Algorithm Design - Exam
01-02-2024, *Time: 120 minutes*
Prof. Stefano Leonardi, Dr. Federico Fusco

The first two problems correspond to the first partial exam, the third and the fourth to the second one.

Problem 1

We are given two strings A and B. A string S is a subsequence of A if it can be obtained by deleting characters of A. For example, the string *abaacba* is a substring of both *abcaacabba* and *aabaabcba*. The longest common subsequence problem (LCS) asks for a string S of maximum length such that S is a subsequence of both A and B.

(a) Give the recursive form used in the dynamic programming formulation of LCS. Specifically consider how the length of the optimal substring $OPT[i; j]$ up to position i in string A and position j in string B can be expressed as a function of $OPT[i-1; j]$, $OPT[i; j-1]$ and $OPT[i-1; j-1]$.

(b) Provide an efficient implementation of the dynamic programming that computes the value of the length of the LCS using a table of size $|A| \times |B|$. Hint: (The running time should be $|A| \times |B|$.)

(c) Provide an efficient implementation of the algorithm that returns the string S solution of the LCS problem.

Problem 2

(a) Show an execution of the Ford and Fulkerson algorithm for the Max-Flow problem that requires $O(C)$ computations of an augmenting path where C is the maximum capacity of an edge.

(b) Illustrate the reason why the algorithm cannot be considered polynomial time in the input data to the problem.

(c) Show the execution of an alternative algorithm that runs in polynomial time on the same instance and analyse its running time on a general instance of the max-flow problem.

Problem 3

(a) *Give the formal definition of polynomial-time reduction from a decision problem X to a decision problem Y and prove the transitivity of such notion, i.e., if A is polynomial-time reducible to B and B is polynomial-time reducible to C, then A is polynomial-time reducible to C.*

(b) *Provide the definition of NP-completeness and give an example of an NP-complete problem. Define the min-cut and max-cut decision problems and discuss whether it is in NP or not.*

(c) *Define the subset-sum and the 3-d matching problems. Are these two problems in NP? Provide a reduction from 3-d matching to subset sum.*

Problem 4

A vertex cover of an undirected graph $G = (V, E)$ is a subset V' of V such that if (u, v) is an edge in G, then either $u \in V'$ or $v \in V'$ (or both). The size of a vertex cover is the number of vertices in it. The optimization version of the vertex-cover problem is to find a vertex cover of minimum size in a given undirected graph. We call such a vertex cover an optimal vertex cover.

(a) Define the decision version of the vertex cover problem, and argue about its NP-completeness.

(b) Design and analyze a polynomial time greedy algorithm for the optimization version of the vertex cover problem that always outputs a vertex cover of size at most twice the optimal one.

(c) Construct an example where the size of the vertex cover output by the greedy algorithm is not optimal.

Problem 1

We are given two strings A and B. A string S is a subsequence of A if it can be obtained by deleting characters of A. For example, the string *abaacba* is a substring of both *abcaacabba* and *aabaabcba*. The longest common subsequence problem (LCS) asks for a string S of maximum length such that S is a subsequence of both A and B.

(a) Give the recursive form used in the dynamic programming formulation of LCS. Specifically consider how the length of the optimal substring $OPT[i; j]$ up to position i in string A and position j in string B can be expressed as a function of $OPT[i-1; j]$, $OPT[i; j-1]$ and $OPT[i-1; j-1]$.

(b) Provide an efficient implementation of the dynamic programming that computes the value of the length of the LCS using a table of size $|A| \times |B|$. Hint: (The running time should be $|A| \times |B|$.)

(c) Provide an efficient implementation of the algorithm that returns the string S solution of the LCS problem.

a) A[1 n], B[1..m] OPT[i,j] = LCS BETWEEN A[1 i] AND B[1..j]

OPT[0,j] = 0 ∀j OPT[i,0] = 0 ∀i

IF A[i] ≠ B[j]

OPT[i,j] = max (OPT[i-1,j], OPT[i,j-1])

ELSE IF A[i] = B[j]

OPT[i,j] = OPT[i-1,j-1] + 1

b) |A| × |B| TABLE

```
LCS_ALG:
  n = |A| , m = |B|

  CREATE OPT [0..n] [0. m]

  FOR i = 0 TO n
     OPT [i] [0] = 0
  FOR j = 0 TO m
     OPT [0] [j] = 0

  FOR i = 0 TO n
    FOR j = 0 TO m
      IF A[i] = B[j]
         OPT [i] [j] = OPT [i-1] [j-1] + 1
      ELSE
         OPT [i] [j] = max (OPT [i-1] [j], OPT [i] [j-1])

  RETURN OPT [n] [m]
```

0 0 0 0 0 ⋯
0
0
⋮

c)

```
LCS_STRING:

    LCS_ALG
    S = ∅
    i = |A| , j = |B|

    WHILE i > 0 ∧ j > 0:
      IF A[i] == B[j]
        S.APPEND (A[i])
        i -= 1
        j -= 1
      ELSE IF OPT [i-1][j] ≥ OPT [i][j-1]
        i -= 1
      ELSE
        j -= 1

    RETURN STRING (S)
```

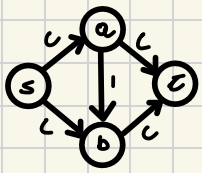
Problem 2

(a) Show an execution of the Ford and Fulkerson algorithm for the Max-Flow problem that requires $O(C)$ computations of an augmenting path where C is the maximum capacity of an edge.

(b) Illustrate the reason why the algorithm cannot be considered polynomial time in the input data to the problem.

(c) Show the execution of an alternative algorithm that runs in polynomial time on the same instance and analyse its running time on a general instance of the max-flow problem.

a)



HERE THE MAX-FLOW = 2C (s-a-t, s-b-t), BUT THE ALG FF CHOOSE THE AUGMENTING PATH s-a-b-t, THAT REQUIRE IN TOTAL C COMPUTATIONS, BECAUSE THE EDGE a-b HAS CAPACITY 1 (BOTTLENECK).

b) CAPACITIES IN INPUT ARE GIVEN IN BINARY. THERE ARE $\log_2 C$ BIT FOR EACH CAPACITY.
$O(C)$ ITERATIONS ARE EQUAL TO $O(2^{\log C})$, SO EXPONENTIAL IN THE INPUT.

c) WE CAN USE EDMOND-KARP ALGORITHM, WHICH ALWAYS CHOOSE THE SHORTEST AUGMENTING PATH (WITH BFS). HERE THE DISTANCE FROM s TO ALL OTHER NODES IN THE RESIDUAL GRAPH NEVER DECREASES AND THE DISTANCE FROM s TO t INCREASES AFTER AT MOST $O(m)$ AUGMENTATIONS.

$O(nm)$ AUGMENTATIONS, $O(m)$ BFS COST $\rightarrow$ $O(nm^2)$

Problem 3

(a) Give the formal definition of polynomial-time reduction from a decision problem X to a decision problem Y and prove the transitivity of such notion, i.e., if A is polynomial-time reducible to B and B is polynomial-time reducible to C, then A is polynomial-time reducible to C.

(b) Provide the definition of NP-completeness and give an example of an NP-complete problem. Define the min-cut and max-cut decision problems and discuss whether it is in NP or not.

(c) Define the subset-sum and the 3-d matching problems. Are these two problems in NP? Provide a reduction from 3-d matching to subset sum.

a) $X \leq_p Y$ IF $\exists$ A FUNCTION f CALCULABLE IN POLYNOMIAL TIME SUCH THAT FOR EACH INSTANCE $x \rightarrow x \in X \Leftrightarrow f(x) \in Y$.
WE TRANSFORM AN x INSTANCE IN A Y INSTANCE. SO IF WE CAN RESOLVE Y, WE CAN ALSO RESOLVE X.

$A \leq_p B$, $B \leq_p C \rightarrow$ $a \in A \Leftrightarrow f(a) \in B$, $b \in B \Leftrightarrow f(b) \in C$

LET $h(x) = g(f(x))$ $\quad a \in A \Leftrightarrow f(a) \in B \Leftrightarrow g(f(a)) \in C$

SO $a \in A \Leftrightarrow h(a) \in C$ AND $A \leq_p C$

b) A PROBLEM X IS NP-COMPLETE IF:

- $X \in NP$
- $\forall y \in NP : Y \leq_p X$

AN EXAMPLE IS 3-SAT:

GIVEN A SET OF CLAUSES $C_1 .. C_k$, EACH OF LENGTH 3, OVER A SET OF VARIABLES $X = \{x_1 .. x_n\}$, DOES THERE EXISTS A SAT TRUTH ASSIGNMENT?

MIN CUT. GIVEN A GRAPH G AND A VALUE K, DOES THERE EXISTS A CUT S-T WITH CAPACITY AT MOST K?

IT'S IN P, AND SO IN NP, BECAUSE WE CAN CALCULATE THE MINCUT IN POL TIME VIA MAX FLOW

MAX CUT: GIVEN AN UNDIRECTED GRAPH G WITH WEIGHTS $w(e)$ AND A VALUE K, DOES THERE EXISTS A CUT $(S, V \setminus S)$ WITH $w(e)$ AT LEAST K?

IT'S NP-COMPLETE

c) SUBSET SUM: GIVEN A SET $S = \{s_1 .. s_n\}$ AND A TARGET x, DOES THERE EXISTS A SUBSET WITH SUM EXACTLY EQUAL TO x?

IT'S IN NP.

3D-MAT: GIVEN THREE DISJUNCTED SETS X, Y, Z WITH $T \subseteq X \times Y \times Z$, DOES THERE EXISTS A SET $M \subseteq T$ OF SIZE $q = |X| = |Y| = |Z|$ S.T. NO ELEMENT OF $X \cup Y \cup Z$ APPEAR IN MORE THAN ONE TRIPLE ?

IT'S IN NP

3DM $\leq_p$ SS

WE WANT TO REPRESENT A TRIPLE (x_i, y_j, z_k) AS A SINGLE NUMBER, S.T. THE SUM OF SOME TRIPLES IS EQUAL TO A TARGET VALUE, USING EACH ELEMENT ONLY ONCE.

SO 3DM HAS A PERFECT MATCHING IFF SUB SUM HAS A SOLUTION OF EXACTLY K.

Problem 4

A vertex cover of an undirected graph $G = (V, E)$ is a subset V' of V such that if (u, v) is an edge in G, then either $u \in V'$ or $v \in V'$ (or both). The size of a vertex cover is the number of vertices in it. The optimization version of the vertex-cover problem is to find a vertex cover of minimum size in a given undirected graph. We call such a vertex cover an optimal vertex cover.

(a) Define the decision version of the vertex cover problem, and argue about its NP-completeness.

(b) Design and analyze a polynomial time greedy algorithm for the optimization version of the vertex cover problem that always outputs a vertex cover of size at most twice the optimal one.

(c) Construct an example where the size of the vertex cover output by the greedy algorithm is not optimal.

a) GIVEN AN UNDIRECTED GRAPH $G=(V,E)$ AND A VALUE K, DOES THERE EXISTS A $V' \subseteq V$, WITH SIZE AT MOST K, S.T. $\forall e (u,v) \in E$, AT LEAST ONE END OF e IS IN V'?

IT'S IN NP SINCE THERE IS AN EFFICIENT CERTIFIER WITH WHICH WE CAN CHECK A SOLUTION.
IT'S NP-COMPLETE SINCE INDIPENDENT SET $\leq_p$ VERTEX COVER, INDEED A SET $S \subseteq V$ IS INDIPSET IFF NO EDGE HAS EACH OF ITS ENDS IN S.

b) A GREEDY ALG THAT ENSURES A 2-APPROX

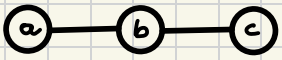
```
GR-ALG:

  C = ϕ
  WHILE EXISTS AN EDGE e=(u,v):
    ADD u AND v IN C
    REMOVE INCIDENT EDGES TO u AND v
  RETURN C
```

AN OPT SOLUTION TAKS AT LEAST ONE END FOR EACH EDGE OF THE MATCHING M, SO $OPT \geq |M|$

GR-ALG TAKES TWO NODES FOR EACH EDGE $\rightarrow |C| = 2|M|$

$$|C| = 2|M| \leq 2 \cdot OPT$$

c) 

$OPT = 1$, BECAUSE IT CHOOSE b.

IF GR-ALG CHOOSE $\{a, b\}$, IT ADDS a AND b, SO $|C| = 2$ X