Exercise sheet 11.

Advanced Algorithms

WiSe 2020/21

Instructors: László Kozma, Mahmoud Elashmawi

Due 18:00, February 7th, 2021

Exercise 1 Maximum flow *4 pts*

Consider a network G with m edges and n vertices, where $m \geq n$. Show that if all edges have capacity 1, then the maximum s-t flow can be computed in time $O(mn^{2/3})$.

Exercise 2 Cycles *4 pts*

Given is a directed graph G. Describe an efficient algorithm that finds a collection C of directed cycles in G, such that every vertex of G is contained in *exactly one* cycle of C (or reports that no such collection of cycles exists). Singleton vertices are not considered cycles, so a cycle is of length at least two. Note that such a collection C may not be unique.
Hint: model as bipartite matching.

Exercise 3 NP *4 pts*

Which of the following problems are in NP? Explain your answer.

(a) Bipartite Matching
Input: a bipartite graph G.
Output: *yes* if G has a perfect matching, *no* otherwise.

(b) Long Path
Input: an undirected graph G, an integer k.
Output: *yes* if G contains a simple path of length at least k, *no* otherwise.

(c) Halting
Input: A Turing machine T and its input x.
Output: *yes* if T halts for x, *no* otherwise.

(d) Graph Isomorphism
Input: two undirected graphs G, H.
Output: *yes* if G and H are isomorphic (i.e. identical up to re-ordering of the vertices), *no* otherwise.

Exercise 4 Satisfiability *14 pts*

(a) (3 pts) Consider the following boolean functions and argue (separately for each) whether there is a satisfying assignment (i.e. an assignment of values T/F to the variables, so that the function evaluates to T):
$\Phi_1 = (x \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{y} \vee \bar{z})$,
$\Phi_2 = (x \vee y) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee z)$,
$\Phi_3 = (x \vee y \vee z) \wedge (\bar{y} \vee t) \wedge (\bar{t} \vee \bar{y}) \wedge (y \vee z \vee t)$.

(b) (2 pts) Review the definitions of disjunctive normal form (DNF) and conjunctive normal form (CNF) of a boolean function. Argue that the satisfiability of a function in DNF can be verified in polynomial time.

(c) (2 pts) Recall that the input to SATISFIABILITY is a CNF boolean function. Argue that every boolean formula can be transformed to an equivalent DNF formula. Why does this not give (in light of the previous question) a polynomial-time algorithm for SATISFIABILITY?

(d) (3 pts) Argue that if we had a polynomial-time algorithm for deciding whether an arbitrary CNF-formula is satisfiable, then we could also find, in polynomial time, a satisfying assignment.

(e) (4 pts) Show that SATISFIABILITY is polynomial-time solvable, if each variable may appear in at most two clauses.

Total: 26 points. Have fun with the solutions!

From: Yumeng Li and Thore Brehmer

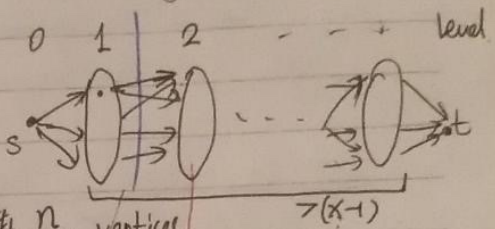
Exercise 1 Maximum flow *4 pts*

Consider a network G with m edges and n vertices, where $m \geq n$. Show that if all edges have capacity 1, then the maximum s-t flow can be computed in time $O(mn^{2/3})$.

Ex Sheet 11.

For a network G w/ all capacity 1. Dimits blocking flow step takes $O(m)$ time

Suppose we did x iterations, then there will be more than x levels in the level graph. So there is at least one level in the graph with less than $\frac{n}{x-1}$ vertices.



Then there are at most.

$(n-\frac{n}{x-1}) \cdot \frac{n}{x-1}$ capacity between these two levels

Say there are $< \frac{n}{x-1}$ vertices in this level.

Then the cut between this level and its neighboring level have $< \frac{n}{x-1} \cdot (n - \frac{n}{x-1})$ edges.

$\Rightarrow$ remaining flow $< \frac{n}{x-1}(n - \frac{n}{x-1})$.

remaining iterations $< \frac{n}{x-1}(n - \frac{n}{x-1})$.

total iterations $\leq x + \frac{n}{x-1}(n - \frac{n}{x-1})$.

$= (x-1) + \frac{n}{x-1}(n - \frac{n}{x-1}) + 1$.

$= t + \frac{n^2}{t} - \frac{n^2}{t^2} + 1$. $[t \leq (x-1)]$.

~~When $t = \frac{n^2}{t}$ it reaches optimum.~~

~~$\Rightarrow \frac{t^3}{t} = n^2(t-1) \Rightarrow t$~~

$t = \frac{n^2}{t} - \frac{n^2}{t^2}$

where is the rest? 2

Exercise 2 Cycles *4 pts*

Given is a directed graph G. Describe an efficient algorithm that finds a collection C of directed cycles in G, such that every vertex of G is contained in *exactly one* cycle of C (or reports that no such collection of cycles exists). Singleton vertices are not considered cycles, so a cycle is of length at least two. Note that such a collection C may not be unique.
Hint: model as bipartite matching.

Idea: search for strongly connected components (cycles)

$O(|V|^3)$ 1. run Floyd Warshalls alg on G and save the resulting matrix

$O(|V|^2)$ 2. For each $x, y \in$ vertice pair test with the matrix:

$d(x,y) \neq \infty$ and $d(y,x) \neq \infty$ and $x \neq y$

$\Rightarrow$ vertex x and y part of one strongly connected component (cycle)

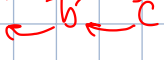
(remember that x and y are part of one cycle, if later a new vertice z is part of one strongly connected component with x and y $\Rightarrow$ x, y, z are part of a cycle)

3. If there is a vertex in no strongly connected component
($\Leftrightarrow \exists x \in$ vertice, $\forall y \in$ vertice : $d(x,y) = \infty$)

return False

$\Rightarrow O(|V|^3)$

- Works because Floyd Warshalls alg will compute all shortest pairs between all vertices or ∞ if there is no path.
- If there is no path from x to y $\Rightarrow$ there is no cycle between x and y
- So we just have to test for every vertice, if there is a path to another vertice

counterexample: 

Exercise 3 NP *4 pts*

Which of the following problems are in NP? Explain your answer.

(a) Bipartite Matching
Input: a bipartite graph G.
Output: *yes* if G has a perfect matching, *no* otherwise.

a) certificate : C = a perfect matching graph := $G'(V, E)$ (vertices)

verifier: A = - test if all vertices and edges of G' are also present in G. if not: return False $O(n)$

```
for v in V:
    if degree(v) ≠ 1:
        return False
return True
```
$O(n)$

Works as each vertice in a perfect matching has 1 edge.

$O(n) \in O(poly)$

(b) Long Path
Input: an undirected graph G, an integer k.
Output: *yes* if G contains a simple path of length at least k, *no* otherwise.

b)

certificate : C = a $\geq k$ long path : (V, E, s)

verifier : A = - test if all vertices and edges of the path are also present in G. if not: return False $O(n)$

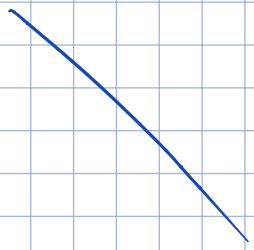
- counter = 0
visited = []

follows the path and tests if it contains a cycle $O(n)$

```
while s.next() ≠ null and path not in visited:
    visited += [s]
    counter += 1
    s = s.next()
if counter ≥ k:
    return True
else:
    return False
```

$O(n) \in O(poly)$

(c) HALTING
Input: A Turing machine T and its input x.
Output: *yes* if T halts for x, *no* otherwise.



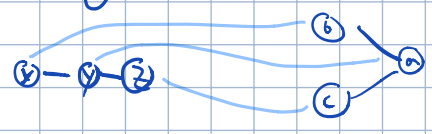
Question:

unsure if the [redacted] marked clause in (a, b, d) is even needed in NP verifiere. Can we just expect the certificate to give a valid output and remove the [redacted] clause? This is just a matter of definition. To be on the safe side always do these checks you marked in purple, since the are poly-time anyways.

(d) GRAPH ISOMORPHISM
Input: two undirected graphs G, H.
Output: *yes* if G and H are isomorphic (i.e. identical up to re-ordering of the vertices), *no* otherwise.

certificate : C = a matching order of the vertices of G and H in which they are isomorph

e.g. G: H: matching:



G' := [x, y, z]
H' := [b, a, c]

verifier : A = test for G' if the vertices and edges are present in G. (same with H') if not: return False

here it is not needed, since you implicitly check that in the for-loop

```
for i in range(len(G')):
    if degree(G'[i]) ≠ degree(H'[i]):
        return False
return True
```

missing: check if both have the same number of vertices

Works because if the graphs are in a matching order, all the vertices of the graph have the same amount of edges, if the graphs are isomorph

Exercise 4 Satisfiability *14 pts*

(a) (3 pts) Consider the following boolean functions and argue (separately for each) whether there is a satisfying assignment (i.e. an assignment of values T/F to the variables, so that the function evaluates to T):
$\Phi_1 = (x \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{y} \vee \bar{z})$,
$\Phi_2 = (x \vee y) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee z)$,
$\Phi_3 = (x \vee y \vee z) \wedge (\bar{y} \vee t) \wedge (\bar{t} \vee \bar{y}) \wedge (y \vee z \vee t)$.

ϕ_1 True for $(x,y,z) \overset{e.g.}{=} (x,1,0)$

$\phi_1 = (x \vee 1 \vee 0) \wedge (\bar{x} \vee 1 \vee \bar{0}) \wedge (\bar{1} \vee \bar{0})$

$= \quad 1 \wedge 1 \wedge 1 = 1$

ϕ_2 True for $(x,y,z) \overset{e.g.}{=} (1,0,1)$ (or $(0,1,0)$)

$\phi_2 = (1 \vee 0) \wedge (\bar{0} \vee \bar{1}) \wedge (\bar{1} \vee 1)$

$= 1 \wedge 1 \wedge 1 = 1$

ϕ_3 True for $(x,y,z,t) \overset{e.g.}{=} (1,0,1,t)$

$\phi_3 = (1 \vee 0 \vee 1) \wedge (\bar{0} \vee t) \wedge (\bar{t} \vee \bar{0}) \wedge (0 \vee 1 \vee t)$

$= 1 \quad \wedge 1 \wedge 1 \wedge 1 = 1$

3

(b) (2 pts) Review the definitions of disjunctive normal form (DNF) and conjunctive normal form (CNF) of a boolean function. Argue that the satisfiability of a function in DNF can be verified in polynomial time.

$CNF = (x_1 \vee x_2 .. \vee x_n) \wedge \ldots \wedge (x_1 \vee x_2 .. \vee x_n)$

$DNF = (x_1 \wedge x_2 .. \wedge x_n) \vee \ldots \vee (x_1 \wedge x_2 .. \wedge x_n)$

CNF is a conjunction of disjunctions
DNF is a disjunction of conjunctions

$n :=$ number of ~~disjunction~~ con
$k :=$ number of literals in each ~~disjunction~~ con

Algorithm: (Input DNF as nested list)

```
for ~~disjunction~~ con in DNF:
    temp = []
    for literal in ~~disjunction~~ con:
        if ¬literal in temp:
            break
        else:
            temp += [literal]
    return True   (if one valid disjunction has been found)
return False   (if each disjunction is unsolvable)
```

Works because in DNF only one ~~disjunction~~ con needs to be True
And a ~~disjunction~~ con is solvable $\Leftrightarrow \forall l \in$ ~~disjunction~~ con, $\bar{l} \notin$ ~~disjunction~~ con

(c) (2 pts) Recall that the input to SATISFIABILITY is a CNF boolean function. Argue that every boolean formula can be transformed to an equivalent DNF formula. Why does this not give (in light of the previous question) a polynomial-time algorithm for SATISFIABILITY?

Because transforming a CNF to a DNF will produce 2^n clauses (in the worst case) ⇒ it will still take exponential time to solve

E.g. Converting $(x_1 \cup y_n) \wedge (x_2 \cup y_2) \wedge \dots \wedge (x_n \cup y_n)$ will take 2^n terms

2

(d) (3 pts) Argue that if we had a polynomial-time algorithm for deciding whether an arbitrary CNF-formula is satisfiable, then we could also find, in polynomial time, a satisfying assignment.

$CNF = (x_1 \cup x_2 .. \vee x_n) \wedge \dots \wedge (x_1 \cup x_2 .. \vee x_n)$

We can only decide if the CNF-formula is satisfiable when we try all possible literals combinations (worst case), as every term is important for the satisfiability, as each term is connected by a conjunction, different from DNF in which terms are connected by disjunctions

⇒ we have to test all possibilitys

⇒ we also find / test a satisfying assignment

- if we could solve it in poly time

⇒ we could also find a satisfying assignment in poly time

2

you would need to formally prove that statement (if you could, you would also prove that SAT ∉ P, and thus P ≠ NP !!, since there can be exponentially many combinations in the worst case)

(e) (4 pts) Show that SATISFIABILITY is polynomial-time solvable, if each variable may appear in at most two clauses.