Lehrstuhl für Informatik I
Algorithmen und Komplexität
Universität Würzburg

Würzburg, 9 November 2020

Jonathan Klawitter
Boris Klemz
Oksana Firman

2nd Exercise Sheet

Advanced Algorithms (WS20)

Exercise 1 – Brute Force Maximum Shiny Set

In this exercise we design a very basic algorithm for testing arbitrary (decidable) subset problems on graphs. Let $G = (V, E)$ be a graph and $n = |V|$. An ordering of the nodes is available, that is, we can write $V[i]$ to index nodes $1 \leq i \leq n$.

Let $shiny(S) :: 2^V \to \{Yes, No\}$ be a black-box function that tells us whether $S \subseteq V$ is a 'shiny' subset. Now we want to find a shiny subset of maximum cardinality, which we call a Maximum Shiny Set.

a) Design a simple *recursive* algorithm that returns the size of a Maximum Shiny Set of a given graph G. It should make at most 2^n calls to $shiny$. Give pseudocode. **3 Points**

b) Now implement your algorithm in a "normal" imperative language, e.g. Java or C++/#. Assume that you are given a function `shiny`. It takes as argument an array of booleans indicating for each node whether or not it is in the subset; it returns whether this subset is shiny.

   Disregarding the time taken by `shiny`, your algorithm should take $O(2^n)$ time. (Equivalently, you may assume `shiny` takes constant time.) Your code should not take many lines. **3 Points**

c) A recursive algorithm goes through its recursion tree in depth-first order. Now specifically consider exponential-time branching algorithms. Would it be a good idea to do a breadth-first search of the branching tree rather than a depth-first recursion? Why or why not? **2 Points**

d) For arbitrary black-box properties it is unclear how one can do better. Now lets say $shiny$ has the following property: if S is shiny, then all $T \subseteq S$ are also shiny. This is a very common property (consider for example independent sets). How could you improve your recursive algorithm to take advantage of this property?

   Note: You do not need to *prove* a runtime better than $O(2^n)$; I know of no such bound in general. Just argue that your improvement is a good idea in practice. **3 Points**

Exercise 2 – Edge-branching Independent Set

In the lecture we talked about a branching algorithm for Maximum Independent Set. The algorithm was based on the following properties:

(Vertex 1) If a node is in the independent set, then its neighbours are not in it.

(Vertex 2) If a node is not in the independent set, then in a maximal independent set at least one of its neighbours is in the independent set.

Branching algorithms are often based on such observations about the properties of feasible and/or optimal solutions. We will now design an algorithm based on a different property of independent sets.

(Edge) Consider an edge (v, w). An independent set does not contain both v and w.

Design a branching algorithm for Maximum Independent Set based on this property of edges. The algorithm should have a runtime in $o(2^n)$.

a) Describe how your algorithm branches (based on property *Edge*). If appropriate, give pseudocode. **4 Points**

b) Prove an appropriate upper bound on the worst-case runtime of your algorithm. Use of $O^*(\cdot)$ notation is allowed. **4 Points**

c) Show that your running time analysis is tight by constructing a suitable family of input instances. **2 Points**

Exercise 3 – Enumerating Maximal Independent Sets

In the lecture we dealt with *maximum* independent sets, which are independent sets of maximum cardinality. In contrast, a *maximimal* independent set I is an independent set such that no proper superset of I is an independent set.

a) Give an algorithm that explicitly enumerates all maximal independent sets in time $O^*(3^{n/3})$. **3 Points**

b) Show that your running time analysis is tight by constructing a suitable family of input instances. **2 Points**

c) How many maximal independent sets can there be in a graph with n vertices? How few? **2 Points**

This assignment is due on November 16 at 5:00 in the morning. Please submit your solutions via WueCampus. The exercises on this assignment will be discussed in the tutorial session on November 16 at 13:00.