ESERICIZI DI
"ALGORITMI E STRUTTURE DATI"

prof. Alan Bertossi
(svolti da **Michele Alberti** www.cs.unibo.it/~alberti)

Esercizio 17.2 (Zaino 0/1) :

Dati un insieme di n oggetti, con profitti {p1,..,pn} e volumi {v1,..,vn} tutti interi positivi, e due interi positivi ***c*** e ***k***, il problema dello ZAINO 0/1 consiste nel determinare un insieme {x1,..,xn} di interi uguali a 0 oppure 1 tali che la sommatoria dei profitti sia maggiore o uguale di k, e la sommatoria dei volumi sia minore o uguale a ***c***.

Svolgimento : i due insiemi dei profitti e dei volumi vengono memorizzati con due array di dimensione *1..n*. Si utilizza un array ***S*** di boolean per determinare se l'oggetto è stato preso oppure no.

```
type vettore = array[1..n] of integer;

procedure NDZAINO (var p,v : vettore; k,c : integer);
var  S : array[1..n] of boolean;
     profitto, capacita, i : integer;
begin

     for i := 1 to n do S[i] := choice({true, false});

     profitto := 0;
     capacita := 0;

     for i := 1 to n do
          if S[i] then begin
               profitto := profitto + p[i];
               capacita := capacita + v[i];
          end;
     if (profitto >= k) and (capacita <= c) then
          success
     else
          failure;
end;
```

Esercizio 6 dell'esame del 19-Set-2006 :

Dati un insieme ***A***, di *n* interi distinti, ed un intero *k*, si vuole decidere se esiste un sottoinsieme ***S*** di ***A*** tale che il prodotto degli elementi in ***S*** sia uguale a *k*. Scrivere (in pseudo-codice) un algoritmo non deterministico che richieda tempo polinomiale.

Svolgimento : l'insieme **A** viene memorizzato con un array di interi di dimensione *1..n*. Si utilizza un array ***S*** di boolean per determinare se l'intero è stato preso oppure no.

```
type insieme = array[1..n] of integer;

procedure NDPROSOTTOINSIEME (var A : insieme; k,n : integer);
var  S : array[1..n] of boolean;
     i, prodotto : integer;
begin
     for i := 1 to n do S[i] := choice({true, false});
     prodotto := 1;

     for i := 1 to n do
          if S[i] then
               prodotto := prodotto * A[i];
     if (prodotto = k) then
          success
     else
          failure;
end;
```

Esercizio 6 dell'esame del 19-Lug-2006 :

Dato un insieme ***A*** di *n* interi distinti, si vuole decidere se esiste un sottoinsieme ***S*** di ***A*** tale che il prodotto degli elementi in ***S*** sia uguale alla somma degli elementi in ***A – S***. Scrivere (in pseudo-codice) un algoritmo non deterministico che richieda tempo polinomiale.

Svolgimento : l'insieme **A** viene memorizzato con un array di interi di dimensione *1..n*. Si utilizza un array ***S*** di boolean per determinare se l'intero è stato preso oppure no.

```
type insieme = array[1..n] of integer;

procedure NDPRODOTTOvsSOMMA (var A : insieme; n : integer);
var  S : array[1..n] of boolean;
     i, prodotto, somma : integer;
begin
     for i := 1 to n do S[i] := choice({true, false});

     somma := 0;
     prodotto := 0;

     for i := 1 to n do
          if S[i] then
               prodotto := prodotto * A[i]
          else
               somma := somma + A[i];
     if (somma = prodotto) then
          success
     else
          failure;
end;
```

Esercizio 6 dell'esame del 12-Lug-2005 :

Dato un grafo non orientato ***G***=(N,A) un sottoinsieme ***S*** di nodi è coprente se ogni nodo di ***N – S*** è adiacente ad almeno un nodo di ***S***. Dati ***G*** ed un intero *k*, si scriva un algoritmo non deterministico di complessità polinomiale per trovare un sottoinsieme coprente di al più *k* nodi.

Svolgimento : Si presuppone che nessun valore del grafo sia uguale a zero. Si salva nell'array ***NODICO*** i valori dei nodi formanti l'insieme coprente. Per tutti gli altri si controlla se sono legati con almeno uno di quelli considerati coprenti. Se almeno uno di questi non è collegato con nessuno di quelli considerati coprenti allora *"failure"*, altrimenti si controlla se quelli considerati sono tanti quanti quelli richiesti in *k*.

```
procedure NDCOPRENTE (var G : grafo; k,n : integer);
var  T, S : array[1..n] of boolean;
     i, arco, ncoprente, j, h : integer;
     v : nodo;
     magagna : boolean;
     NODICO : array[1..n] of integer;
begin
     for i := 1 to n do begin
          NODICO[i] := 0;
          S[i] := choice({true, false});
          T[i] := false;
     end;
     ncoprente := 0;

     for i := 1 to n do
          if S[i] then begin
               ncoprente := ncoprente + 1;
               NODICO[ncoprente] := i;
          end;
     for i := 1 to n do begin
          if S[i] then begin
               for j := G.NODI[i] to G.NODI[i + 1] – 1 do
                    T[j] := true;
               T[i] := true;
          end else
          for j := G.NODI[i] to G.NODI[i + 1] – 1 do begin
               v := G.ARCHI[j];
               h := 1;
               while (h <= n) and (v <> NODICO[h]) do
                    h := h + 1;
               if (h <= n) then
                    T[i] := true;
          end;
     end;
     magagna := false;
     for i := 1 to n do if not(T[i]) then magagna := true;
     if magagna then failure else
          if (ncoprente = k) then success else failure;
end;
```