ALGORITHMICS AND PROGRAMMING III
PROBLEM SET

Albert Oliveras Enric Rodríguez-Carbonell

(Editors)

September 20, 2021

Departament de Ciències de la Computació
Universitat Politècnica de Catalunya

1

Tractability

1. **A Tragicomedy in Three Acts**. Consider the following **NP**-complete problems:

   (a) BIN-PACKING: Given n items with sizes $d_1, \ldots, d_n$ and k bins with capacity m, determine whether all the items can be packed into the bins so that the capacity is not exceeded.

   (b) TRIPARTITE-MATCHING: Given three disjoint sets C_1, C_2, C_3 having n elements each and given a subset $S \subseteq C_1 \times C_2 \times C_3$, determine whether it is possible to select n elements from S so that every element from C_1, C_2, and C_3 appears only once in these n triples.

   (c) 3-COLORABILITY: Given a graph, determine whether it is possible to assign colors to vertices, choosing among three possible colors, so that no edge has its endpoints of the same color.

   (d) HAMILTONIAN-GRAPH: Given a graph, determine whether it has a cycle visiting each vertex exactly once.

   (e) DOMINATING-SET: Given a graph and a positive integer k, determine whether it has a subset with k vertices such that every vertex outside the subset is adjacent to a vertex in the subset.

   Each of the following scenes presents a problem that is an instance of one of the previous **NP**-complete problems. Determine to which one it corresponds and why.

   Scene 1. Johnny asked Roy to organize a dinner for the usual group of friends. Then, Roy has booked a table at a restaurant; it's a very big round table and everybody fits. However, when they get at the place, the first difficulties arise.

   JOHNNY: Listen Roy, it seems that some people in the group can't stand each other. It would be better if two guys who are angry with each other don't have to sit side by side.

   ROY: Come on... Look, let's get straight to the point. I want you to collect all the

relevant information; that is, for each pair of people, whether they are angry with each other or not. I'll figure out how to seat them so that nobody has a neighbor at the table with whom he or she is angry.

Scene 2. With all the data in his hands, after a long while, Roy hasn't yet finished finding a solution. Moreover, Johnny is back with a worried face.

JOHNNY: Hey, it seems that the guys who are angry with each other don't even want to sit at the same table. But don't worry, I've asked at the restaurant and they said they can prepare three large tables for us.

ROY: Let's see, I'll try to distribute people into three groups so that there's no couple of angry fellows inside any of the groups.

Scene 3. While Roy thinks on how to distribute people around the three tables, Johnny appears again saying that the restaurant is about to close and that they'd better look for a different place for dinner.

ROY: Look, before looking for another place, I'd like to make some things clear to the group: When they say "hey, hey, I don't want to sit near this or that person", they're being childish! I want to talk to them seriously. The problem is that I'm a little aphonic and if I speak to everybody, I'll ran out of voice. I want you to choose k representatives so that, for any person in the group, there is one of the representatives with whom that person is not angry. This way, the k representatives will spread the message to the whole group.

Scene 4. Since Johnny doesn't succeed in choosing the k representatives, he finally takes a megaphone and tells everybody that if they keep acting like that, they'll have to go home very soon.

JOHNNY: Listen, I have a city map where I've located the squares having good restaurants and the streets joining them. We could walk through all of them and check whether they have free tables.

ROY: OK, give me the map for a while so that I can choose the tour; it's cold outside and I'm not feeling like visiting the same place twice.

Scene 5. After a long while, Roy still can't find the solution in spite of having all the available information. Johnny wants to help him:

JOHNNY: Listen Roy, I have a handful of colleagues in town and for each of these squares, a friend of mine lives there. Give me some bucks and I'll phone them and ask them if we can have dinner at any restaurant in their squares.

ROY: Man, I'm skint and can't afford so many calls. I'll give you enough for m phone calls. You can ask your friends to look for restaurants not only in their squares but also in the squares one street away from theirs. Choose the right friends so that they can cover all the squares.

Scene 6. Since he doesn't succeed, Johnny calls to the first person in the list and it turns out that there is a restaurant with free tables near him. When they get there, there is only one table with k chairs, and they are closing soon.

JOHNNY: I got it, here's a solution: We'll have dinner by turns; when somebody finishes, he or she will go out and somebody else will come in. Look, some time

ago I collected information about how long everyone takes for dinner. You know I have some weird hobbies...

ROY: My God...

JOHHNY: With all this data, we can plan the distribution of people on the chairs, so that everybody can have dinner before they close.

ROY: OK, give me that list with the time everybody takes to have dinner and I'll see what can I do. Someday you'll tell me about what information you collect from people...

Scene 7. In the meanwhile, Johnny gets a phone call from a friend saying that near that place there's a very cool restaurant. They all go there at once. When they get there, it seems there are only a few small tables and, on top of that, they all have different colors.

JOHNNY: Listen, Roy. It seems that people are a little bored and they want to take advantage of the situation to sit properly by couples. We are as many boys as girls, and everybody has their own preferences. But the colored tables play a role, too. For example, that person accepts to sit with that other person only if the table is pink, and with some other person if the table is blue. We'll have to find a distribution that satisfies everyone's restrictions. It seems that after all this time without eating, the troop is feeling a bit strange.

ROY: I'm not in the mood with all this nonsense, but OK, I'll try.

Scene 8. They haven't succeeded and it is already very late. Roy and Johnny can only find a small bar having just one table with one chair. However, it is open until very late.

JOHNNY: It seems that the band is a bit bored. Some guys want to go out and come back later. Here I have, for every person, the time they'll come back and the time they'll have to go home. And I still have the list of the time everybody needs to have dinner. I think that, with this information, it should be easy to see if everybody can be assigned a time so that they can sit on the chair and eat according to their schedules.

ROY: Listen Johnny, I'm fed up. I want you to know that this is going to be the last thing I do for this gang. Come on, give me this data and I'll see what can I do.

[In this scene, it is not enough to identify a problem from the list; it is necessary to find a reduction from one of the **NP**-complete problems to it.]

Scene 9. Finally, Roy and Johnny decide to have dinner together at that place, they have only had to ask for another chair to the waitress. They have given an address to the rest of the group where they will just find a wasteland with the highest criminality index in town.

JOHNNY: Come on, man, don't take it like that. Today we just had bad luck.

ROY: Look, it's not bad luck at all. It's not the first time this happens to me. I'm foolish and I'll never learn. Now, I want you to listen carefully. I know I told you before, but this time I'm serious: I'll never ever organize anything else for this gang. And when I say never, it's never. In fact, I don't want even to meet them again.

JOHNNY: OK, right! But it's a pity. Next week, we were thinking of going together to a holiday camp. You know, nature, pure air, quiet, fun, roast meat, naked baths in the river...

ROY: Well, that sounds good! But I don't know, these things always go wrong in the end.

JOHNNY: Besides, this time Steffy is coming. Do you remember?

ROY: STEFFY!?!?!?

JOHNNY: Yes, and she said she's longing very much to see you. I think she's going to have a big disappointment if you don't show up.

ROY: Hummm... well... maybe I've gone a bit too far. In the end, they are good guys... Sometimes they're unbearable, but they're good people, and that's what matters. Friendship, in fact, is full of bad moments, but it's mainly full of marvelous moments of bliss. Come on! What the hell! I'm coming. And if you want, I can help you organize things.

JOHNNY: Perfect! You're the best! A winner, for sure! Come and let's organize it right now. Let's see, the only problem is that there are k cars and...

2. **Find in the code**. Recall that the problem of HAMILTONIAN GRAPH consists in, given an (undirected) graph, to decide whether it is Hamiltonian, that is to say, whether there exists a cycle that visits all its vertices exactly once. It is well-known that HAMILTONIAN GRAPH is an NP-complete problem.

   It is also known that, from the proof that a problem belongs to class NP, one can derive a brute force algorithm that solves it. The following function **bool** *ham*(**const vector**<**vector**<**int**≫& *G*) which, given a graph *G* represented with adjacency lists, returns if *G* is Hamiltonian, implements this algorithm in the case of HAMILTONIAN GRAPH.

```
1  bool ham_rec(const vector<vector<int>>& G, int k, vector<int>& p) {
2    int n = G.size ();
3    if (k == n) {
4      vector<bool> mkd(n, false);
5      for (int u : p) {
6        if (mkd[u]) return false;
7        mkd[u] = true;
8      }
9      for (int i = 0; i < n; ++i) {
10       int u = p[i];
11       int v;
12       if (i < n-1) v = p[i+1];
13       else         v = p[0];
14       if (find(G[u].begin (), G[u].end (), v) == G[u].end()) return false;
15     }
16     return true;
17   }
18   for (int v = 0; v < n; ++v) {
19     p[k] = v;
```

```
20        if (ham_rec(G, k+1, p)) return true;
21      }
22      return false;
23    }
24
25    bool ham(const vector<vector<int>>& G) {
26      int n = G.size ();
27      vector<int> p(n);
28      return ham_rec(G, 0, p);
29    }
```

Note: The function of the STL library

```
Iterator find ( Iterator first , Iterator last , int val );
```

returns an iterator to the first element in the range $[first, last)$ that compares equal to *val*. If no such element exists, the function returns last.

(a) Identify the variable in the previous program which represents the witness when the function *ham* returns **true**.

(b) Identify the code corresponding to the verifier in the previous program. To that end, use the line numbers on the left margin.

3. **CLIQUE**. Reduce problem INDEPENDENT-SET to CLIQUE: Given a graph and a positive integer k, determine whether there is a subset of k vertices such that there is an edge between every pair of vertices in the subset. Additionally, prove that CLIQUE belongs to **NP**.

4. **DELETED-SUBGRAPH**. Reduce problem HAMILTONIAN-GRAPH to DELETED-SUBGRAPH: Given two graphs G_1 and G_2, determine whether it is possible to delete some edges of G_2 in such a way that the resulting graph is isomorphic to G_1. Additionally, prove that DELETED-SUBGRAPH belongs to **NP**.

5. **INDUCED-SUBGRAPH**. Reduce problem CLIQUE to problem INDUCED-SUBGRAPH: Given two graphs G_1 and G_2, determine whether G_1 is isomorphic to an induced subgraph of G_2. Additionally, prove that INDUCED-SUBGRAPH belongs to **NP**.

6. **PARTITION**. Reduce problem SUBSET-SUM to problem PARTITION: Given a set of integers S (possibly with repetitions), determine whether S can be *partitioned* into two subsets S_1 i S_2 (i.e., each element of S belongs either to S_1 or to S_2, but not the two at the same time) such that $\sum_{x \in S_1} x = \sum_{x \in S_2} x$. In addition, prove that PARTITION belongs to **NP**.

7. **Reductions in What Direction?**. The problems of satisfiability of Boolean formulas (SAT) and of determining whether a given graph is Hamiltonian (HAM) are two well-known **NP**-complete problems. The problem of determining whether a given graph contains an Eulerian circuit (EUL) has a solution with cost $\Theta(n + m)$, where n is the number of vertices in the graph and m is its number of edges.

   Which one of the following sentences can we say it is true?

(a) SAT reduces to HAM and HAM reduces to EUL.

(b) EUL reduces to SAT and SAT reduces to HAM.

(c) SAT and HAM reduce to each other; EUL only reduces to HAM.

(d) SAT reduces to EUL and EUL reduces to HAM.

8. **True or false?**. For each of the following statements, mark with an X its cell depending on whether it is true or false.

   Recall: SAT is the problem of, given a propositional formula, to decide whether it is satisfiable or not; and HAM is the problem of, given a graph, to decide whether it is hamiltonian.

   (1) If there exists an NP problem that belongs to P then P $=$ NP.

   (2) If there is a polynomial reduction from a problem X to SAT then there is a polynomial reduction from X a HAM.

   (3) There are problems of the class NP for which there is no known algorithm that solves them.

   (4) If a problem X can be reduced polynomially to a problem $Y \in$ P then $X \in$ P.

   (5) If A and B are decision problems such that $A \in$ NP and B can be reduced polynomially to A, then $B \in$ NP.

   (6) If $A \in$ NP $-$ hard then $A \in$ NP.

	(1)	(2)	(3)	(4)	(5)	(6)
TRUE						
FALSE						

9. **Understanding the Definitions**. We have a problem X and we have proved that SAT reduces to X (formally, SAT $\leq_m$ X). Which one of the following statements can we derive?

   (a) X belongs to **NP**, but we do not know whether it is **NP**-complete.

   (b) X is an **NP**-complete problem.

   (c) None of the above.

   Now, a colleague has additionally found a reduction in the reverse direction, $X \leq_m$ SAT. What can we conclude now?

   (i) X belongs to **NP**, but we do not know whether it is **NP**-complete.

   (ii) X is an **NP**-complete problem.

   (iii) None of the above.

10. **True, false or open?**. Consider the following decision problems:

    - SAT: given a propositional formula, to determine whether it is satisfiable.

- COL: given a graph $G = (V, E)$ and a natural number k, to determine whether G can be painted with k colors so that the extremes of edges are painted with different colors.

- SOR: given an array of n different integers, to determine whether it is sorted in increasing order.

For each of the statements that follow, mark with an 'X' the corresponding column, depending on whether the statement is true, false, or is an open problem and it is still unknown if it is true or false.

	True	**False**	**Open**
SOR is in class P			
SOR is in class NP			
SOR is NP-hard			
SAT is in class P			
SAT is in class NP			
SAT is NP-hard			
SOR can be reduced polynomially to COL			
COL can be reduced polynomially to SOR			
SAT cannot be reduced polynomially to SOR			
COL can be reduced polynomially to SOR, and SAT cannot be reduced polynomially to SOR			

11. **DISEQUALITIES**. The problem of DISEQUALITIES consists in, given an interval of integers and a set of disequalities ($\neq$) between variables, to determine whether there is a solution to all disequalities with values in the interval. More formally, given:

    - a (finite) interval $[l, u] \subset \mathbb{Z}$,
    - a set of variables V, and
    - a set D of disequalities of the form $x \neq y$, where $x, y \in V$,

    we have to determine where there is $s : V \to \mathbb{Z}$ such that:

    - for any $x \in V$ we have $s(x) \in [l, u]$, and
    - for any $x \neq y \in D$ it holds $s(x) \neq s(y)$.

    We will consider that the set of variables is of the form $V = \{x_0, x_1, \ldots, x_{n-1}\}$ for a certain $n > 0$, and will identify the variables with integers between 0 and $n - 1$. Besides, we will represent the inputs for the problem of DISEQUALITIES by means of the type *inp_DISEQUALITIES* defined as follows:

```
struct inp_DISEQUALITIES {
  int l, u, n;
  set<pair<int,int>> D; // each pair (i,j) is a disequality x_i ≠ x_j
```

};

(a) Consider the problem of COLORING: given an undirected graph G and a natural number $c > 0$, to determine whether one can paint the vertices of G with c colors in such a way that any edge has the endpoints painted with different colors.

Suppose that we represent the inputs for the problem of COLORING with the type *inp_COLORING* defined as follows:

```
struct inp_COLORING {
  vector<vector<int>> G; // graph represented with adjacency lists
  int c;
};
```

Implement a polynomial reduction from COLORING to DISEQUALITIES:

```
inp_DISEQUALITIES reduction(const inp_COLORING& ec);
```

(b) Do you see feasible to find a polynomial algorithm for DISEQUALITIES? Why?

12. **One About Steffy**. Probably you will remember that Johnny and Roy organized an excursion with their big group of friends (and Steffy!) to spend a weekend in a holiday camp. Today they have decided to go rafting, but there is only one boat. Additionally, the boat is only safe if people aboard weigh exactly a total of T kilograms (because if they weigh more, the boat sinks and, if they weigh less, the stream is not strong enough to drag them to destination).

Read the following dialog assuming that all the characters say the truth.

JOHNNY: Roy, ask everybody's weight so that we can see if it's worth to rent a boat.

ROY: Here you have, you know I like to make lists of people.

STEFFY: Hey, guys... It's better you don't try. The problem you want to solve (given an integer T and given n strictly positive integers $p_1, \ldots, p_n$, to determine whether there are a few whose sum is T) is **NP**-complete.

JOHNNY: Damn, we've got a problem!

ROY: Wait! By coincidence, I have with me a magic lantern I bought at Istanbul market, in a shop where they speak English. They told me that if you rub, a genie will come out who can efficiently solve any **NP** problem.

JOHNNY: OK, come on, give it to me!

[Johnny rubs the lantern and a genie comes out in a smoke cloud.]

GENIE: Greetings, boy. If you give me m integers $b_1, \ldots, b_m$, I'll determine at the moment whether there's some nonempty subset with zero sum.

JOHNNY: Damn it, Roy! This is not our problem!

ROY: Hummm... Maybe that shopper in Istanbul cheated me.

STEFFY: Don't worry, guys. The shopper didn't lie... I have the solution!

(a) Explain which is the solution Steffy has found to know whether some people in the group can go rafting (that is, give a reduction from Johnny's problem to the genie's problem).

(b) Finish the proof that the genie's problem is **NP**-complete.

2

Greedy Algorithms

1. **Trip with children**. A family that travels with children goes from city A to city B by car at constant speed. The children must go to the restroom at intervals not longer than d kilometers. The route has n stopping points, being the first city A and the last city B. We are given the distance of each stopping point from city A. The distance between consecutive stopping points is at most d kilometers. A travel plan specifies at which intermediate stopping points the family should stop. Write an algorithm that computes a travel plan with the minimum number of stops. Justify the correctness of your algorithm and analyze its cost.

2. **Storing files**. We want to store a set of n files in a device with capacity L. The size of the i-th file is given by $l[i]$ ($1 \leq i \leq n$). We know that $\sum_{i=1}^{n} l[i] > L$. Design an algorithm that chooses a subset of the files that maximizes the number of files stored in the device. Justify the correctness of your algorithm and analyze its cost.

3. **Security company**. Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month. Each license is currently selling for a price of 100 €. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100 \cdot r_j^t$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the same price of 100 €).

   The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible? Give an algorithm that takes the n rates of price growth $r_1, r_2, \ldots, r_n$, and computes an order in which to buy the licenses so that the total amount of money spent is minimized. Prove the correctness and analyze the cost of your algorithm.

4. **Caching**. We consider a set U of n data items stored in main memory. We also have a faster memory, the *cache*, that can hold $k < n$ data items at any one time. It is assumed that the cache initially holds some set of k items. We are given a sequence of data items $D = d_1, d_2, \ldots, d_m$ drawn from U to be processed. When item d_i is processed, we can access it very quickly if it is already in the cache; otherwise, we are required to bring it from main memory into the cache and, if the cache is full, to evict some other item that is currently in the cache to make room for d_i. This is called a *cache miss*, and we want to have as few of these as possible. On a particular sequence of data items, an *eviction schedule* specifies which items should be evicted from the cache at which points in the sequence (hence determining the contents of the cache and the number of misses over time). Given a sequence S of data items and the initial contents of the cache, what is the eviction schedule that incurs as few cache misses as possible? Design an algorithm for solving this problem. Prove its correctness and analyze its cost.

5. **Fractional knapsack**. There is a collection of n objects, each of which has a (positive) *weight* and a (positive) *value*. The weight of the i-th object is w_i, and its value is v_i. We have a knapsack that can store weight W at most. The goal is to determine which objects to put in the knapsack so that the accumulated value is maximized while respecting the knapsack capacity. In this version of the problem, objects can be fragmented: we can put a *fraction* of an object into the knapsack (and then the value of this fraction is the proportional part). That is, we want to maximize $\sum_{i=1}^{n} x_i v_i$ subiect to $\sum_{i=1}^{n} x_i w_i \leq W$ and $0 \leq x_i \leq 1$, $x_i \in \mathbb{R}$ for all $1 \leq i \leq n$, where x_i is the fraction of the i-th obiect to be placed into the knapsack. Design an algorithm for solving this problem, prove its correctness and analyze its cost.

6. **Coin changing**. Consider the problem of making change for n cents using the fewest number of coins.

   (a) Describe a greedy algorithm to make change for coins of 25, 10, 5 and 1 cents. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have values that are consecutive powers, that is, are c^0, c^1, ..., c^k for some integers $c > 1$ and $k \geq 1$. Show that your greedy algorithm always yields an optimal solution in this case too.

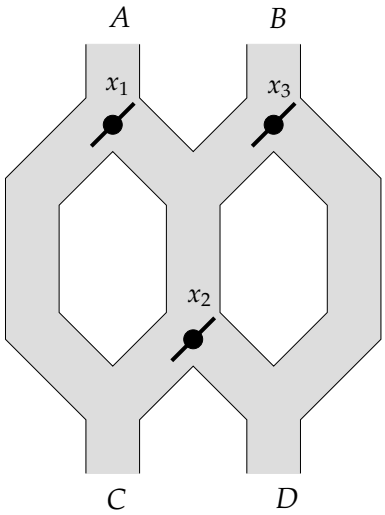
   (c) Find a combination of (integer) values for the coins so that your algorithm does not necessarily find the minimum number of coins for the change.

3

Finite Automata and Regular Expressions

1. **Playing with marbles**.

   The following figure represents a marble-rolling toy:



   A marble is dropped at A or B. Levers x_1, x_2 and x_3 cause the marble to fall either to the left or to the right. Whenever a marble encounters a lever, it causes the lever to reverse **after** the marble passes, so the next marble will take the opposite branch.

   Model this toy with a finite automaton. Let the symbols A and B represent the point at which the marble is dropped. A sequence of marbles if accepted if the last marble of the sequence exits at D (and not accepted when it exits at C).

2. **From informal description to DFA**.

   Give DFA's accepting the following languages over the alphabet $\{0, 1\}$:

(a) The set of all strings ending in 00.

(b) The set of all strings beginning with a 1 that, when interpreted as a binary integer, is a multiple of 5. For example, strings 101, 1010, and 1111 are in the language.

3. **Verification of a DFA**.

   Consider the DFA with the following transition table:

	0	1
$\rightarrow A$	A	B
$*B$	B	A

   where the initial state is marked with a $\rightarrow$ and accepting states with $*$.

   Informally describe the language accepted by this DFA, and prove by induction on the length of an input string that your description is correct.

   Hint: When setting up the inductive hypothesis, it is wise to make a statement about what inputs get you to each state, not just what inputs get you to the accepting state.

4. **From NFA to DFA**.

   Convert the following NFA to a DFA:

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\emptyset$
$*s$	$\{s\}$	$\{s\}$

   where the initial state is marked with a $\rightarrow$ and accepting states with $*$.

5. **From informal description to NFA**.

   Give a non-deterministic finite automaton that accepts the language of all strings over the alphabet $\{0, 1, \ldots, 9\}$ such that the final digit has appeared before.

   Try to take advantage of nondeterminism as much as possible.

6. **From informal description to DFA via NFA**.

   Give a non-deterministic finite automaton that accepts the language over the alphabet $\{a, b, c, d\}$ of the strings that end in *abc*, *abd*, or *aacd* .

   Convert the NFA into a DFA.

7. **From λ-NFA to DFA** .

   Consider the following λ-NFA:

	λ	a	b	c
$\rightarrow p$	$\emptyset$	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	$\emptyset$
$*r$	$\{q\}$	$\{r\}$	$\emptyset$	$\{p\}$

   where the initial state is marked with a $\rightarrow$ and accepting states with $*$.

(a) Compute the λ-closure of each state.

(b) Give all the strings of length three or less accepted by the automaton.

(c) Convert the automaton into a DFA.

8. **Writing regular expressions**. Write regular expressions for the following languages:

   (a) The set of strings over the alphabet $\{a, b, c\}$ containing at least one a and at least one b.

   (b) The set of all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.

9. **An informal description**.

   Give an informal description of the language of the regular expression $(1 + \lambda)(00^*1)^*0^*$.

10. **From regular expressions to λ-NFA**.

    Convert the regular expression 01^* to an NFA with λ-transitions.

11. **Adding transitions to a λ-NFA**.

    Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a λ-NFA such that there are no transitions into q_0 and no transitions out of q_f. Describe the language accepted by each of the following modifications of A, in terms of $L = L(A)$:

    (a) The automaton constructed from A by adding a λ-transition from q_f to q_0.

    (b) The automaton constructed from A by adding an λ-transition from q_0 to every state reachable from q_0 (along a path whose labels may include symbols of Σ as well as λ).

12. **Minimizing DFA's**.

    Consider the following the transition table of a DFA:

	0	1
$\rightarrow A$	B	A
B	A	C
C	D	B
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

    where the initial state is marked with a $\rightarrow$ and accepting states with $*$.

    (a) Draw the table of distinguishabilities for this automaton.

    (b) Construct the minimum-state equivalent DFA.

13. **Uniqueness of minimum DFAs**.

(a) Construct a DFA with the minimum possible number of states that accepts the language of all strings over the alphabet $\{0,1\}$ that do not have two consecutive 1's.

(b) Construct a regular expression for the same language.

(c) Starting from the regular expression you have constructed, build a λ-NFA automaton that recognizes the same language. Then, convert it into a DFA with the minimum posssible number of states and check that it is the same automaton you constructed in (a).