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

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).

1.1.3 KSkSubsetLexRank

▷ `KSkSubsetLexRank(T, k, n)` (function)

Finds the rank of *T*, among all *k*-subsets of an *n*-set.

1.1.4 KSkSubsetLexUnrank

▷ `KSkSubsetLexUnrank(r, k, n)` (function)

Given an integer *r* between 0 and $\binom{n}{k} - 1$, returns the *k*-subset of an *n*-set with rank *r*.

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 $\{1,..,$*number*$\}$) that is an exact cover of $\{1,..,$*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 KSRationalKnapsack-Sorted 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*.

2.4.6 KSTSP1

▷ `KSTSP1(G)` (function)

Solves the TSP problem, for the instance `G`, traversing the whole tree space.

2.4.7 KSMinCostBound

▷ `KSMinCostBound(V, G)` (function)

A bounding function for the TSP problem.

2.4.8 KSReduce

▷ `KSReduce(M)` (function)

Reduce function for matrices, which will be useful to implement a secound bounding function for the TSP problem.

2.4.9 KSReduceBound

▷ `KSReduceBound(V, M)` (function)

A second bounding function for the TSP problem. `V` is a partial solution, and `M` is the problem instance. This implements Algorithm 4.12.

2.4.10 KSTSP2

▷ `KSTSP2(G, F)` (function)

Solves the TSP problem for instance `G`, using the bounding function `F`.

2.4.11 KSMaxClique1

▷ `KSMaxClique1(G)` (function)

Adapts the function that lists the complete subgraphs of `G`, to find the size of the largest clique of `G`. This implements Algorithm 4.14.

2.4.12 KSMaxClique2

▷ `KSMaxClique2(G, F)` (function)

Finds the size of the maximum clique in the graph `G`, using the bounding function `F`. This implements Algorithm 4.19.

2.4.13 KSSizeBound

▷ `KSSizeBound(XX, G, Cl)` (function)

A bounding function for the MaxClique problem. `XX` is a complete subgraph of `G`, and `Cl` is the set of candidates to extend `XX`.

2.4.14 KSGenerateRandomGraph

▷ `KSGenerateRandomGraph(n)` (function)

Returns a list of edges of a random graph on `n` vertices. This implements Algorithm 4.20.

2.4.15 KSEdgeListToAdjacencyList

▷ `KSEdgeListToAdjacencyList(Ged, n)` (function)

Given the list of edges `Ged` of a graph with `n` vertices, returns the adjacency list of such graph.

2.4.16 KSGreedyColor

▷ `KSGreedyColor(G)` (function)

Colors the vertices of a graph `G` using a greedy strategy. This implements Algorithm 4.16.

2.4.17 KSSamplingBound

▷ `KSSamplingBound(XX, G, Cl)` (function)

A bounding function for the MaxClique problem. `XX` is a complete subgraph of `G`, and `Cl` is the set of candidates to extend `XX`. This function uses a fixed greedy coloring of the graph `G`. Implements Algorithm 4.17.

2.4.18 KSInducedSubgraph

▷ `KSInducedSubgraph(G, L)` (function)

Returns the adjacency list of the subgraph of `G` induced by the vertices in `L`.

2.4.19 KSGreedyBound

▷ `KSGreedyBound(XX, G, Cl)` (function)

A bounding function for the MaxClique problem. `XX` is a complete subgraph of `G`, and `Cl` is the set of candidates to extend `XX`. This uses a greedy coloring of the subgraph of `G` induced by `L`.

2.4.20 KSGenerateRandomGraph2

▷ `KSGenerateRandomGraph2(n, delta)` (function)

Returns the list of edges of a random graph on `n` vertices with edge density `delta`.

2.4.21 KSTSP3

▷ `KSTSP3(G, F)` (function)

Solves the TSP problem for instance `G`, using bounding function `F`, applying the branch and bound technique.

2.5 Exercises

2.5.1 KSQueens

▷ `KSQueens(size)` (function)

Solves the n queens problem for a `size` $\times$ `size` board.

Example

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

Index