Esame di Fondamenti di informatica II - prova di Modelli (2 h)
5 novembre 2019 - appello straordinario

Linguaggi e parsing

1. Si consideri il linguaggio L con alfabeto {a,b}, tale che ogni occorrenza di 'a' è immediatamente seguita da almeno due occorrenze di 'b'; stringhe che appartengono al linguaggio sono abbb, b, babb; ab e aabbbb non appartengono al linguaggio.

1.1. Definire una grammatica che genera il linguaggio.

1.2. Stabilire il tipo (stretto) del linguaggio, giustificando l'asserzione. (N.B. Scrivere una grammatica di tipo t non implica l'inesistenza di una grammatica equivalente di tipo t+1).

2. Si scriva una grammatica che genera il linguaggio delle espressioni booleane scritte correttamente (operatori: $\vee$, $\wedge$, $\neg$; usare il generico terminale *i* per le variabili; se necessario è possibile usare le parentesi tonde e i due simboli speciali λ (stringa nulla) e ε (stringa vuota)).

3. Definire il concetto di grammatica ambigua fornendo e discutendo un esempio di grammatica ambigua. Discutere brevemente la necessità di avere grammatiche non ambigue per definire linguaggi di programmazione.

4. Data un grammatica libera dal contesto illustrarele condizioni per affermare che è in forma LL(1). Discutere inoltre i vantaggi nel processo di costruzione dell'albero sintattico quando la grammatica è in forma LL(1).

Complessità

5. Definire con precisione le classi P ed NP e descrivere il problema P vs NP ("la classe P coincide con la classe NP?").

6. Per ciascuno dei risultati (ipotetici) di seguito descritti, descrivere l'impatto che avrebbe sulla questione P vs NP.

6.1. Determinazione di un upper bound $O(n^6)$ per il problema 3-COLORABILITÀ.

6.2. Individuazione di un lower bound $\Omega(2^{n/3})$ del problema VERTEX-COVER.

6.3. Costruzione di un upper bound esponenziale per 3-SAT.

Pattern matching

7. Si consideri il seuente programma di pattern matchng che verifica se esiste .

```
    Algorithm BoyerMooreMatch(T, P)  {T testo, P pattern}
  L ← lastOccurenceFunction(P, S )
  i ← m - 1
  j ← m - 1
  repeat
  if T[i] = P[j]
          if j = 0
                  return  i  { match at i }
          else
                  i ← i - 1
                  j ← j - 1
          else
          { character-jump }
          l ← L[T[i]]
          i ← i + m – min(j, 1 + l)
          j ← m - 1
     until  i > n - 1
     return  -1 { no match }
```

7.1. Descriverne l'esecuzione sulla ricerca del pattern *ababb* nella stringa
abaababababaabbbababbababb

7.2. Discutere la complessità del programma in funzione delle dimensioni T del testo e P del pattern.