KnapSack

P. Baillehache

November 4, 2018

Contents

Introduction

KnapSack is a C library providing structures and functions to solve the 0/1 knapsack problem: given a set of item with a cost and value, which subset of items has the maximum total value while the total cost stays under a given budget.

It uses the `PBErr` and `GSet` library.

1 Definitions

The definition of the knapsack problem and its solution can be found on Wikipedia:

https://en.wikipedia.org/wiki/Knapsack_problem

2 Interface

```
// ============ KNAPSACK.H ================

#ifndef KNAPSACK_H
#define KNAPSACK_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "gset.h"

// ---- KnapSackPod

// ================= Data structure ===================

typedef struct KnapSackPod {
  // ID
  int _id;
  // Cost
  int _cost;
  // Value
  int _val;
} KnapSackPod;

// ================ Functions declaration ====================

// Create a new KnapSackPod with id 'id', cost 'cost' and value 'val'
KnapSackPod* KnapSackPodCreate(const int id, const int cost,
  const int val);

// Free the memory used by the KnapSackPod 'that'
void KnapSackPodFree(KnapSackPod** that);

// Get the id of the KnapSackPod 'that'
#if BUILDMODE != 0
inline
#endif
int KSPGetId(const KnapSackPod* const that);

// Get the cost of the KnapSackPod 'that'
#if BUILDMODE != 0
inline
#endif
int KSPGetCost(const KnapSackPod* const that);

// Get the value of the KnapSackPod 'that'
#if BUILDMODE != 0
inline
#endif
int KSPGetValue(const KnapSackPod* const that);
```

```
// Set the cost of the KnapSackPod 'that' to 'cost'
#if BUILDMODE != 0
inline
#endif
void KSPSetCost(KnapSackPod* const that, const int cost);

// Set the value of the KnapSackPod 'that' to 'val'
#if BUILDMODE != 0
inline
#endif
void KSPSetValue(KnapSackPod* const that, const int val);

// ---- 0-1 KnapSack

// ================= Data structure ==================

typedef struct KnapSack {
  // Budget
  int _budget;
  // GSet of selectable KnapSackPod
  GSetKnapSackPod _pods;
  // GSet of selected KnapSackPod
  GSetKnapSackPod _sel;
} KnapSack;

// ================ Functions declaration ====================

// Create a new KnapSack with the budget 'budget'
KnapSack* KnapSackCreate(const int budget);

// Free the memory used by the KnapSack 'that'
// The two GSetKnapSackPod are flushed but it's the responsibility
// of the user to free each KnapSackPod
void KnapSackFree(KnapSack** that);

// Get the budget of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
int KSGetBudget(const KnapSack* const that);

// Set the budget of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
void KSSetBudget(KnapSack* const that, const int budget);

// Get the GSet of selectable KnapSackPod of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
GSetKnapSackPod* KSPods(const KnapSack* const that);

// Select the best pods of the KnapSack 'that'
// https://en.wikipedia.org/wiki/Knapsack_problem
void KSSelect(const KnapSack* const that);

// Get the GSet of selected KnapSackPod of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
GSetKnapSackPod* KSSelectedPods(const KnapSack* const that);
```

```
// Get the 'iPod'-th KnapSackPod of selectable pods of the
// KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
const KnapSackPod* KSGetPod(const KnapSack* const that, const int iPod);

// Get the number of selectable pods of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
int KSGetNbPod(const KnapSack* const that);

// Get the 'iPod'-th KnapSackPod of selected pods of the
// KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
const KnapSackPod* KSGetSelectedPod(const KnapSack* const that,
  const int iPod);

// Get the number of selected pods of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
int KSGetNbSelectedPod(const KnapSack* const that);

// Get the cost of the KnapSack 'that' for currently selected pods
int KSGetCost(const KnapSack* const that);

// Get the value of the KnapSack 'that' for currently selected pods
int KSGetValue(const KnapSack* const that);

// Add a new KnapSackPod with cost 'cost' and value 'val' to the
// KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
void KSAdd(KnapSack* const that, const int cost, const int val);

// ================ Inliner ====================

#if BUILDMODE != 0
#include "knapsack-inline.c"
#endif


#endif
```

3 Code

3.1 knapsack.c

```
// ============ KNAPSACK.C ================

// ================= Include =================
```

```
#include "knapsack.h"
#if BUILDMODE == 0
#include "knapsack-inline.c"
#endif

// ---- KnapSackPod

// ================ Functions implementation ====================

// Create a new KnapSackPod with id 'id', cost 'cost' and value 'val'
KnapSackPod* KnapSackPodCreate(const int id, const int cost,
  const int val) {
#if BUILDMODE == 0
  if (cost <= 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'cost' is invalid (0<%d)", cost);
    PBErrCatch(GSetErr);
  }
  if (val <= 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'val' is invalid (0<%d)", val);
    PBErrCatch(GSetErr);
  }
#endif
  // Declare the new pod
  KnapSackPod* pod = PBErrMalloc(KnapSackErr, sizeof(KnapSackPod));
  // Set properties
  pod->_id = id;
  pod->_cost = cost;
  pod->_val = val;
  // Return the new pod
  return pod;
}

// Free the memory used by the KnapSackPod 'that'
void KnapSackPodFree(KnapSackPod** that) {
  // Check argument
  if (that == NULL || *that == NULL)
    // Nothing to do
    return;
  // Free memory
  free(*that);
  *that = NULL;
}

// ---- KnapSack

// ================ Define ====================

#define KSMAX(a,b) ((a)>(b)?(a):(b))

// ================ Functions implementation ====================

// Create a new KnapSack with the budget 'budget'
KnapSack* KnapSackCreate(const int budget) {
#if BUILDMODE == 0
  if (budget < 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'budget' is invalid (0<=%d)", budget);
    PBErrCatch(GSetErr);
  }
#endif
```

```
  // Declare the new KnapSack
  KnapSack* that = PBErrMalloc(KnapSackErr, sizeof(KnapSack));
  // Set properties
  that->_pods = GSetKnapSackPodCreateStatic();
  that->_sel = GSetKnapSackPodCreateStatic();
  that->_budget = budget;
  // Return the new KnapSack
  return that;
}

// Free the memory used by the KnapSack 'that'
// The two GSetKnapSackPod are flushed but it's the responsibility
// of the user to free each KnapSackPod
void KnapSackFree(KnapSack** that) {
  // Check argument
  if (that == NULL || *that == NULL)
    // Nothing to do
    return;
  // Free memory
  while (GSetNbElem(&((*that)->_pods))) {
    KnapSackPod* pod = GSetPop(&((*that)->_pods));
    free(pod);
    pod = NULL;
  }
  GSetFlush(&((*that)->_sel));
  free(*that);
  *that = NULL;
}

// Select the best pods of the KnapSack 'that'
// https://en.wikipedia.org/wiki/Knapsack_problem
void KSSelect(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  // Flush the selected set
  GSetFlush(KSSelectedPods(that));
  // If there is no budget or no selectable pods
  if (KSGetNbPod(that) <= 0 || KSGetBudget(that) <= 0)
    // Nothing to do
    return;
  // Declare an array to calculate the solution
  int b = KSGetBudget(that) + 1;
  int* m = PBErrMalloc(KnapSackErr,
    sizeof(int) * (KSGetNbPod(that) + 1) * b);
  // Initialise the array
  for (int cost = b; cost--;)
    m[cost] = 0;
  // Calculate values in the array
  for (int iPod = 1; iPod <= KSGetNbPod(that); ++iPod) {
    const KnapSackPod* pod = KSGetPod(that, iPod - 1);
    for (int cost = 0; cost <= KSGetBudget(that); ++cost) {
      if (KSPGetCost(pod) > cost) {
        m[iPod * b + cost] = m[(iPod - 1) * b + cost];
      } else {
        m[iPod * b + cost] =
          KSMAX(m[(iPod - 1) * b + cost],
            m[(iPod - 1) * b + cost - KSPGetCost(pod)] +
```

```
            KSPGetValue(pod));
      }
    }
  }
  // Find the selected pods
  int iPod = KSGetNbPod(that);
  int cost = KSGetBudget(that);
  while (iPod > 0 && cost > 0) {
    if (m[iPod * b + cost] != m[(iPod - 1) * b + cost]) {
      const KnapSackPod* pod = KSGetPod(that, iPod - 1);
      GSetPush(KSSelectedPods(that), pod);
      cost -= KSPGetCost(pod);
    }
    --iPod;
  }
  // Free memory
  free(m);
}

// Get the cost of the KnapSack 'that' for currently selected pods
int KSGetCost(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  // Declare a variable to calculate the cost
  int cost = 0;
  // Loop on selected pods and sum the cost
  if (KSGetNbSelectedPod(that) > 0) {
    GSetIterForward iter =
      GSetIterForwardCreateStatic(KSSelectedPods(that));
    do {
      KnapSackPod* pod = GSetIterGet(&iter);
      cost += KSPGetCost(pod);
    } while (GSetIterStep(&iter));
  }
  // Return the value
  return cost;
}

// Get the value of the KnapSack 'that' for currently selected pods
int KSGetValue(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  // Declare a variable to calculate the value
  int val = 0;
  // Loop on selected pods and sum the value
  if (KSGetNbSelectedPod(that) > 0) {
    GSetIterForward iter =
      GSetIterForwardCreateStatic(KSSelectedPods(that));
    do {
      KnapSackPod* pod = GSetIterGet(&iter);
      val += KSPGetValue(pod);
    } while (GSetIterStep(&iter));
```

```
  }
  // Return the value
  return val;
}
```

3.2 knapsack-inline.c

```
// ============ KNAPSACK-INLINE.C ================

// ---- KnapSackPod

// ================ Functions implementation ====================

// Get the id of the KnapSackPod 'that'
#if BUILDMODE != 0
inline
#endif
int KSPGetId(const KnapSackPod* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return that->_id;
}

// Get the cost of the KnapSackPod 'that'
#if BUILDMODE != 0
inline
#endif
int KSPGetCost(const KnapSackPod* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return that->_cost;
}

// Get the value of the KnapSackPod 'that'
#if BUILDMODE != 0
inline
#endif
int KSPGetValue(const KnapSackPod* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return that->_val;
}

// Set the cost of the KnapSackPod 'that' to 'cost'
```

```
#if BUILDMODE != 0
inline
#endif
void KSPSetCost(KnapSackPod* const that, const int cost) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
  if (cost <= 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'cost' is invalid (0<%d)", cost);
    PBErrCatch(GSetErr);
  }
#endif
  that->_cost = cost;
}

// Set the value of the KnapSackPod 'that' to 'val'
#if BUILDMODE != 0
inline
#endif
void KSPSetValue(KnapSackPod* const that, const int val) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
  if (val <= 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'val' is invalid (0<%d)", val);
    PBErrCatch(GSetErr);
  }
#endif
  that->_val = val;
}

// ---- KnapSack

// ================ Functions implementation ====================

// Get the budget of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
int KSGetBudget(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return that->_budget;
}

// Set the budget of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
```

```
void KSSetBudget(KnapSack* const that, const int budget) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
  if (budget < 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'budget' is invalid (0<=%d)", budget);
    PBErrCatch(GSetErr);
  }
#endif
  that->_budget = budget;
}

// Get the GSet of selectable KnapSackPod of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
GSetKnapSackPod* KSPods(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return (GSetKnapSackPod*)&(that->_pods);
}

// Get the GSet of selected KnapSackPod of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
GSetKnapSackPod* KSSelectedPods(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return (GSetKnapSackPod*)&(that->_sel);
}

// Get the 'iPod'-th KnapSackPod of selectable pods of the
// KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
const KnapSackPod* KSGetPod(const KnapSack* const that, const int iPod) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
  if (iPod < 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'iPod' is invalid (0<=%d)", iPod);
    PBErrCatch(GSetErr);
```

```
  }
#endif
  return GSetGet(KSPods(that), iPod);
}

// Get the number of selectable pods of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
int KSGetNbPod(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return GSetNbElem(KSPods(that));
}

// Get the 'iPod'-th KnapSackPod of selected pods of the
// KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
const KnapSackPod* KSGetSelectedPod(const KnapSack* const that,
  const int iPod) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
  if (iPod < 0) {
    GSetErr->_type = PBErrTypeInvalidArg;
    sprintf(GSetErr->_msg, "'iPod' is invalid (0<=%d)", iPod);
    PBErrCatch(GSetErr);
  }
#endif
  return GSetGet(KSSelectedPods(that), iPod);
}

// Get the number of selected pods of the KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
int KSGetNbSelectedPod(const KnapSack* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  return GSetNbElem(KSSelectedPods(that));
}

// Add a new KnapSackPod with cost 'cost' and value 'val' to the
// KnapSack 'that'
#if BUILDMODE != 0
inline
#endif
```

```
void KSAdd(KnapSack* const that, const int cost, const int val) {
#if BUILDMODE == 0
  if (that == NULL) {
    GSetErr->_type = PBErrTypeNullPointer;
    sprintf(GSetErr->_msg, "'that' is null");
    PBErrCatch(GSetErr);
  }
#endif
  KnapSackPod* pod = KnapSackPodCreate(KSGetNbPod(that), cost, val);
  GSetAppend(&(that->_pods), pod);
}
```

4 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=knapsack
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$
```

5 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include "pberr.h"
#include "knapsack.h"

void UnitTestKnapSackPodCreateFree() {
  KnapSackPod* pod = KnapSackPodCreate(1, 2, 3);
  if (pod == NULL ||
    pod->_id != 1 ||
```

```
    pod->_cost != 2 ||
    pod->_val != 3) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KnapSackPodCreate failed");
    PBErrCatch(KnapSackErr);
  }
  KnapSackPodFree(&pod);
  if (pod != NULL) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KnapSackPodFree failed");
    PBErrCatch(KnapSackErr);
  }
  printf("UnitTestKnapSackPodCreateFree OK\n");
}

void UnitTestKnapSackPodSetGet() {
  KnapSackPod* pod = KnapSackPodCreate(1, 2, 3);
  if (KSPGetCost(pod) != pod->_cost) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetCost failed");
    PBErrCatch(KnapSackErr);
  }
  KSPSetCost(pod, 4);
  if (KSPGetCost(pod) != 4) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSSetCost failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSPGetValue(pod) != pod->_val) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetValue failed");
    PBErrCatch(KnapSackErr);
  }
  KSPSetValue(pod, 5);
  if (KSPGetValue(pod) != 5) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSSetValue failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSPGetId(pod) != pod->_id) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetId failed");
    PBErrCatch(KnapSackErr);
  }
  KnapSackPodFree(&pod);
  printf("UnitTestKnapSackPodSetGet OK\n");
}

void UnitTestKnapSackPod() {
  UnitTestKnapSackPodCreateFree();
  UnitTestKnapSackPodSetGet();
  printf("UnitTestKnapSackPod OK\n");
}

void UnitTestKnapSackCreateFree() {
  KnapSack* ks = KnapSackCreate(1);
  if (ks == NULL ||
    ks->_budget != 1 ||
    GSetNbElem(&(ks->_pods)) != 0 ||
    GSetNbElem(&(ks->_sel)) != 0) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KnapSackCreate failed");
```

```
    PBErrCatch(KnapSackErr);
  }
  KnapSackFree(&ks);
  if (ks != NULL) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KnapSackFree failed");
    PBErrCatch(KnapSackErr);
  }
  printf("UnitTestKnapSackCreateFree OK\n");
}

void UnitTestKnapSackSetGet() {
  KnapSack* ks = KnapSackCreate(1);
  if (KSGetBudget(ks) != ks->_budget) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetBudget failed");
    PBErrCatch(KnapSackErr);
  }
  KSSetBudget(ks, 2);
  if (KSGetBudget(ks) != 2) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSSetBudget failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSPods(ks) != &(ks->_pods)) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSPods failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSSelectedPods(ks) != &(ks->_sel)) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSSelectedPods failed");
    PBErrCatch(KnapSackErr);
  }
  KnapSackPod* podA = KnapSackPodCreate(1, 2, 3);
  KnapSackPod* podB = KnapSackPodCreate(4, 5, 6);
  GSetAppend(KSPods(ks), podA);
  GSetAppend(KSPods(ks), podB);
  if (KSGetNbPod(ks) != 2) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetNbPod failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSGetPod(ks, 0) != podA ||
    KSGetPod(ks, 1) != podB) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetPod failed");
    PBErrCatch(KnapSackErr);
  }
  GSetFlush(KSPods(ks));
  GSetAppend(KSSelectedPods(ks), podA);
  GSetAppend(KSSelectedPods(ks), podB);
  if (KSGetNbSelectedPod(ks) != 2) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetNbSelectedPod failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSGetSelectedPod(ks, 0) != podA ||
    KSGetSelectedPod(ks, 1) != podB) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetSelectedPod failed");
    PBErrCatch(KnapSackErr);
```

```
  }
  GSetFlush(KSSelectedPods(ks));
  KnapSackPodFree(&podA);
  KnapSackPodFree(&podB);
  KnapSackFree(&ks);
  printf("UnitTestKnapSackSetGet OK\n");
}

void UnitTestKnapSackSelect() {
  KnapSack* ks = KnapSackCreate(15);
  int data[10] = {12, 4, 1, 2, 2, 2, 1, 1, 4, 10};
  for (int i = 5; i--;)
    KSAdd(ks, data[2 * i], data[2 * i + 1]);
  KSSelect(ks);
  if (KSGetNbSelectedPod(ks) != 4) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSSelect failed");
    PBErrCatch(KnapSackErr);
  }
  int check[4] = {0, 1, 2, 3};
  for (int i = KSGetNbSelectedPod(ks); i--;)
    if (KSPGetId(KSGetSelectedPod(ks, i)) != check[i]) {
      KnapSackErr->_type = PBErrTypeUnitTestFailed;
      sprintf(KnapSackErr->_msg, "KSSelect failed");
      PBErrCatch(KnapSackErr);
    }
  if (KSGetCost(ks) != 8) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetCost failed");
    PBErrCatch(KnapSackErr);
  }
  if (KSGetValue(ks) != 15) {
    KnapSackErr->_type = PBErrTypeUnitTestFailed;
    sprintf(KnapSackErr->_msg, "KSGetValue failed");
    PBErrCatch(KnapSackErr);
  }
  KnapSackFree(&ks);
  printf("UnitTestKnapSackSelect OK\n");
}

void UnitTestKnapSack() {
  UnitTestKnapSackCreateFree();
  UnitTestKnapSackSetGet();
  UnitTestKnapSackSelect();
  printf("UnitTestKnapSack OK\n");
}

void UnitTestAll() {
  UnitTestKnapSackPod();
  UnitTestKnapSack();
  printf("UnitTestAll OK\n");
}

int main() {
  UnitTestAll();
  // Return success code
  return 0;
}
```

6 Unit tests output

```
UnitTestKnapSackPodCreateFree OK
UnitTestKnapSackPodSetGet OK
UnitTestKnapSackPod OK
UnitTestKnapSackCreateFree OK
UnitTestKnapSackSetGet OK
UnitTestKnapSackSelect OK
UnitTestKnapSack OK
UnitTestAll OK
```