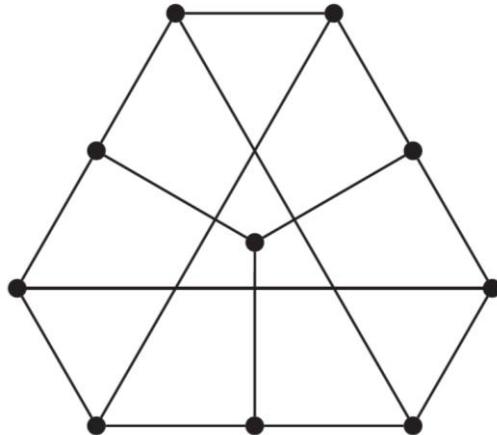


## Algorithmics II (H)

### Tutorial Exercises on Algorithms for “Hard” Problems

1. Let  $C_1, \dots, C_n$  be a set of courses offered at a certain college, and let  $t_1, \dots, t_n$  be the time intervals (not necessarily disjoint) during which the courses are offered. For example,  $t_1$  might be Tuesday 1000-1100,  $t_2$  Tuesday 1030-1200, etc. Your job is to assign classrooms to the courses. The only requirement is that no two courses ever overlap in the same room (so you can assume that any room is suitable, in terms of size etc., for any class). The goal is to use the minimum number of rooms that satisfy the requirement.
  - (a) Reduce this problem to a graph colouring problem.
  - (b) Design an efficient algorithm to solve the original problem.
  - (c) Given that graph colouring is an NP-complete problem, why does your solution not imply that  $P = NP$ ?
2. Recall the backtracking algorithm for Graph Colouring described in lectures. How can the next vertex to colour, and the next colour to try for this vertex, be chosen to minimise the running time of the algorithm? Justify your choice, and use it to colour the following graph with a minimum number of colours. You may stop before all colourings have been explored, but must argue that a colouring with fewer colours does not exist if you do.



3. Describe a possible backtracking algorithm for the 3-Dimensional Matching problem, specified as follows:

*Instance:* 3 disjoint sets  $X$ ,  $Y$  and  $Z$  of equal size  $n$  and a set  $S$  of triples of the form  $(x,y,z)$  where  $x \in X$ ,  $y \in Y$ ,  $z \in Z$ .

*Question:* does there exist a perfect matching, i.e., a subset  $M$  of  $S$  in which every element of  $X$ ,  $Y$  and  $Z$  appears exactly once?

4. Recall from the lectures that the Subset Sum (SS) decision problem is defined as follows:

*Instance:* a sequence  $x_1, \dots, x_n$  of positive integers, and a target positive integer  $K$ .

*Question:* does there exist a subset  $S$  of  $\{1, \dots, n\}$  such that  $\sum_{r \in S} x_r = K$ ?

Also recall from the lectures that the Knapsack Problem (KP) is an optimisation problem defined as follows:

*Instance:*  $n$  items, where item  $i$  has weight  $w_i$  and profit  $p_i$  ( $1 \leq i \leq n$ ); knapsack capacity  $C$ .

*Solution:* the maximum  $k$  such that there exists a subset  $S$  of  $\{1, \dots, n\}$  such that  $\sum_{r \in S} w_r \leq C$  and  $\sum_{r \in S} p_r = k$ ?

- (a) Write down the decision version, KP-D, of KP.
  - (b) Prove that KP-D is NP-complete. You may assume that SS is NP-complete.
5. Recall from the lectures that there is an  $O(nC)$  algorithm for KP based on dynamic programming. The algorithm constructs a table of dimensions  $(n+1) \times (C+1)$  such that the optimal value  $k$  of a knapsack packing can be obtained from entry  $S(n, C)$ . Show how to use the table in order to recover an actual collection of items whose total profit is  $k$  and whose total weight does not exceed  $C$ . Illustrate your answer by constructing the table corresponding to the KP instance involving 4 items, with weights 2, 1, 3, 2 and profits 12, 10, 20, 15 respectively, and knapsack capacity 5.
6. Let  $\Pi = \langle I, \text{SOL}, m, \text{GOAL} \rangle$  be an optimisation problem in NPO, where  $I$  is the set of instances of  $\Pi$ ,  $\text{SOL}(x)$  is the set of feasible solutions for a given instance  $x$  of  $\Pi$ ,  $m(x, y)$  is the measure of a given feasible solution  $y \in \text{SOL}(x)$ , for a given instance  $x$  of  $\Pi$ , and  $\text{GOAL}$  is max or min.
- (a) Define the decision version  $\Pi_d$  of  $\Pi$ .
  - (b) Suppose that  $\Pi \in \text{PO}$  and  $\Pi$  is NP-hard. Show that  $\text{P}=\text{NP}$ .
7. Let  $G = (V, E)$  be an undirected graph. A *vertex cover* in  $G$  is a set  $S \subseteq V$  such that, for any edge  $\{u, v\} \in E$ , either  $u \in S$  or  $v \in S$  (that is,  $\{u, v\}$  is *covered* by either  $u$  or  $v$ ). Let MVC denote the problem of finding a vertex cover in  $G$  of minimum size.
- (a) Give a polynomial-time algorithm to find a minimum vertex cover in a bipartite graph  $G$ . (*Hint:* assume that  $G = (V, E)$  has bipartition  $V = U \cup W$ , and start off with a maximum matching  $M$  in  $G$ . Define an edge in  $G$  to be *reachable* if it lies on an alternating path that starts at an exposed vertex of  $U$ . Define a set  $S$  of vertices as follows:  $S$  contains one end vertex of each edge in  $M$ ; if the edge is reachable its  $W$ -endpoint is in  $S$ , otherwise its  $U$ -endpoint is in  $S$ .)
  - (b) Give a 2-approximation algorithm for MVC.

- (c) Suppose that  $n$  is an arbitrary positive integer. Describe a connected graph  $G_n$ , with  $2n$  vertices, for which your approximation algorithm from Part (b) might return a vertex cover  $S_n$  of size  $2n$ , whereas the size of a minimum vertex cover  $C_n$  is  $n$ . Your construction should be valid for all positive integers  $n$ . Indicate how the approximation algorithm  $A$  might arrive at  $S_n$ , and give an example of a minimum vertex cover  $C_n$  in  $G_n$ .
8. Let  $G = (V, E)$  be an undirected graph. An *independent set* in  $G$  is a set  $S \subseteq V$  such that, for any edge  $\{u, v\} \in E$ , at most one of  $u, v$  belongs to  $S$ . Let MIS denote the problem of finding an independent set in  $G$  of maximum size. It is known that MIS is NP-hard, and hence the problem is unlikely to be polynomial-time solvable. Consider the following approximation algorithm  $A$  for MIS:
- ```

X = V;
S = ∅; //store the independent set
while ( X ≠ ∅ )
{
    choose v ∈ X;
    S = S ∪ {v};
    X = X \ {v};
    for ( each w adjacent to v )
        X = X \ {w};
}
return S;

```
- (a) Show that  $A$  returns an independent set in  $G$ .
- (b) Show that  $A$  does not have performance guarantee  $c$ , for any  $c > 1$ .
9. A vertex cover  $S$  of a graph  $G$  is *independent* if no two vertices in  $S$  are adjacent in  $G$ .
- (a) Show that a graph need not admit an independent vertex cover.
- (b) Give a polynomial-time algorithm to determine whether  $G$  admits an independent vertex cover. Assume that  $G$  is connected.
- (c) Show that a graph might admit independent vertex covers of different sizes.
- (d) Give a polynomial-time algorithm to find an independent vertex cover of maximum size, assuming one exists. Again assume that  $G$  is connected.
10. Consider the MVC problem as defined in Question 7. It is known that MVC is NP-hard, and hence the problem is unlikely to be polynomial-time solvable. Give an integer programming formulation for MVC.
11. Let  $G = (V, E)$  be an undirected graph. A *colouring* in  $G$  is a function  $f: V \rightarrow \mathbb{Z}^+$  such that  $f(u) \neq f(v)$  for all  $\{u, v\} \in E$ .  $f$  is a  $k$ -colouring if  $\{f(v) : v \in V\} = \{1, 2, \dots, k\}$ . Let MGC denote the problem of finding the minimum  $k$  such that  $G$  admits a  $k$ -colouring. It is known that MGC is NP-hard, and hence the problem is unlikely to be polynomial-time solvable. Give an integer programming formulation for MGC.

12. Consider the following optimisation problem in NPO:

**Subset Sum Optimisation (SSO)**

*Instance:* a collection of positive integers  $a_1, a_2, \dots, a_n$ , and a threshold value  $t$

*Feasible solutions:* any subset  $S$  of  $\{1, \dots, n\}$  such that  $\sum_{i \in S} a_i \leq t$

*Measure:*  $\sum_{i \in S} a_i$

*GOAL:* max

Show that SSO is NP-hard. You may assume that SS, defined in Question 4, is NP-complete.

13. Use the trimming algorithm to find a solution whose measure is at least one third of the optimal measure for the following instance of SSO:  $\{3, 7, 13, 15, 19, 21\}$ ,  $t = 26$ .

14. Recall from the lectures that the Minimum Bin Packing problem (MBP) is NP-hard, and hence is unlikely to have a polynomial-time algorithm. An approximation algorithm, *Next Fit*, for MBP can be expressed as follows:

```
j = 0;
for ( item x : items ) {
    if (bin j has no room for x)
        j++;
    place x in bin j;
}
```

- (a) Show that Next Fit is a 2-approximation algorithm for MBP.
- (b) Show that, asymptotically, Next Fit might return a packing using twice the number of bins that would be required in an optimal bin packing.

15. The *First Fit (FF)* algorithm for Minimum Bin Packing can be expressed as follows:

```
for (item x : items)
    if (no existing bin can accommodate x)
        place x in a new bin;
    else
        place x in the first bin that can accommodate it;
```

- (a) Show that First Fit is a 2-approximation algorithm for MBP.
- (b) By considering bin capacity 126, and an appropriate number of items of weights 19, 43, and 64, prove that there are arbitrarily large instances of MBP for which FF produces a solution that uses  $5/3$  times as many bins as the optimal solution.

(In fact, it is known that  $N \leq 17/10T+2$ , where  $N$  is the number of bins returned by FF and  $T$  is the number in an optimal solution, but this is hard to prove.)