Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Thomas Tschager
Antonis Thomas

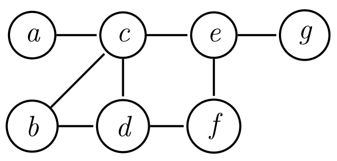
11th May 2016

Datenstrukturen & Algorithmen Exercise Sheet 11 FS 16

Exercise 11.1 *Branch and Bound.*

In this exercise, we wish to apply the *branch and bound* method to a classical optimization problem. Given an (undirected) graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, the *Minimum Dominating Set Problem* asks to compute a set of vertices $D \subseteq V$ of smallest possible size, such that every vertex in $V \backslash D$ has a neighbor in D (such a set is called *dominating* set).

We consider the following graph. In this graph, $D = \{a, b, f, g\}$ is a dominating set. No proper subset of D is a dominating set, but D is not as small as possible, so it is not a *minimum* dominating set. The exercise is to find a minimum dominating set for this graph by applying the branch and bound method by hand. Proceed as follows:



a) First, develop a lower bound on the number of required vertices for a given partial solution. A partial solution is described by two sets In and Out. The set In contains the vertices that are chosen to be part of the dominating set, while Out contains the vertices that are chosen *not* be part of it.

b) Provide a branching rule to specify which vertex is the next one to be considered.

c) Perform branch and bound using the answers you gave in a) and b). You do not have to use the learning module. Draw a complete decision tree and indicate the order of the decisions and the corresponding lower bounds. Identify the final solution.

Exercise 11.2 *Paging-Problem.*

We consider a two-level memory hierarchy that consists of a slow main memory and a small fast cache. The memory is split into pages of fixed size. The cache can store k pages and the main memory m pages, where $m \gg k$. A request of a page can be answered if the page is in the cache. Otherwise, a *page fault* occurs and the requested page has to be copied to the cache (by replacing another page).

The *Paging problem* is an *online minimization problem*, that is the input is a sequence of n requests of pages, such that the i-th request has to be answered without considering the following requests, and the number of page faults has to be minimized. An algorithm for this problem has to specify which page to evict from the cache in case of a page fault. We assume that the cache contains the first k pages before the first request.

a) Show that the following strategy is not competitive

Last In – First Out (LIFO): Evict the page from the cache that has been moved to the cache at last. At the first page fault, evict the first page from the cache.

b) Show that the following strategy is k-competitive:

First In – First Out (FIFO): Evict the page that has been in the case longest. The first k pages are evicted in increasing order.

Hint: Partition the input in consecutive, disjoint phases. The first phase ends after the first page fault of FIFO. All other phases end after exactly k page faults of FIFO or with the last request. Show that an optimal algorithm has at least one page fault in all phases besides the last phase.

Exercise 11.3 *Knapsack problem (Programming exercise).*

In this exercise we are going to implement a dynamic programming algorithm for solving the *Knapsack* problem. For this problem, a set $S = \{1, \ldots, n\}$ of n objects with values $v_1, \ldots, v_n \in \mathbb{N}$ and weights $w_1, \ldots, w_n \in \mathbb{N}$ is provided along with a weight limit $W \in \mathbb{N}$. A subset $I \subseteq S$ with $\sum_{i \in I} w_i \leq W$ is called *packing* and has the value $\sum_{i \in I} v_i$. The goal is to find a packing I with maximum value.

Input The first line contains only the number t of test instances. After that, we have exactly three lines per test instance. The first line contains the values n and W in this order, with $n, W \in \mathbb{N}$, $1 \leq n \leq 30$ and $1 \leq W \leq 200$. The second line contains the values $v_1, \ldots, v_n$, and the third line the weights $w_1, \ldots, w_n$. For each object $i \in \{1, \ldots, n\}$, we have $1 \leq v_i \leq 1000$ and $1 \leq w_i \leq 20$.

Output For every test instance we output only one line. This line contains the *value* of an optimum packing and the *index* of every object that is part of the optimal packing computed by the above algorithm. To avoid ambiguity, the indices must be printed in ascending order.

Example

Input:

```
2
2 3
1 1
2 5
3 5
1 3 2
2 2 1
```

Output:

```
1 1
6 1 2 3
```

For this exercise, there is only one testset for 100 points.

Hand-in: Wednesday, 18th May 2016 in your exercise group.