Homework 7: Subsets
Due: 11:59 PM, Oct 23, 2018

Contents

1  Bookends (Practice)  2
2  Subsets  3
3  Subset Sum  4
4  k-Subsets  5
5  k-Subset Sum  5

Objectives

By the end of this homework you will be able to:

1. write recursive combinatorial procedures
2. write recursive search procedures

How to Hand In

For this (and all) homework assignments, you should hand in answers for all the non-practice questions. For this homework specifically, this entails answering the Subsets, Subset Sum, k-Subsets, and k-Subset Sum questions

In order to hand in your solutions to these problems, they must be stored in appropriately-named files. In particular, each should be named for the corresponding problem, as follows (e.g., subsets.rkt corresponds to Subsets):

- README.txt
- subsets.rkt
- subset-sum.rkt
- ksubsets.rkt
- ksubset-sum.rkt

For this assignment, all files you turn in that contain code must be Racket files, so they must end with extension .rkt
For this and every future assignment, you should also have a README.txt file whose first line contains only your Banner ID, and optionally with a message to the person grading explaining anything peculiar about the handin. For example:

README.txt:
B01234567
There’s nothing to say except that I’m turning in some files plus this README the way the instructions say that I should.

To hand in your solutions to these problems, you must upload them to Gradescope. Do not zip or compress them. If you re-submit your homework, you must re-submit all files. If you choose to also store these files on department machines, all your solution files should reside in your ~/course/cs0170/homeworks/hw07 directory.

Practice

1 Bookends (Practice)

Task: Write the procedure bookends, which consumes a list of data, alod, and produces a sublist of alod of length at least 2 whose first and last elements are equal to each other. If no such sublist exists, bookends produces the empty list.

Examples:

(bookends (quote (1 2 -3 4 2 0)))
=> (2 -3 4 2)

(bookends (quote (a b b c)))
=> (b b)

(bookends (quote (a)))
=> empty

Problems

Introduction

A set X is a collection of objects in no particular order, and with no repetition. A subset of the set X is a set containing only elements of X, but not necessarily all of its elements. Note that a subset might contain nothing at all.

For example, \{1, 2, 3\} is a set, and its subsets are:

\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}

In this homework, you will use lists to represent sets. The set \{1, 2, 3\} can be represented by the list (quote (1 2 3)) or the list (quote (3 1 2)), or any of the other 4 combinations. The set of all subsets of \{1, 2, 3\} can be represented by a list of lists as follows:
(define subsets-123
  (quote ((())
    (2) (3)
    (1 2) (2 3) (1 3)
    (1 2 3)))))

Remember, sets don’t necessarily have to be of integers! For instance, the set of all subsets of
(quote (a b c)) can be represented in a similar manner as above.

**Note:** Remember that the empty list is a proper subset of every set and should be included in the
list of subsets that you generate.

**Note:** As with all homeworks, you should follow the design recipe for each problem.

**Testing (important!)**

We’ve substantially emphasized testing in this course — specifically, you’ve been writing unit tests.
That is, testing that individual procedure work as you’d expect.

We’ve also recommended you write tests before starting the coding portion. Imagine a program
that for a given input should return 1.235325. Say you write the program, test it after writing it,
and it returns 1.235326. It’s very easy to say - “that looks right to me.” It’s important to write test
cases before coding, so that you can think about exactly what your procedure should output.

This strategy has a flaw, and that’s the fact that sometimes the exact output of our program may
depend on implementation details. Consider the set of subsets of \{1, 2\}:

\{\}, \{1\}, \{2\}, \{1, 2\}

In Racket, we could represent this as (quote ((()) (1) (2) (1 2))). However, with sets, order
doesn’t matter. That representation is no more correct than (quote ((2 1) (2) (1) ()))).
Before writing your procedure, it’s hard to know which ordering will be produced.

What we could instead do, is write a procedure that checks that two sets of subsets are equiva-
 lent so that regardless of your percise implementation, your test cases will prove correctness (or
incorrectness).

When writing a test case for this homework, you can write something like this, where (subsets ....)
is a call to the procedure you are writing, and expected-answer is the set of sets you’re expecting
as a result.

(check-expect (set-of-sets-equal? (subsets ....) expected-answer) true)

This requires a procedure (set-of-sets-equal? sets1 sets2), which we will provide a
stencil for. **You can download the stencil by clicking here**, and fill in the areas where
there is an ellipsis. You should include this code in both subsets.rkt and ksubsets.rkt
- and use it to test both of those problems as both procedures produce a set of sets as
solutions.
2 Subsets

Task: Write a procedure, subsets, which consumes a set, set, represented as a list, and produces the set of all subsets of set.

Note: The order of the subsets produced need not match the order in our examples.

Examples:

**(subsets (quote (-3 4 -5))**

=> (() (-3) (4) (-5) (-3 4) (4 -5) (-3 -5) (-3 4 -5))

**(subsets (quote (a b))**

=> (() (a) (b) (a b))

**(subsets (quote (1))**

=> (() (1))

**(subsets empty)**

=> ()

Just for Fun: Try out your procedure on sets of length 0 through 10.

3 Subset Sum

Task: Write a procedure, subset-sum, which consumes a set of integers, weights, and a target weight, target, and produces true if there exists a subset of weights whose elements sum to target, and false otherwise.

Note: Each weight and the target can be positive, negative, or zero. Furthermore, the empty set has weight 0.

Hint: What should \( (\text{subset-sum} \ (\text{quote} \ (1 \ 2 \ 3 \ 4)) \ 0) \) return?

Examples:

**(subset-sum (quote (0 2 -3 4)) 1)**

=> #true

**(subset-sum (quote (1 2 3 4)) -17)**

=> #false

**(subset-sum (quote (1 17 3 4)) 21)**

=> #true

Hint: One solution to this problem is the following:

```scheme
(define (slow-subset-sum weights target)
  (member? target (map lambda (alon) (foldr + 0 alon)) (subsets weights)))
```

This solution first computes all the subsets of weights, then sums all these subsets, and finally checks to see whether target is equal to any of the resulting sums. The disadvantage of this solution
is that it wastes a whole lot of space computing all subsets, before checking to see whether any of
them sum to \texttt{target}. There is a “better” solution, one that does not use \texttt{subsets} as a helper, and
does not require as much space\footnote{We’ve only used operation-counting as a way to measure performance so far, but the amount of memory a procedure uses also matters in some cases. For our purposes, we can measure this as a combination of two things: (i) how deep our nested function-calling goes (do you have to recur 10 times or 100 times?), and (ii) The maximum, over the execution of a program, of sum of the lengths of all lists in the program at any moment. When you compute all subsets of an \texttt{n}-element list, you’ve got a total of \(2^n\) subsets whose average length is about \(\frac{n}{2}\), so your “memory use” looks like \(\frac{n}{2}2^n\). By contrast, if you compute and evaluate each subset and then move on to the next, you never have a list with more than \texttt{n} elements. Other approaches can have similar impact on space-use.} as \texttt{slow-subset-sum}. The idea underlying the better algorithm is
similar to that used to compute \texttt{subsets}: consider whether each weight in \texttt{weights} is part of a
subset that sums to \texttt{target}, or not.

\textbf{Just for Fun:} Try out your procedure on sets of size 10, 11, 12, and 20. Estimate how long it
would take on a list of size 40.

\section{\(k\)-Subsets}

\textbf{Task:} Write a procedure, \texttt{\(k\)-subsets}, which consumes a set, \texttt{set}, and a natural number, \(k\), and
produces a list of the subsets of \texttt{set} that are of size \(k\).

\textbf{Note:} As in \texttt{subsets}, the order of the subsets produced need not match the order in our examples.

Note also that \(k\) is a natural number; as such, it cannot be negative, but it can be very large.

\textbf{Examples:}

\begin{verbatim}
(k-subsets (quote (1 2)) 0) => ()

(k-subsets (quote (1 2)) 1) => ((1) (2))

(k-subsets (quote (-1 -2 3)) 3) => ((-1 -2 3))

(k-subsets (quote (2 3 4)) 9) => ()

(k-subsets (quote (1 2 3 4 5)) 3) => ((3 4 5) (2 4 5) (2 3 5) (2 3 4)
(1 4 5) (1 3 5) (1 3 4) (1 2 5)
(1 2 4) (1 2 3))
\end{verbatim}

In the next problem, we refer to these subsets as \(k\)-element subsets.

\textbf{Hint:} One way to solve this problem is to generate all the subsets of \texttt{set}, and then to filter out
those of size \(k\). This is not acceptable; for small \(k\), it wastes a large amount of time and space, as
the majority of the subsets of \texttt{set} do not have \(k\) elements.

\textbf{More Hints:}

\begin{itemize}
  \item If \(k\) is zero, then \texttt{set} has only one subset of size \(k\) — the empty set.
\end{itemize}
• Otherwise, if set is empty (and k is not equal to 0), then set has no subsets of size k.

• Otherwise (i.e., if set is not empty and k is not equal to 0), there are at least two ways to construct subsets of set of size k. We leave it to you to discover exactly how.

Once again, your solution should not use subsets as a helper.

5 k-Subset Sum

Task: Write a procedure, k-subset-sum, which consumes a set of integers, weights, a natural number, k, and a target weight, target, and produces true if there exists a k-element subset of weights whose elements sum to target, and false otherwise.

Note: Once again, as in subset-sum, each weight and the target weight can be positive, negative, or zero. Also, as in k-subsets, k cannot be negative.

Examples:

(k-subset-sum (quote (1 2 3 4)) 2 8)
=> #false

(k-subset-sum (quote (1 2 3 4)) 3 8)
=> #true

(k-subset-sum (quote (1 2 3 4)) 4 11)
=> #false

(k-subset-sum (quote (-1 2 -3 4)) 2 1)
=> #true

(k-subset-sum (quote (-1 2 -3 4)) 2 -4)
=> #true

Hint: As above, your solution should not use subsets, or any of the other procedures you have already written, as a helper. Instead, try combining ideas from your k-subsets and subset-sum procedures.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback)