diesen Fall abzufangen, können Sie mit dem zweistelligen Prädikat `\=` auf Ungleichheit prüfen.

g) Schreiben Sie ein Prädikat `anfaengerGeeignet`, sodass `anfaengerGeeignet(X)` genau dann gilt, wenn es einen Weg von `X` in Richtung Tal gibt, auf dem nur einfache Piste liegen, einschließlich der Piste `X`.

Aufgabe 9 (Prolog mit Listen und eigenen Datenstrukturen): (5 + 11 + 10 + 2.5 + 4.5 + 10 = 43 Punkte)

In dieser Aufgabe betrachten wir erneut den Skipisten-Plan aus Aufgabe 6. Wie dort sind die Kanten dieses Graphen bergabwärts gerichtet (d.h. die Sonnalm beginnt am Punkt Start und endet am Beginn des Vorkogels und des Arbiskogels). Einfache (blaue) Pisten sind mit durchgehenden Linien, die einzige mittelschwere (rote) Piste ist mit einer gestrichelten und die einzige schwere (schwarze) Piste ist mit einer gepunkteten Linie eingezeichnet.



Verwenden Sie in allen Teilaufgabe **keine** vordefinierten Prädikate. Nutzen Sie Prädikate, deren Implementierung in früheren Teilaufgaben gefordert wurde, falls dies sinnvoll ist. Ebenso sollten Sie die Prädikate `blau`, `rot`, `schwarz`, `start` und `endetIn` aus Aufgabe 6 benutzen. Gehen Sie davon aus, dass diese Prädikate bereits implementiert wurden.

a) Wie in Aufgabe 8(a) wollen wir Listen in Prolog durch Verwendung eines nullstelligen Funktionssymbols `nil` zur Repräsentation der leeren Liste und eines zweistelligen Funktionssymbols `cons` zur Repräsentation nicht-leerer Listen darstellen, wobei das erste Argument von `cons` der in dem aktuellen Listenelement gespeicherte Wert und das zweite Argument von `cons` die Restliste ist. Auf diese Art und Weise kann z.B. die Liste `[a,b,c]` durch den Term `cons(a, cons(b, cons(c, nil)))` dargestellt werden.

Implementieren Sie ein Prädikat `pathOfLength`, so dass `pathOfLength(P,L)` genau dann wahr ist, wenn `P` einen Pfad der Länge `L` vom Startpunkt einer Piste hinab ins Tal repräsentiert. Dazu müssen die folgenden beiden Bedingungen gelten:

- `P` ist eine mit Hilfe der Funktionssymbole `nil` und `cons` beschriebene Liste $[\mathtt{P}_1, \ldots, \mathtt{P}_n]$, wobei für $2 \leq i \leq n$ gilt, dass $\mathtt{P}_{i-1}$ eine Piste ist, die in $\mathtt{P}_i$ endet, und $\mathtt{P}_n$ das Tal ist.
- `L` ist eine natürliche Zahl in Peano-Notation (vgl. Aufgabe 7), die genau der Anzahl der Pisten in `P` entspricht.

Beispielsweise sollen `pathOfLength(cons(tal, nil), 0)` und

```
pathOfLength(cons(plattenalm, cons(wiesenalm, cons(tal, nil))), s(s(0)))
```

gelten, während

```
pathOfLength(cons(plattenalm, cons(wiesenalm, nil)), s(s(0)))
```

nicht gilt.

b) Implementieren Sie ein Prädikat `tourOfLength`, so dass `tourOfLength(T,L)` genau dann wahr ist, wenn `T` eine Tour der Länge `L` durch das Skigebiet repräsentiert. Dazu müssen die folgenden vier Bedingungen gelten:

(i) `T` ist eine mit Hilfe der Funktionssymbole `nil` und `cons` beschriebene Liste.

(ii) `T` beginnt im Tal und endet im Tal (d.h., das erste und das letzte Listenelement sind `tal`).

(iii) In `T` folgt nur dann ein Listenelement e_2 auf ein Listenelement e_1, wenn e_1 eine Piste ist, die in e_2 endet, oder wenn e_1 das Tal und e_2 eine der Startpisten ist. (Bei einer Tour kann also beliebig oft der Lift vom Tal zum Start benutzt werden.)

(iv) `L` ist eine natürliche Zahl in Peano-Notation, die genau der Anzahl der Pisten in `T` entspricht.

Ein Pfad `T` ist also eine *Tour*, falls er die Bedingungen (i) - (iii) erfüllt. Beispielsweise soll der Aufruf `tourOfLength(X,0)` die Antwort `X = cons(tal, nil)` und der Aufruf `tourOfLength(X,s(s(s(s(0)))))` unter anderem die Antwort

```
X =
    cons(tal,
        cons(teufeltal,
            cons(wiesenalm,
                cons(tal,
                    cons(teufeltal,
                        cons(wiesenalm,
                            cons(tal, nil)))))))
```

liefern.

Hinweise:

- Verwenden Sie das Prädikat `pathOfLength` aus der vorherigen Teilaufgabe und das in der Vorlesung vorgestellte Prädikat

  ```
  add(X,0,X).
  add(X,s(Y),s(Z)) :- add(X,Y,Z).
  ```

- Implementieren Sie ein Prädikat zum Aneinanderhängen von Listen analog zum Prädikat `append` für vordefinierte Listen aus Aufgabe 7(b).

c) Implementieren Sie ein Prädikat `partTour`, so dass `partTour(P,T)` genau dann wahr ist, wenn `P` und `T` Touren sind und `P` aus `T` dadurch entsteht, dass man vorne und hinten einen Teil der Tour `T` weglässt. Dieser weggelassene Teil kann auch leer sein.

d) Implementieren Sie ein Prädikat `convert`, so dass `convert(X,Y)` genau dann wahr ist, wenn

- `X` eine mit Hilfe der Funktionssymbole `nil` und `cons` beschriebene Liste ist und
- `Y` die gleiche Liste wie `X` beschreibt, dazu jedoch die vordefinierten Prolog-Listen nutzt.

Es soll also beispielsweise `convert(cons(tal, cons(wiesenalm, nil)), [tal, wiesenalm])` gelten.

e) Implementieren Sie ein Prädikat `enumerateTours`, so dass `enumerateTours(T)` in aufsteigender Reihenfolge alle möglichen Touren als vordefinierte Prolog-Liste sortiert nach ihrer Länge berechnet. Die ersten drei Antworten für `enumerateTours(T)` sollen also lauten

```
T = [tal] ;
T = [tal, teufeltal, wiesenalm, tal] ;
T = [tal, sonnalm, vorkogel, isskogel, tal]
```

Hinweise:

- Implementieren Sie ein Hilfsprädikat `enumerateTours`, so dass `enumerateTours(T,L)` alle Touren mit einer Länge von mindestens `L` berechnet und rufen Sie dieses mit `L = 0` auf.

f) Implementieren Sie ein Prädikat `tourRotSchwarz`, so dass `tourRotSchwarz(T,R,S)` genau dann wahr ist, wenn die Tour `T` genau `R` rote und `S` schwarze Pisten beinhaltet. Implementieren Sie das Prädikat so, dass im Falle konkreter Werte für `R` und `S` alle Touren mit der passenden Anzahl roter und schwarzer Pisten sortiert nach Länge in aufsteigender Reihenfolge ausgegeben werden. Beispielsweise soll der Aufruf `tourRotSchwarz(T, s(0), 0)` als erstes die Antwort `T = [tal, sonnalm, vorkogel, isskogel, tal]` liefern.