TRIES (40/40 points)

The data structure called *trie* is very convenient to represent a dictionary whose keys are strings. It is space-efficient way while providing a very fast lookup function.
See the page on WikiPedia.
In this exercise, we will implement such a data structure, assuming that we want to associate integers to the strings of the dictionary.
Let us define a trie using two mutually defined types (given in the prelude):

- `trie` which represents a trie, that is a tree whose root may contain an integer and whose children are indexed by characters ;
- `char_to_children` which implements the associative data structure whose keys are characters and whose values are trie (childrens).

As a trade-off between speed and memory consumption, we choose an associative list to represent the association between characters and children.
The prelude also gives examples of empty trie and of another one that contains the following pairs (key, value):
`[("A", 15); ("to", 7); ("tea", 3);("ted", 4); ("ten", 12); ("i", 11); ("in", 5); ("inn",`
.

1. Write a function `children_from_char : char_to_children -> char -> trie option` such that
    1. `children_from_char m c = Some t` if `(c, t)` is the first pair in `m` with `c` as a first component ;
    2. `children_from_char m c = None` if no such pair exists in `m`.
2. Write a function `update_children : char_to_children -> char -> trie -> char_to_children` such that
    1. `children_from_char (update_children m c t) c = Some t` ;
    2. `children_from_char (update_children m c t) c' = children_from_char m c'` for `c <> c'` ;
    3. If `children_from_char m c = Some t` then `List.length (update_children m c t') = List.length m`.
3. Write a function `lookup : trie -> string -> int option` such that `lookup trie w = Some i` if `i` is the value of the key `w` in `trie` and `lookup trie w = None` if `w` is not a key of `trie`.
   To look for a key in a trie, iterate over the characters of the key from left to right. Given the current character `c` and the current node of the trie `n`, look for the children `n` for character `c`. If such a children exists, continue with that trie and the remainder of the key. If no such children exists, the key is not in the trie. When the characters of the key are entirely consumed, look at the root of the current trie. If there is an integer, this is the value you are looking for. If there is no integer, the key not in the trie.
4. Write a function `insert : trie -> string -> int -> trie` such that `lookup (insert trie w k) w = Some k` and `lookup (insert trie w k) w' = lookup trie w'` for `w <> w'`.

THE GIVEN PRELUDE

```
let empty =
  Trie (None, [])

let example =
  Trie (None,
        [('i', Trie (Some 11,
                     [('n', Trie (Some 5, [('n', Trie (Some 9, []))]))]));
         ('t',
          Trie (None,
                [('e',
                  Trie (None,
                        [('n', Trie (Some 12, [])); ('d', Trie (Some 4, []));
                         ('a', Trie (Some 3, []))]));
                 ('o', Trie (Some 7, []))]));
         ('A', Trie (Some 15, []))])
```

YOUR OCAML ENVIRONMENT

```
1  let rec children_from_char m c = match m with
2    | [] -> None
3    | y :: ys -> match y with
4      | (a, b) ->  if a = c then Some b else children_from_char ys c
5  ;;
6
7  let rec update_children m c t = match m with
8    | [] -> (c, t) :: []
9    | x :: xs -> match x with
10     | (a, t') -> if a = c then (c, t) :: xs else x :: update_children xs c t
11 ;;
12
13 let rec lookup trie w =
14   match w, trie with
15   | "", Trie (e, _) -> e
16   | str, Trie (_, liste) ->
17       let result = children_from_char liste str.[0] in
18       match result with
19       | None -> None
20       | Some b -> lookup b (String.sub str 1 (String.length str - 1))
21 ;;
22
23 let rec insert trie w k =
24   match w, trie with
25   | "", Trie (e, liste) -> Trie (Some k, liste)
26   | str, Trie (e, liste) ->
27       let wbis = (String.sub str 1 (String.length str - 1)) in
28       let result = children_from_char liste str.[0] in
29       match result with
30       | None -> Trie (e, update_children liste str.[0] (insert empty wbis k))
31       | Some b -> Trie (e, update_children liste str.[0] (insert b wbis k))
32 ;;
33
```

Evaluate >

Switch >>

Typecheck

Reset Templ

Full-screen

Check & Sa

Exercise complete (click for details) — 40 pts

v Exercise 1: `children_from_char` — Completed, 10 pts

Found `children_from_char` with compatible type.

Computing

```
children_from_char
  [('m',
    Trie (None,
     [('j', Trie (None, [('a', Trie (Some 3, []))]));
      ('d', Trie (None, [('j', Trie (Some (-2), []))]))]));
   ('d', Trie (None, [('g', Trie (Some 1, []))]));
   ('j',
    Trie (Some 1,
     [('p', Trie (None, [('a', Trie (Some (-3), []))]));
      ('d', Trie (Some 3, []))]))]
  'g'
```

Correct value None — 1 pt

Computing

```
children_from_char
  [('s', Trie (None, [('j', Trie (None, [('d', Trie (Some 1, []))]))]));
   ('j', Trie (None, [('m', Trie (Some 1, []))]));
   ('a', Trie (None, [('d', Trie (Some 3, []))]))]
  'g'
```

Correct value None — 1 pt

Computing

```
children_from_char
  [('m', Trie (None, [('s', Trie (Some 2, []))]));
   ('p', Trie (None, [('s', Trie (None, [('d', Trie (Some 3, []))]))]));
   ('s',
    Trie (None,
     [('j', Trie (None, [('s', Trie (Some (-5), []))]));
      ('p', Trie (None, [('g', Trie (Some (-2), []))]))]));
```

```
Computing
  children_from_char
    [('p', Trie (None, [('p', Trie (Some 0, []))]));
     ('a', Trie (None, [('m', Trie (Some (-3), []))]));
     ('s', Trie (None, [('d', Trie (Some (-2), []))]))]
    'p'
```
Correct value (Some (Trie (None, [('p', Trie (Some 0, []))]))) 1 pt

```
Computing
  children_from_char
    [('m', Trie (None, [('s', Trie (Some (-1), []))]));
     ('p', Trie (None, [('m', Trie (Some 1, []))]))]
    'm'
```
Correct value (Some (Trie (None, [('s', Trie (Some (-1), []))]))) 1 pt

```
Computing
  children_from_char
    [('j', Trie (None, [('a', Trie (Some (-4), []))]));
     ('p',
      Trie (None,
       [('d', Trie (None, [('p', Trie (Some 0, []))]));
        ('j', Trie (Some 1, [])); ('s', Trie (Some (-2), []));
        ('p', Trie (None, [('a', Trie (Some (-2), []))]))]));
     ('d', Trie (None, [('d', Trie (Some (-3), []))]))]
    's'
```
Correct value None 1 pt

```
Computing
  children_from_char
    [('g', Trie (None, [('g', Trie (None, [('m', Trie (Some 4, []))]))]));
     ('a',
      Trie (None, [('a', Trie (Some 1, [])); ('g', Trie (Some (-1), []))]));
     ('p', Trie (None, [('j', Trie (Some 3, []))]));
     ('s', Trie (Some 2, [('g', Trie (None, [('g', Trie (Some 3, []))]))]))]
    'j'
```
Correct value None 1 pt

```
Computing
  children_from_char
    [('m',
      Trie (Some 0,
       [('m', Trie (None, [('a', Trie (Some 2, []))]));
        ('a', Trie (None, [('p', Trie (Some 3, []))]))]));
     ('g',
      Trie (Some (-1),
       [('s', Trie (Some 4, []));
        ('g', Trie (None, [('j', Trie (Some (-2), []))]))]))]
    'g'
```
Correct value 1 pt

```
  (Some
    (Trie (Some (-1),
      [('s', Trie (Some 4, []));
       ('g', Trie (None, [('j', Trie (Some (-2), []))]))])))
```

```
Computing
  children_from_char
    [('a', Trie (None, [('m', Trie (None, [('a', Trie (Some 4, []))]))]));
     ('s',
      Trie (None,
       [('a', Trie (Some 2, [])); ('j', Trie (Some (-1), []));
        ('s', Trie (None, [('g', Trie (Some (-3), []))]))]))]
    'j'
```
Correct value None 1 pt

```
Computing
  children_from_char
    [('j', Trie (None, [('g', Trie (None, [('s', Trie (Some (-3), []))]))]));
     ('s', Trie (None, [('a', Trie (Some 2, []))]));
     ('a', Trie (None, [('a', Trie (Some 3, []))]));
     ('d', Trie (None, [('j', Trie (None, [('g', Trie (Some (-3), []))]))]));
     ('m', Trie (None, [('j', Trie (Some 2, []))]))]
    'a'
```
Correct value (Some (Trie (None, [('a', Trie (Some 3, []))]))) 1 pt

v Exercise 2: `update_children` Completed, 10 pts

Found `update_children` with compatible type.

```
Computing
  update_children
    [('j', Trie (None, [('p', Trie (Some 1, []))]));
     ('a', Trie (None, [('a', Trie (None, [('m', Trie (Some (-5), []))]))]));
     ('p', Trie (None, [('s', Trie (Some (-2), []))]));
     ('s', Trie (None, [('s', Trie (None, [('a', Trie (Some (-5), []))]))]));
     ('g', Trie (Some 0, [('d', Trie (None, [('j', Trie (Some (-4), []))]))]))]
    'j'
    (Trie (None,
      [('s',
        Trie (None, [('m', Trie (Some 3, [])); ('a', Trie (Some (-3), []))]));
```

```
  [('j',
    Trie (None,
     [('s',
       Trie (None, [('m', Trie (Some 3, [])); ('a', Trie (Some (-3), [])) ]));
      ('j', Trie (Some 0, [('s', Trie (None, [('d', Trie (Some 4, [])) ]))]));
      ('p',
       Trie (Some (-3), [('s', Trie (None, [('s', Trie (Some (-1), [])) ]))]))]));
   ('a', Trie (None, [('a', Trie (None, [('m', Trie (Some (-5), [])) ]))]));
   ('p', Trie (None, [('s', Trie (Some (-2), [])) ]));
   ('s', Trie (None, [('s', Trie (None, [('a', Trie (Some (-5), [])) ]))]));
   ('g', Trie (Some 0, [('d', Trie (None, [('j', Trie (Some (-4), [])) ]))]))]
```

Computing

```
  update_children
    [('a',
      Trie (None,
       [('d', Trie (Some (-1), []));
        ('j', Trie (Some 1, [('p', Trie (Some 0, [])) ]))]));
     ('m',
      Trie (None,
       [('g', Trie (None, [('m', Trie (Some (-4), [])) ]));
        ('p', Trie (Some 1, [])); ('m', Trie (Some 1, [])) ]))]
    'm'
    (Trie (Some 4,
      [('m', Trie (None, [('d', Trie (Some 0, [])); ('j', Trie (Some 0, [])) ]));
       ('a', Trie (None, [('a', Trie (None, [('p', Trie (Some 3, [])) ]))]));
       ('j', Trie (None, [('p', Trie (None, [('g', Trie (Some (-1), [])) ]))]))]))
```

Correct value 1 pt

```
  [('a',
    Trie (None,
     [('d', Trie (Some (-1), []));
      ('j', Trie (Some 1, [('p', Trie (Some 0, [])) ]))]));
   ('m',
    Trie (Some 4,
     [('m', Trie (None, [('d', Trie (Some 0, [])); ('j', Trie (Some 0, [])) ]));
      ('a', Trie (None, [('a', Trie (None, [('p', Trie (Some 3, [])) ]))]));
      ('j', Trie (None, [('p', Trie (None, [('g', Trie (Some (-1), [])) ]))]))]))]
```

Computing

```
  update_children
    [('j', Trie (None, [('d', Trie (Some (-3), [])) ]));
     ('m', Trie (None, [('d', Trie (Some (-5), [])) ]));
     ('s',
      Trie (None,
       [('p', Trie (None, [('d', Trie (Some (-5), [])) ]));
        ('d', Trie (None, [('s', Trie (Some 0, [])) ]))]));
     ('a',
      Trie (None, [('j', Trie (Some (-3), [('p', Trie (Some (-5), [])) ]))]))]
    'g'
    (Trie (Some (-3),
      [('g', Trie (None, [('p', Trie (Some 3, [])) ]));
       ('m', Trie (None, [('d', Trie (None, [('m', Trie (Some 1, [])) ]))]));
       ('s', Trie (None, [('a', Trie (None, [('s', Trie (Some (-2), [])) ]))]));
       ('p',
        Trie (None,
         [('a', Trie (None, [('j', Trie (Some 4, [])) ]));
          ('m', Trie (Some 0, [])) ]))]))
```

Correct value 1 pt

```
  [('j', Trie (None, [('d', Trie (Some (-3), [])) ]));
   ('m', Trie (None, [('d', Trie (Some (-5), [])) ]));
   ('s',
    Trie (None,
     [('p', Trie (None, [('d', Trie (Some (-5), [])) ]));
      ('d', Trie (None, [('s', Trie (Some 0, [])) ]))]));
   ('a', Trie (None, [('j', Trie (Some (-3), [('p', Trie (Some (-5), [])) ]))]));
   ('g',
    Trie (Some (-3),
     [('g', Trie (None, [('p', Trie (Some 3, [])) ]));
      ('m', Trie (None, [('d', Trie (None, [('m', Trie (Some 1, [])) ]))]));
      ('s', Trie (None, [('a', Trie (None, [('s', Trie (Some (-2), [])) ]))]));
      ('p',
       Trie (None,
        [('a', Trie (None, [('j', Trie (Some 4, [])) ]));
         ('m', Trie (Some 0, [])) ]))]))]
```

Computing

```
  update_children
    [('d', Trie (None, [('p', Trie (Some 3, [])) ]));
     ('p', Trie (Some 4, [('m', Trie (None, [('j', Trie (Some 2, [])) ]))]))]
    'm'
    (Trie (Some (-3),
      [('j', Trie (None, [('a', Trie (Some 1, [])) ]));
       ('s', Trie (Some 4, [('a', Trie (None, [('m', Trie (Some 1, [])) ]))]))]))
```

Correct value 1 pt

```
  [('d', Trie (None, [('p', Trie (Some 3, [])) ]));
   ('p', Trie (Some 4, [('m', Trie (None, [('j', Trie (Some 2, [])) ]))]));
   ('m',
```

```
  update_children
    [('j', Trie (None, [('s', Trie (None, [('p', Trie (Some (-3), [])])])]));
     ('m', Trie (None, [('j', Trie (Some (-5), [])])])]
    'd'
    (Trie (Some (-5),
      [('a',
        Trie (None,
         [('d', Trie (None, [('j', Trie (Some (-3), [])])]));
          ('g', Trie (Some 1, [])])]));
       ('p',
        Trie (None,
         [('g', Trie (None, [('j', Trie (Some (-1), [])])]));
          ('j', Trie (None, [('a', Trie (Some (-2), [])])])]));
       ('g',
        Trie (Some (-4),
         [('m', Trie (Some (-1), []));
          ('p', Trie (None, [('d', Trie (Some (-5), [])])])])])]))
```

Correct value 1 pt

```
  [('j', Trie (None, [('s', Trie (None, [('p', Trie (Some (-3), [])])])]));
   ('m', Trie (None, [('j', Trie (Some (-5), [])])]));
   ('d',
    Trie (Some (-5),
     [('a',
       Trie (None,
        [('d', Trie (None, [('j', Trie (Some (-3), [])])]));
         ('g', Trie (Some 1, [])])]));
      ('p',
       Trie (None,
        [('g', Trie (None, [('j', Trie (Some (-1), [])])]));
         ('j', Trie (None, [('a', Trie (Some (-2), [])])])]));
      ('g',
       Trie (Some (-4),
        [('m', Trie (Some (-1), []));
         ('p', Trie (None, [('d', Trie (Some (-5), [])])])])])]
```

Computing

```
  update_children
    [('p', Trie (None, [('m', Trie (None, [('m', Trie (Some 3, [])])])]));
     ('m', Trie (None, [('m', Trie (None, [('m', Trie (Some (-4), [])])])])]
    'm'
    (Trie (Some (-5),
      [('a',
        Trie (None,
         [('s', Trie (Some (-3), []));
          ('j', Trie (None, [('d', Trie (Some 2, [])])])]));
       ('d', Trie (None, [('m', Trie (None, [('a', Trie (Some (-4), [])])])]));
       ('j', Trie (None, [('g', Trie (Some (-1), [])])]);
       ('s',
        Trie (None,
         [('j', Trie (None, [('g', Trie (Some 4, [])])]);
          ('s', Trie (None, [('g', Trie (Some (-4), [])])])])])]))
```

Correct value 1 pt

```
  [('p', Trie (None, [('m', Trie (None, [('m', Trie (Some 3, [])])])]));
   ('m',
    Trie (Some (-5),
     [('a',
       Trie (None,
        [('s', Trie (Some (-3), []));
         ('j', Trie (None, [('d', Trie (Some 2, [])])])]));
      ('d', Trie (None, [('m', Trie (None, [('a', Trie (Some (-4), [])])])]));
      ('j', Trie (None, [('g', Trie (Some (-1), [])])]);
      ('s',
       Trie (None,
        [('j', Trie (None, [('g', Trie (Some 4, [])])]);
         ('s', Trie (None, [('g', Trie (Some (-4), [])])])])])]
```

Computing

```
  update_children
    [('s', Trie (None, [('s', Trie (Some (-2), [])])]));
     ('p', Trie (None, [('s', Trie (None, [('p', Trie (Some (-2), [])])])]));
     ('a', Trie (None, [('j', Trie (Some 3, [])); ('p', Trie (Some 4, [])])])]
    'm'
    (Trie (Some 2,
      [('d', Trie (None, [('p', Trie (None, [('g', Trie (Some 1, [])])])]));
       ('a',
        Trie (None, [('a', Trie (Some (-2), [])); ('m', Trie (Some 3, [])])]));
       ('j',
        Trie (Some (-4), [('d', Trie (None, [('g', Trie (Some (-5), [])])])])])]))
```

Correct value 1 pt

```
  [('s', Trie (None, [('s', Trie (Some (-2), [])])]));
   ('p', Trie (None, [('s', Trie (None, [('p', Trie (Some (-2), [])])])]));
   ('a', Trie (None, [('j', Trie (Some 3, [])); ('p', Trie (Some 4, [])])]);
   ('m',
    Trie (Some 2,
     [('d', Trie (None, [('p', Trie (None, [('g', Trie (Some 1, [])])])]));
      ('a',
```

```
  update_children
    [('s', Trie (None, [('j', Trie (Some 3, []))]));
     ('g', Trie (None, [('m', Trie (Some 4, []))]));
     ('a', Trie (None, [('s', Trie (Some (-3), []))]));
     ('d', Trie (None, [('s', Trie (Some (-1), []))]));
     ('j',
      Trie (None,
       [('s', Trie (Some 4, []));
        ('p', Trie (None, [('j', Trie (Some 1, []))]))]))]
    'j'
    (Trie (Some (-1),
      [('m', Trie (None, [('p', Trie (Some (-4), []))]));
       ('g', Trie (None, [('p', Trie (Some (-1), []))]))]))
```

Correct value 1 pt

```
  [('s', Trie (None, [('j', Trie (Some 3, []))]));
   ('g', Trie (None, [('m', Trie (Some 4, []))]));
   ('a', Trie (None, [('s', Trie (Some (-3), []))]));
   ('d', Trie (None, [('s', Trie (Some (-1), []))]));
   ('j',
    Trie (Some (-1),
     [('m', Trie (None, [('p', Trie (Some (-4), []))]));
      ('g', Trie (None, [('p', Trie (Some (-1), []))]))]))]
```

Computing

```
  update_children
    [('j', Trie (None, [('s', Trie (None, [('g', Trie (Some 1, []))]))]));
     ('p', Trie (None, [('g', Trie (Some (-5), []))]));
     ('g', Trie (None, [('a', Trie (Some (-1), []))]));
     ('s', Trie (None, [('d', Trie (Some 1, []))]));
     ('a', Trie (None, [('m', Trie (Some (-4), []))]))]
    'j'
    (Trie (Some 4,
      [('a', Trie (None, [('j', Trie (Some (-3), []))]));
       ('d',
        Trie (None,
         [('a', Trie (None, [('d', Trie (Some 2, []))]));
          ('d', Trie (Some 2, []))]))]))
```

Correct value 1 pt

```
  [('j',
    Trie (Some 4,
     [('a', Trie (None, [('j', Trie (Some (-3), []))]));
      ('d',
       Trie (None,
        [('a', Trie (None, [('d', Trie (Some 2, []))]));
         ('d', Trie (Some 2, []))]))]));
   ('p', Trie (None, [('g', Trie (Some (-5), []))]));
   ('g', Trie (None, [('a', Trie (Some (-1), []))]));
   ('s', Trie (None, [('d', Trie (Some 1, []))]));
   ('a', Trie (None, [('m', Trie (Some (-4), []))]))]
```

Computing

```
  update_children
    [('a', Trie (None, [('g', Trie (None, [('m', Trie (Some 3, []))]))]));
     ('s', Trie (None, [('a', Trie (Some (-1), []))]));
     ('p',
      Trie (None, [('j', Trie (Some (-1), [])); ('m', Trie (Some (-3), []))]));
     ('d', Trie (None, [('j', Trie (None, [('s', Trie (Some (-3), []))]))]));
     ('j', Trie (None, [('j', Trie (Some 3, []))]))]
    'g'
    (Trie (Some 3,
      [('a', Trie (None, [('m', Trie (Some 1, []))]));
       ('g', Trie (None, [('d', Trie (Some (-3), []))]));
       ('j', Trie (None, [('j', Trie (Some (-3), []))]));
       ('s', Trie (None, [('d', Trie (None, [('p', Trie (Some (-3), []))]))]));
       ('m',
        Trie (Some (-1), [('p', Trie (None, [('j', Trie (Some 2, []))]))]))]))
```

Correct value 1 pt

```
  [('a', Trie (None, [('g', Trie (None, [('m', Trie (Some 3, []))]))]));
   ('s', Trie (None, [('a', Trie (Some (-1), []))]));
   ('p',
    Trie (None, [('j', Trie (Some (-1), [])); ('m', Trie (Some (-3), []))]));
   ('d', Trie (None, [('j', Trie (None, [('s', Trie (Some (-3), []))]))]));
   ('j', Trie (None, [('j', Trie (Some 3, []))]));
   ('g',
    Trie (Some 3,
     [('a', Trie (None, [('m', Trie (Some 1, []))]));
      ('g', Trie (None, [('d', Trie (Some (-3), []))]));
      ('j', Trie (None, [('j', Trie (Some (-3), []))]));
      ('s', Trie (None, [('d', Trie (None, [('p', Trie (Some (-3), []))]))]));
      ('m', Trie (Some (-1), [('p', Trie (None, [('j', Trie (Some 2, []))]))]))]))]
```

∨ Exercise 3: `lookup` Completed, 10 pts

Found `lookup` with compatible type.

Computing

```
  lookup
    (Trie (Some 2,
      [('d', Trie (None, [('s', Trie (None, [('p', Trie (Some (-1), []))]))]));
       ('s', Trie (None, [('m', Trie (None, [('a', Trie (Some 1, []))]))]));
```

```
          ('s', Trie (None, [('a', Trie (Some (-5), [])])])])])))
    "ga"
```
Correct value (Some (-4)) 1 pt
```
Computing
  lookup
    (Trie (Some (-4),
      [('s', Trie (None, [('d', Trie (None, [('p', Trie (Some 0, [])])])]));
       ('d', Trie (None, [('p', Trie (Some (-1), [])])]));
       ('m',
        Trie (Some (-5), [('g', Trie (None, [('g', Trie (Some (-3), [])])])])])]))
    "jjm"
```
Correct value None 1 pt
```
Computing
  lookup
    (Trie (Some 3,
      [('g', Trie (None, [('a', Trie (Some 0, [])])]));
       ('m',
        Trie (Some 0,
         [('p', Trie (Some 2, []));
          ('s', Trie (None, [('p', Trie (Some (-3), [])])])])])]))
    "ga"
```
Correct value (Some 0) 1 pt
```
Computing
  lookup
    (Trie (Some (-1),
      [('p', Trie (None, [('s', Trie (Some 4, [])])]));
       ('a', Trie (None, [('a', Trie (None, [('p', Trie (Some (-4), [])])])]));
       ('m', Trie (None, [('d', Trie (Some (-3), [])])])]))
    "asp"
```
Correct value None 1 pt
```
Computing
  lookup
    (Trie (Some 2,
      [('g', Trie (None, [('j', Trie (None, [('j', Trie (Some 1, [])])])])]));
       ('s', Trie (None, [('d', Trie (None, [('a', Trie (Some (-5), [])])])])]));
       ('a', Trie (None, [('a', Trie (Some (-1), [])])])]))
    "jp"
```
Correct value None 1 pt
```
Computing
  lookup
    (Trie (Some 2,
      [('s', Trie (None, [('g', Trie (None, [('j', Trie (Some 1, [])])])])]));
       ('m', Trie (None, [('p', Trie (Some 1, [])])]));
       ('g', Trie (None, [('a', Trie (Some 4, [])])]));
       ('d', Trie (None, [('d', Trie (None, [('a', Trie (Some (-5), [])])])])]));
       ('p',
        Trie (Some 0,
         [('j', Trie (Some (-3), []));
          ('s', Trie (None, [('j', Trie (Some 4, [])])])])])]))
    "sgj"
```
Correct value (Some 1) 1 pt
```
Computing
  lookup
    (Trie (Some (-4),
      [('g', Trie (None, [('d', Trie (None, [('a', Trie (Some 1, [])])])])]));
       ('s', Trie (None, [('d', Trie (None, [('s', Trie (Some 3, [])])])])]));
       ('a', Trie (None, [('g', Trie (Some 1, [])])]));
       ('m', Trie (None, [('d', Trie (Some 2, [])); ('a', Trie (Some 4, [])])])]))
    "mp"
```
Correct value None 1 pt
```
Computing
  lookup
    (Trie (Some 2,
      [('a', Trie (None, [('d', Trie (None, [('g', Trie (Some (-1), [])])])])]));
       ('p',
        Trie (Some 0,
         [('m', Trie (Some 1, []));
          ('g', Trie (None, [('s', Trie (Some (-2), [])])])])])]))
    "pgs"
```
Correct value (Some (-2)) 1 pt
```
Computing
  lookup
    (Trie (Some 4,
      [('d', Trie (None, [('m', Trie (Some 2, [])])]));
       ('j', Trie (None, [('d', Trie (Some 4, [])])])]))
    "dm"
```
Correct value (Some 2) 1 pt
```
Computing
  lookup
    (Trie (Some (-3),
      [('m', Trie (None, [('m', Trie (None, [('a', Trie (Some (-5), [])])])])]));
```

Exercise 4: `insert` — Completed, 10 pts

Found `insert` with compatible type.

```
Computing
  insert
    (Trie (Some 3,
      [('a', Trie (None, [('p', Trie (Some 3, []))]));
       ('m', Trie (None, [('a', Trie (Some 4, []))]));
       ('g',
        Trie (Some (-4), [('m', Trie (None, [('s', Trie (Some 2, []))]))]))]))
    "ag"
    1
```

Correct value — 1 pt

```
  (Trie (Some 3,
    [('a', Trie (None, [('p', Trie (Some 3, [])); ('g', Trie (Some 1, []))]));
     ('m', Trie (None, [('a', Trie (Some 4, []))]));
     ('g', Trie (Some (-4), [('m', Trie (None, [('s', Trie (Some 2, []))]))]))]))
```

```
Computing
  insert
    (Trie (Some 4,
      [('j',
        Trie (None, [('d', Trie (Some (-2), [])); ('g', Trie (Some (-4), []))]));
       ('g', Trie (None, [('a', Trie (Some (-3), []))]));
       ('m', Trie (None, [('g', Trie (Some (-5), []))]))]))
    "jd"
    0
```

Correct value — 1 pt

```
  (Trie (Some 4,
    [('j',
      Trie (None, [('d', Trie (Some 0, [])); ('g', Trie (Some (-4), []))]));
     ('g', Trie (None, [('a', Trie (Some (-3), []))]));
     ('m', Trie (None, [('g', Trie (Some (-5), []))]))]))
```

```
Computing
  insert
    (Trie (Some (-5),
      [('s', Trie (None, [('p', Trie (Some (-2), []))]));
       ('j', Trie (None, [('p', Trie (None, [('g', Trie (Some 3, []))]))]));
       ('g', Trie (None, [('j', Trie (Some (-5), []))]));
       ('d', Trie (None, [('s', Trie (None, [('d', Trie (Some 0, []))]))]))]))
    "gmm"
    1
```

Correct value — 1 pt

```
  (Trie (Some (-5),
    [('s', Trie (None, [('p', Trie (Some (-2), []))]));
     ('j', Trie (None, [('p', Trie (None, [('g', Trie (Some 3, []))]))]));
     ('g',
      Trie (None,
       [('j', Trie (Some (-5), []));
        ('m', Trie (None, [('m', Trie (Some 1, []))]))]));
     ('d', Trie (None, [('s', Trie (None, [('d', Trie (Some 0, []))]))]))]))
```

```
Computing
  insert
    (Trie (Some (-5),
      [('m',
        Trie (None,
         [('j', Trie (None, [('s', Trie (Some 2, []))]));
          ('a', Trie (None, [('d', Trie (Some (-4), []))]))]));
       ('g',
        Trie (None,
         [('m', Trie (Some (-5), []));
          ('g', Trie (None, [('j', Trie (Some (-2), []))]))]));
       ('s',
        Trie (Some (-1), [('a', Trie (None, [('g', Trie (Some 1, []))]))]))]))
    "sag"
    3
```

Correct value — 1 pt

```
  (Trie (Some (-5),
    [('m',
      Trie (None,
       [('j', Trie (None, [('s', Trie (Some 2, []))]));
        ('a', Trie (None, [('d', Trie (Some (-4), []))]))]));
     ('g',
      Trie (None,
       [('m', Trie (Some (-5), []));
        ('g', Trie (None, [('j', Trie (Some (-2), []))]))]));
     ('s', Trie (Some (-1), [('a', Trie (None, [('g', Trie (Some 3, []))]))]))]))
```

```
Computing
  insert
    (Trie (Some 4,
      [('p',
        Trie (Some 3,
         [('d', Trie (None, [('j', Trie (Some (-5), []))]));
          ('j', Trie (None, [('d', Trie (Some (-5), []))]))]))]))
    ""
    -4
```

```
      [('d', Trie (None, [('j', Trie (Some (-5), []))]));
       ('j', Trie (None, [('d', Trie (Some (-5), [])])))]))))
Computing
  insert
    (Trie (Some (-2),
      [('g', Trie (None, [('s', Trie (None, [('p', Trie (Some (-3), [])])))]));
       ('s',
        Trie (None,
         [('g', Trie (None, [('p', Trie (Some 0, [])))]));
          ('d', Trie (Some (-4), []))]));
       ('m',
        Trie (Some (-3),
         [('m', Trie (None, [('a', Trie (Some (-1), []))]));
          ('g', Trie (None, [('d', Trie (Some (-2), [])])))]))]))
    "dds"
    -1
```

Correct value 1 pt

```
  (Trie (Some (-2),
    [('g', Trie (None, [('s', Trie (None, [('p', Trie (Some (-3), [])])))]));
     ('s',
      Trie (None,
       [('g', Trie (None, [('p', Trie (Some 0, [])))]);
        ('d', Trie (Some (-4), []))]));
     ('m',
      Trie (Some (-3),
       [('m', Trie (None, [('a', Trie (Some (-1), []))]));
        ('g', Trie (None, [('d', Trie (Some (-2), [])])))]));
     ('d', Trie (None, [('d', Trie (None, [('s', Trie (Some (-1), [])])))]))]))
```

```
Computing
  insert
    (Trie (Some (-4),
      [('p', Trie (None, [('j', Trie (Some (-4), []))]));
       ('m', Trie (None, [('m', Trie (None, [('p', Trie (Some (-1), [])])))]))]))
    "psa"
    -4
```

Correct value 1 pt

```
  (Trie (Some (-4),
    [('p',
      Trie (None,
       [('j', Trie (Some (-4), []));
        ('s', Trie (None, [('a', Trie (Some (-4), [])])))]));
     ('m', Trie (None, [('m', Trie (None, [('p', Trie (Some (-1), [])])))]))]))
```

```
Computing
  insert
    (Trie (Some (-1),
      [('d', Trie (None, [('j', Trie (None, [('a', Trie (Some 4, [])])))]));
       ('g',
        Trie (None,
         [('j', Trie (None, [('g', Trie (Some (-5), []))]));
          ('g', Trie (None, [('j', Trie (Some 3, [])])))]));
       ('s', Trie (None, [('p', Trie (None, [('a', Trie (Some 2, [])])))]))]))
    "spa"
    -3
```

Correct value 1 pt

```
  (Trie (Some (-1),
    [('d', Trie (None, [('j', Trie (None, [('a', Trie (Some 4, [])])))]));
     ('g',
      Trie (None,
       [('j', Trie (None, [('g', Trie (Some (-5), []))]));
        ('g', Trie (None, [('j', Trie (Some 3, [])])))]));
     ('s', Trie (None, [('p', Trie (None, [('a', Trie (Some (-3), [])])))]))]))
```

```
Computing
  insert
    (Trie (Some 2,
      [('p',
        Trie (None, [('p', Trie (Some 1, [])); ('j', Trie (Some (-1), []))]));
       ('a',
        Trie (Some 3, [('a', Trie (None, [('s', Trie (Some (-1), [])])))]))]))
    "pj"
    -2
```

Correct value 1 pt

```
  (Trie (Some 2,
    [('p',
      Trie (None, [('p', Trie (Some 1, [])); ('j', Trie (Some (-2), []))]));
     ('a', Trie (Some 3, [('a', Trie (None, [('s', Trie (Some (-1), [])])))]))]))
```

```
Computing
  insert
    (Trie (Some 4,
      [('g', Trie (None, [('g', Trie (Some 1, []))]));
       ('a', Trie (None, [('d', Trie (None, [('g', Trie (Some 1, [])])))]));
       ('p',
        Trie (Some (-3),
         [('a', Trie (Some (-5), [('p', Trie (Some (-5), [])])))]))]))
    "pd"
    -5
```

Correct value 1 pt

```
Trie (Some (-3),
  [('a', Trie (Some (-5), [('p', Trie (Some (-5), []))]));
   ('d', Trie (Some (-5), []))]))]))
```