kreher-stinson

Algorithms from the book implemented in GAP

Version 1.0

29 January 2016

Rafael Villarroel-Flores
Citlalli Zamora-Mejía

Rafael Villarroel-Flores Email: rvf0068@gmail.com
Homepage: http://rvf0068.github.io

Citlalli Zamora-Mejía Email: cizame@gmail.com

Copyright

© 2016 by Rafael Villarroel-Flores and Citlalli Zamora-Mejía

kreher-stinson package is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Contents

Chapter 1

Generating Combinatorial Objects

1.1 Subsets

1.1.1 KSSubsetLexRank

▷ KSSubsetLexRank(*number, subset*)   (function)

Returns the rank of *subset* as a subset of the set of numbers from 1 to *number* (Algorithm 2.1).

1.1.2 KSSubsetLexUnrank

▷ KSSubsetLexUnrank(*number, rank*)   (function)

Returns the subset of $\{1..number\}$ whose rank is *rank*. (Algorithm 2.2).

Chapter 2

Bactracking

2.1 Knapsack

2.1.1 KSCheckKnapsackInput

▷ KSCheckKnapsackInput(*profits, weights, capacity*)   (function)

Checks for valid input data for the Knapsack problems (Problems 1.1-1.4).

2.1.2 KSKnapsack1

▷ KSKnapsack1(*profits, weights, capacity*)   (function)

Implementation of Algorithm 4.1.

2.1.3 KSKnapsack2

▷ KSKnapsack2(*profits, weights, capacity*)   (function)

Implementation of Algorithm 4.3.

2.2 Generating all cliques

2.2.1 KSAllCliques

▷ KSAllCliques(*graph*)   (function)

Implementation of Algorithm 4.4. A graph G is defined by the list `graph`, which must be a list of subsets of $\{1,...,n\}$, for some integer n. The neighbors of vertex i are the elements of `graph[i]`.

2.3   Exact cover

2.3.1   KSExactCover

▷ KSExactCover(`number, cover`) (function)

Finds an subcollection of `cover` (which is a set of subsets of $\{1,..,\texttt{number}\}$) that is an exact cover of $\{1,..,\texttt{number}\}$, if it exists.

2.4   Bounding functions

2.4.1   KSSortForRationalKnapsack

▷ KSSortForRationalKnapsack(`profits, weights`) (function)

Given two vectors `profits`, `weights` of the same length, this function returns a vector of the two vectors, sorted in non-decreasing order of values of `profits[i]`/`weights[i]`.

2.4.2   KSRationalKnapsackSorted

▷ KSRationalKnapsackSorted(`profits, weights, capacity`) (function)

Solves the rational Knapsack problem with parameters given. The vectors `profits`, `weights` must already be sorted.

2.4.3   KSKnapsack3

▷ KSKnapsack3(`profits, weights, capacity`) (function)

Solves the Knapsack problem with parameters given, using the function KSRationalKnapsackSorted as bounding function.

2.4.4   KSRandomKnapsackInstance

▷ KSRandomKnapsackInstance(`size, maximum_weight`) (function)

Returns a random instance of a Knapsack problem, for `size` objects. The maximum weight is `maximum_weight`. For each i, the profit $P[i]$ is $2 * W[i] * \varepsilon$, where ε is a random number between 0.9 and 1.1.

2.4.5   KSRandomTSPInstance

▷ KSRandomTSPInstance(`n, Wmax`) (function)

Returns a random instance of the TSP problem, which is a symmetric `n` by `n` matrix, such that its ij entry is the cost to travel from city i to city j. The entries in the diagonal are made equal to ∞. Each cost is a random integer between 1 and `Wmax`.

2.5 Exercises

2.5.1 KSQueens

▷ KSQueens(*size*) (function)

Solves the *n* queens problem for a *size* × *size* board.

─── Example ───
```
gap> KSQueens(4);
[ 2, 4, 1, 3 ]
[ 3, 1, 4, 2 ]
```

Index