Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 8: SOLUTIONS

1. Consider the following context free grammar with start symbol S:

S	→	NP VP	PP	→	Pre NP
S	→	*I* VP PP	V	→	*ate*
NP	→	Det N	Det	→	*the* | *a*
VP	→	*ate* NP	N	→	*fork* | *salad*
VP	→	V	Pre	→	*with*

   (a) *Convert this grammar to Chomsky Normal Form (see Lecture 22).*

   Here we follow the order and numbering of steps given in the 2024 lecture slides and live lecture. (This is slightly different from the order in the video lecture from 2021. In practice, there is a lot of flexibility and the order of steps doesn't matter much for small examples.)

   Step 1: Eliminate the ternary rule S → *I* VP PP. We can do this by introducing a fresh non-terminal X and replacing the rule by S → *I* X and X → VP PP.

   Steps 2 and 3: There are no ϵ-rules, so these can be omitted. (In general, ϵ-rules are more typical of formal language grammars than natural language ones.)

   Step 4: We deal with the unit rule VP → V by replacing it with VP → *ate*.

   Step 5: We introduce a separate non-terminal for each terminal, which we'll do by writing the same word capitalized in Roman font (e.g. I, Ate, The). We also add corresponding expansion rules (e.g. I → *I*; Ate → *ate*) and then replace all terminals within non-unary right-hand sides by the corresponding non-terminal (e.g. S → I VP PP; VP → Ate).

   Discarding the rules for non-terminals now unreachable from S (e.g. V, The), the resulting grammar is now as follows:

S	→	NP VP	I	→	*I*
PP	→	Pre NP	Ate	→	*ate*
S	→	I X	Det	→	*the* | *a*
X	→	VP PP	N	→	*fork* | *salad*
NP	→	Det N	Pre	→	*with*
VP	→	Ate NP			
VP	→	*ate*			

   (Other minor variations are of course acceptable, provided they are indeed in CNF and are equivalent to the original grammar.)

(b) *Use the CYK algorithm from Lecture 22 to parse 'I ate the salad with a fork'.* Using the above CNF grammar, the CYK chart would be:

	1	2	3	4	5	6	7
0	I						S
1		VP,Ate		VP			X
2			Det	NP			
3				N			
4					Pre		PP
5						Det	NP
6							N

(c) *How many complete analyses of the sentence do you get?* Just the one:

(S (I *I*) (X (VP (V *ate*)(NP(Det *the*)(N *salad*)))
(PP (Pre *with*)(NP (Det *a*) (N *fork*)))))

(d) *Now add a further production rule to your CNF grammar to allow for the alternative prepositional phrase attachment, i.e. 'the salad with a fork'. Revise your CYK chart or graph to include any new entries this introduces.*

(e) We could add a new rule

$$\text{NP} \rightarrow \text{NP PP}$$

This would add an NP entry to cell (2,7), hence a VP entry to (1,7). However, no new S entry would be added to (0,7), so there is still just one complete parse. This is because the grammar lacks the means to derive *I* from NP.

2. The grammar:

Terminals:	(,), $*$, n
Nonterminals:	Exp, Ops
Productions:	Exp $\rightarrow$ n Ops | (Exp)
	Ops $\rightarrow$ ϵ | $*$ n Ops
Start symbol:	Exp

The parse table:

	(	)	$*$	n	$
Exp	(Exp)			n Ops	
Ops		ϵ	$*$ n Ops		ϵ

(a) *Using this table, apply the LL(1) parsing algorithm to the input* (n $*$ n).

Operation	Input remaining	Stack state
	(n * n)$	Exp
Lookup (,Exp	(n * n)$	(Exp)
Match (	n * n)$	Exp)
Lookup n, Exp	n * n)$	n Ops)
Match n	* n)$	Ops)
Lookup *, Ops	* n)$	* n Ops)
Match *	n)$	n Ops)
Match n	)$	Ops)
Lookup), Ops	)$	)
Match)	$	STACK EMPTIES AT END OF STRING: SUCCESS!

(b) *For each of the following three input strings, explain how and where an error arises in the course of the LL(1) parsing algorithm. In each case, suggest a reasonable error message.*

( ) n) n *

- For ( ), the parser will encounter a blank table entry at), Exp.
  Message: ") Found where expression expected."
- For n), the stack will empty before end of input is reached.
  Message: ") Found after end of expression."
- For n *, the end of input will be reached with n Ops still on the stack, and the parser gets stuck since the top of the stack is a terminal n no different from $.
  Message: "End of input found where numeric literal expected."

3. (a) *Define the appropriate* decision *version of the* Maximum Independent Set *set problem, called* Independent Set.

The decision version of the problem is defined as follows:

Independent Set: Given a graph $G = (V, E)$ and an non-negative integer k as input, decide whether there is an independent set I of size at least k is G or not.

(b) *Show that* Independent Set *is NP-complete. For the NP-hardness, construct a polynomial-time reduction from* 3SAT.

Hint: Construct "clause gadgets", i.e., triangles of nodes corresponding to the three literals of a clause, similarly to the reduction from 3SAT *to* Vertex Cover *presented in class to show the latter problem is NP-hard.*

To construct our reduction, we will start with an arbitrary instance of 3SAT, i.e., an arbitrary 3CNF formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$, and we will construct a *specific* instance of Independent Set with $k = m$ such that

$$\phi \text{ is satisfiable } \Leftrightarrow \text{ there is an independent set of size at least } k \text{ in G.}$$

We construct the graph G as follows: For every clause C_j with variables x_{j1}, x_{j2} and x_{j3} we construct a triangle (a *clause gadget*) with nodes v_{j1}, v_{j2} and v_{j3}, such that each one of them is connected with each other via an edge. See Figure 1.

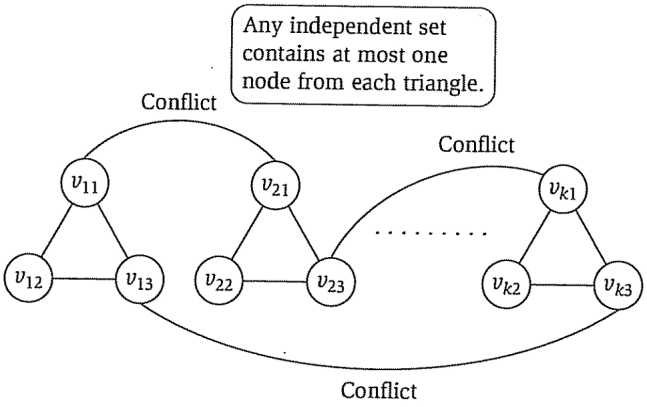


Figure 1: The construction of the graph G in the reduction. Figure taken from KT (Figure 8.3.).

We also add an edge between nodes that correspond to "complementary" variables x_i and $\neg x_i$. For example, if $x_{21} = \neg x_{11}$, then we add an edge (v_{11}, v_{21}) in the graph. Again, see Figure 1. We refer to those edges as "conflict" edges.

We will argue both directions of the equivalence above. First, assume that ϕ is satisfiable and let T be a truth assignment to its variables. That means that at least one variable $x_{j\ell}$ in clause C_j has been assigned the value TRUE, i.e., there exists $\ell \in \{1, 2, 3\}$ such that $T(x_{j\ell}) = 1$. For each clause pick (only) one of these variables arbitrarily and call it a *chosen variable*. In our graph G, let I be a set of nodes in G such that $v_{j\ell} \in I$ if any only if $x_{j\ell}$ is a chosen variable. First, observe that I has size $k = m$, since we only pick one node from each clause gadget (triangle) and we have $k = m$ triangles. Now observe that I is an independent set: for each triangle, only one node is selected, so there are no edges shared between nodes in I within each triangle. Additionally, for each conflict edge, only one of the two endpoints is selected (as only one of them can be set to 1 by T) by construction.

Now assume that there exists an independent set I of size at least k in G. By virtue of being an independent set, it must contain at most one node from each triangle, so its size is at most k. That means that its size is exactly k and that it contains exactly one node from each triangle. To create a truth assignment, we set $T(x_{j\ell}) = 1$ if the node corresponding to $x_{j\ell}$ is in I and $T(x_{j\ell}) = 0$ if the node corresponding to $\neg x_{j\ell}$ is in I. Node that it is not possible for both of these nodes to be in I, as there is an edge between them in G. Our truth assignment satisfies one variable from each clause and does not assign "conflicting" values to complementary variables, hence it satisfies the formula ϕ.

(c) *Assume that you have an* oracle *(i.e., an algorithm that runs in time $O(1)$ every-time it is called) to solve the* Maximum Independent Set *problem. Explain how to use this oracle to solve the* Independent Set *problem in polynomial time.*

This is straightforward, by observing that our algorithm can simply solve the Maximum Independent Set problem and then check if the size of the returned

indepndent set is at least k or not. This takes time $O(|I|)$ assuming we don't have access to the size of the set.

(d) *Assume that you have an oracle to solve the* Independent Set *problem. Explain how to use this oracle to solve the* Maximum Independent Set *problem in polynomial time.*

This is the other direction, which is more intricate. Let $|V| = n$. First, we will find the *size* of the maximum independent set. For that, we can check every possible size from $k = 0$ to $k = n$. We know that there is definitely an independent set of size 0, so the "answer" to the input with G and $k = 0$ is "yes". If there is an independent set of size n (which happens only when all the nodes are isolated), then the answer to the input with G and $k = n$ is also "yes" and we return $k = n$. Otherwise, it is "no", and the answer has to switch from "yes" to "know" for some value of k between 0 and n. We can find that value using binary search in time $O(\lg n)$ because the size of the maximum independent set is monotone: If there is an independent set of size k_1, there is also an independent set of size $k_2 < k_1$, and if there is no independent set of size k_1, there there is no independent set of size $k_2 > k_1$. Let k^* be the size of the maximum independent set we find this way.

Next, we need to find the independent set I itself. Here we work similarly to the case of Vertex Cover. We consider node n of the graph and ask our oracle whether there is an independent set of size k^* on input G', where G' is obtained from G by removing node n and all its incident edges. If the answer is "yes", that means that n does not have to be part of the optimal solution, so we do not include it in I and continue in the same manner. If the answer is "no", then we have to include n in the optimal solution I, and we run our oracle on input G' and $k = k^* - 1$. In time $O(n)$ we will construct the independent set I with size k^*.

4. (*) *A k-colouring of a graph G is a function $f : V \to \{1, 2, \ldots, k\}$ mapping nodes to* colours*, such that for any nodes u and v such that $(u, v) \in E$, it holds that $f(u) \neq f(v)$.*

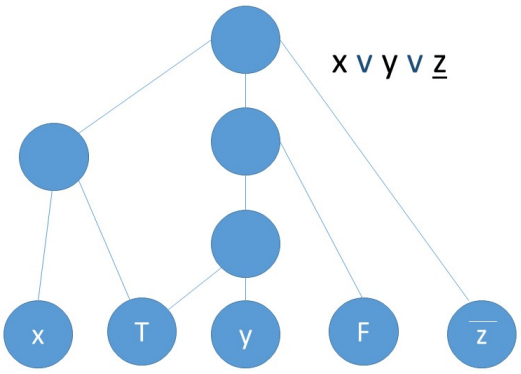
Consider the 3-colouring *problem: Given a graph G as input, decide whether there is a 3-colouring of G. Prove that* 3-colouring *is NP-complete. For the NP-hardness, construct a polynomial-time reduction from* 3SAT*.*

It is easy to see that 3-colouring is in NP: if we are given an assignment of colours to the vertices, we can check in polynomial time whether there exist neighbouring vertices with the same colour. To how the NP-hardness of the problem, we will construct a polynomial time reduction from 3SAT. Let ϕ be a 3SAT formula. We will have the following gadgets.

Gadget 1: A triangle of three nodes, labelled T, F and O. The colour assigned to T will be interpreted as "true" for ϕ and the colour assigned to F will be interpreted as "false". The colour assigned to O will simply be the third colour.

Gadget 2: For each variable v in ϕ, construct two vertices v and $\bar{v}$. Add an edge $(v, \bar{v})$ and edges (v, O) and $(\bar{v}, O)$ (i.e., $v, \bar{v}$ and O form a triangle).

Gagdet 3: This is the more complicated gadget shown in the figure (corresponding to the clause $x \vee y \vee \bar{z}$).



Suppose that ϕ is satisfiable, and let x be a satisfying assignment. For a variable v, if v is set to true, colour the corresponding vertex by T (and since $\bar{v}$ is set to false, colour $\bar{v}$ by T). Likewise, if v is set to false, colour the corresponding vertex v by F and $\bar{v}$ by T. Node that since v and $\bar{v}$ are connected only to O and $\bar{v}$ and v respectively in the Gadget 2 triangles, that part of the graph is 3-colourable. It remains to assign colours to Gadget 3, avoiding having any neighbours with the same colour.

Looking at the gadget of the figure, label the non-labelled vertices $1, 2, 3, 4$ starting from the left and then starting from the top in the middle column. Consider vertex 1, which is a neighbour of both x and T. If x is coloured T, then vertex 1 can be set to either O or F and if x is coloured F, then it must be set to O. Similarly for vertex 4, which is a neighbour of both T and y. If $\bar{z}$ is coloured T, we can set vertex 2 to F, vertex 1 to O, vertex 2 to T and vertex 4 to O and we have a 3-colouring. If $\bar{z}$ is set to F, then we consider the labelling of vertices x and y. Considering a few cases, we can verify that there is always a 3-colouring of the gadget.

Supppose now that we have a 3-colouring of the graph. Then, for every vertex u is coloured either T or F, we set the corresponding variable in ϕ to true or false accordingly. From the fact that the labelling is a 3-colouring and the way the graph is constructed, we know that two nodes v and $\bar{v}$ cannot receive the same colour, and therefore it is not possible for both variable v and its negation to receive the same value in the truth assignment. Finally, the correctness of the reduction follows from the fact that it is not possible for any vertex v or $\bar{v}$ to receive the colour O, because all of these vertices are connected to a vertex coloured O, and that would violate the fact that the graph is 3-colourable.

John Longley and Aris Filos-Ratsikas

February 2023