

Algoritmi e Strutture Dati

Elaborato a.a. 2023-2024

Prof.ssa Marina Zanella

PROBLEMA E MODELLAZIONE DEL DOMINIO


Problema: calcolo degli hitting set minimali

- Input: una collezione finita N di insiemi finiti dove gli elementi di ogni insieme appartengono al dominio M
- Output: tutti gli hitting set minimali di N

Hitting set e hitting set minimali

- **Hitting Set (HS)** di una collezione N di insiemi definita sul dominio M = insieme di elementi appartenenti a M che presenta una intersezione non vuota con ciascun insieme della collezione N
- **Hitting Set minimale (MHS)** = HS tale che nessun suo sottoinsieme è un HS


Calcolo dei MHS

- Il problema di decisione teso a stabilire se esista un HS di dimensione $\leq k$ per una collezione di insiemi data è NP completo [Karp 1972] [Garey and Johnson 1979]
- Il problema di ottimizzazione relativo al calcolo degli HS di cardinalità minima è una riformulazione equivalente del problema Set Covering, che è NP-hard 
- Un HS di cardinalità minima è un MHS ma non viceversa
- Il problema del calcolo dei MHS è spesso affrontato da algoritmi approssimati oppure da algoritmi che determinano solo i MHS di cardinalità minima
- In questo tema per elaborato sono proposti solo **algoritmi esatti completi** (cioè che calcolano tutti i MHS)

Calcolo dei MHS: esempio

$$N = \{ \{B3, B4\}, \\ \{A1, A2, B4\}, \\ \{A2, A5, B3, B4\} \}$$

$$M = \{A1, A2, A5, B3, B4\}$$

$$MHS = \{ \{B4\}, \\ \{A1, B3\}, \\ \{A2, B3\} \}$$


N.B. M coincide con l'unione degli insiemi della collezione N


Rilevanza dei MHS

- Usati (non solo) per
 - Convertire un'espressione booleana dalla forma normale disgiuntiva (DNF) a quella congiuntiva (CNF) e viceversa
 - Colpire un insieme di landmark nel planning AI
 - Isolare difetti nel codice di programmi software
 - Determinare le diagnosi (esse sono i MHS dei conflitti) negli approcci diagnostici AI basati sui modelli

Input: la matrice

Ogni elemento di M è univocamente identificato da un indice intero appartenente all'intervallo $[1 .. |M|]$


Esempio

- $M = \{A1, A2, B3, B4, A5\}$
- 1 2 3 4 5 

Input: la matrice (cont.)

Analogamente ogni elemento di N è univocamente identificato da un indice intero appartenente all'intervallo $[1 .. |N|]$

Esempio

$N = \{ \{B3, B4\},$	1	
$\{A1, A2, B4\},$	2	
$\{A2, A5, B3, B4\} \}$	3	

Input: la matrice (cont.)

- I dati d'ingresso del problema possono essere rappresentati come una matrice $A_{N,M}$, dove il valore del componente $a_{i,j}$ della matrice è 1 se l'elemento (di M) di indice j appartiene all'insieme (in N) di indice i , 0 altrimenti
- Assunzione: ogni riga della matrice contiene almeno un 1 (cioè nessun insieme della collezione N è vuoto)

Esempio: la matrice

{A1, A2, B3, B4, A5}

1 2 3 4 5

- {B3,B4}
- {A1,A2,B4}
- {A2,B3,B4,A5}

1

0	0	1	1	0
---	---	---	---	---

2

1	1	0	1	0
---	---	---	---	---


3

0	1	1	1	1
---	---	---	---	---

Spazio di ricerca (= spazio delle ipotesi)

- Per il calcolo dei MHS di una collezione N su un dominio M , lo spazio di ricerca è l'insieme (potenza) 2^M
- Indichiamo tale spazio con H e chiamiamo *ipotesi* ogni suo elemento

Poset (= Partially ordered set)


- Un poset è un insieme parzialmente ordinato da una relazione riflessiva, antisimmetrica e transitiva 
- Ad es. la relazione \subseteq fra insiemi è riflessiva, antisimmetrica e transitiva, pertanto **H è un poset** inerentemente a tale relazione

Preferenza

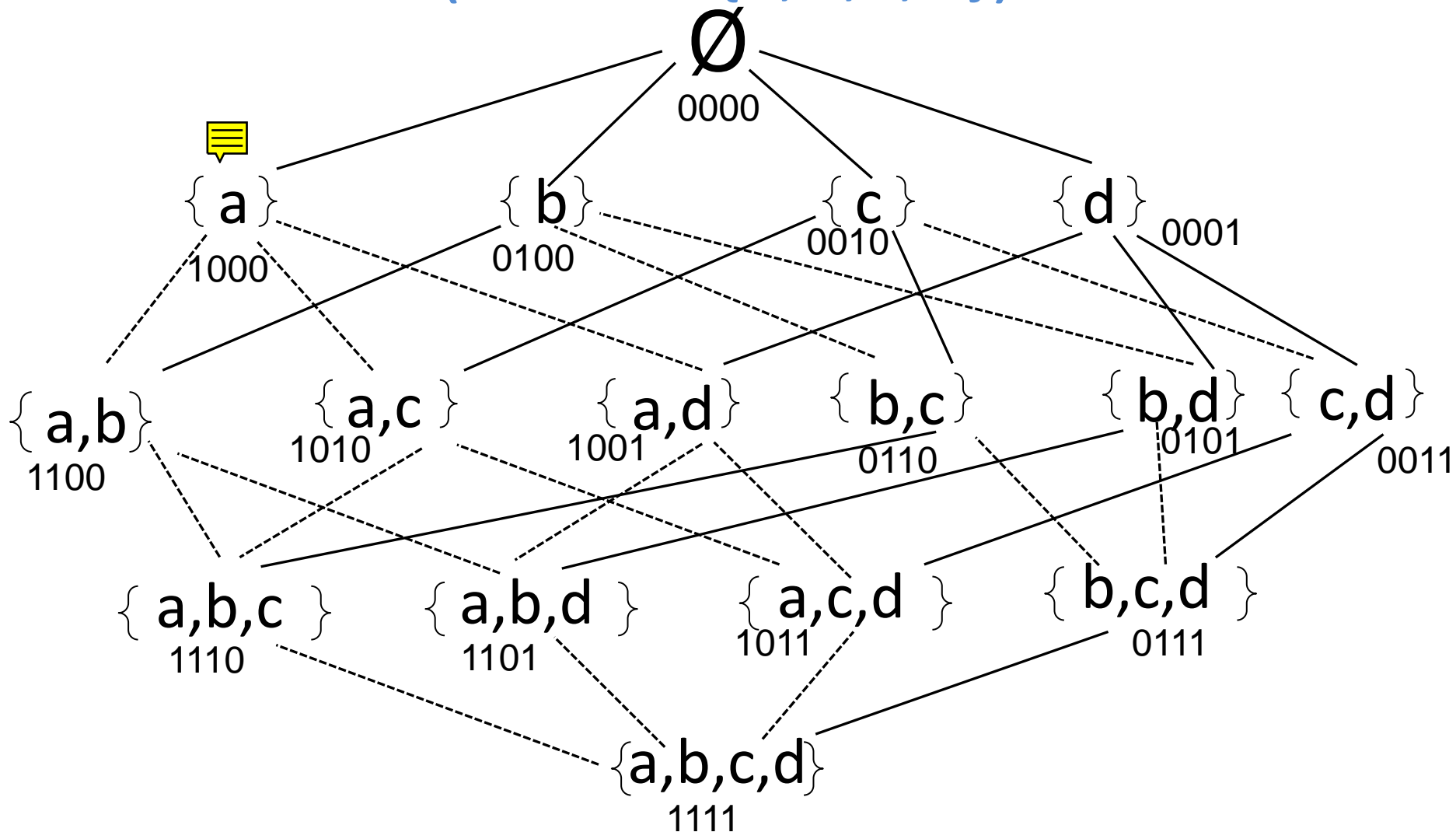
- La relazione \subseteq esprime una preferenza fra HS: dati due HS h_1 e h_2 , h_1 è preferibile ad h_2 sse $h_1 \subseteq h_2$
- Due ipotesi h_1 e h_2 per le quali né $h_1 \subseteq h_2$ né $h_2 \subseteq h_1$ si dicono **incomparabili**
- Il calcolo dei MHS corrisponde pertanto alla ricerca degli HS preferiti, cioè, indicato con **HS** l'insieme di tutti gli HS relativi all'istanza di problema assegnata, la soluzione di tale istanza di problema è **MHS** \subseteq **HS** dove
MHS = $\{hs \in \text{HS} \mid \forall h \in \text{HS}, h \subseteq hs \rightarrow h = hs\}$



Rappresentazione grafica dello spazio di ricerca (H)

- Ogni poset può essere rappresentato come un grafo orientato aciclico 
- Nel grafo rappresentante il poset H, ogni nodo è un'ipotesi, cioè un sottoinsieme di M, e ogni arco (implicitamente orientato dall'alto verso il basso) è una relazione \subset
- Il grafo rappresentante il poset H è caratterizzato da una struttura gerarchica a livelli (diagramma di Hasse)

Rappresentazione grafica di H (con $M=\{a,b,c,d\}$)



Cardinalità delle ipotesi nei livelli

- Tutte le ipotesi che appartengono allo stesso livello hanno la stessa cardinalità (la cardinalità di un'ipotesi h è indicata con $\text{card}(h)$)
- La cardinalità aumenta di un'unità passando da un livello a quello adiacente inferiore (partendo dalla cardinalità 0 che è quella del livello estremo superiore)
- Tutte le ipotesi distinte che appartengono allo stesso livello sono reciprocamente incomparabili
- Esiste una sola ipotesi in ciascuno dei due livelli estremi (essa è \emptyset per l'estremo superiore e M per l'estremo inferiore)



Successori e predecessori

- Si dicono **successori** di un'ipotesi h tutte le ipotesi a cui h è preferibile, cioè tutte quelle che nel grafo orientato sono raggiungibili a partire da h (ciascuna di esse è un superinsieme di h)
- L'insieme di tutti i successori di h si indica con **$\text{succ}^*(h)$**
- Con **$\text{succ}(h) \subseteq \text{succ}^*(h)$** si indica l'insieme dei **successori immediati** di h , ovvero i successori che appartengono al livello adiacente sottostante a quello di h
- I concetti duali rispetto a quelli di successore e successore immediato sono rispettivamente **predecessore** e **predecessore immediato**

Successori e predecessori (cont.)

- Nell'esempio di H con $M=\{a,b,c,d\}$ già illustrato graficamente,
 $\text{succ}^*(\{a,b\})=\{\{a,b,c\}, \{a,b,d\}, \{a,b,c,d\}\}$
 $\text{succ}(\{a,b\})=\{\{a,b,c\}, \{a,b,d\}\}$
- Casi particolari: l'ipotesi che si trova nel livello estremo superiore del grafo non ha predecessori mentre quella che si trova nel livello estremo inferiore non ha successori

Rappresentazione binaria

- Ogni ipotesi h è rappresentabile in forma binaria, indicata con $\text{bin}(h)$, mediante una sequenza di $|M|$ bit, ciascuno individuato da un indice che spazia da 1 a $|M|$
- Ciascun bit $\text{bin}(h)[j]$ indica l'appartenenza ad h dell'elemento di M di cui alla colonna di indice j della matrice (se il bit vale 1, l'elemento appartiene ad h , altrimenti no) → **il numero di istanze di 1 presenti in $\text{bin}(h)$ coincide con $\text{card}(h)$**
- Nella precedente illustrazione grafica di H con $M=\{a,b,c,d\}$ è stata fornita la rappresentazione binaria di ogni ipotesi
- Indichiamo con $\text{dist}(h_a, h_b)$ la **distanza di Hamming fra due** ipotesi h_a e h_b (aventi la medesima lunghezza di rappresentazione)

Teorema

- *La rappresentazione binaria di ciascuna ipotesi $h' \in \text{succ}(h)$ si ottiene complementando uno 0 in $\text{bin}(h)$*
- **Corollario.** *Il numero di successori immediati di h coincide con il numero di istanze di 0 presenti in $\text{bin}(h)$*
- **Corollario.** *La rappresentazione binaria di ciascun predecessore immediato di un'ipotesi h' si ottiene complementando un 1 in $\text{bin}(h')$*
- **Corollario.** *Il numero di predecessori immediati di h' coincide con il numero di istanze di 1 presenti in $\text{bin}(h')$, cioè coincide con $\text{card}(h')$*

Teoremi

- *Per ogni $h' \in \text{succ}(h)$ è vero che $\text{dist}(h, h') = 1$*
- *Date due ipotesi $h_a \in H$ e $h_b \in H$, con $h_a \neq h_b$, che condividono un successore immediato, è vero che $\text{dist}(h_a, h_b) = 2$*
- *Ciascuna coppia di ipotesi distinte appartenenti allo stesso livello di H condivide al più un successore immediato*

Aggiunta di regolarità ad H

- La regolarità del grafo rappresentante H viene aumentata ordinando le ipotesi appartenenti a ciascun livello. Ad es. se ogni elemento di M fosse un carattere, potremmo virtualmente ordinare il contenuto di ogni ipotesi secondo l'ordinamento alfabetico, ottenendo una stringa virtuale, e quindi ordinare alfabeticamente tutte le stringhe relative alle ipotesi del livello (è quello che è stato fatto nella precedente illustrazione di H con $M=\{a,b,c,d\}$)

Aggiunta di regolarità ad H (cont.)

- L'alfabeto con cui vengono costruite le ipotesi è M
- Tale alfabeto è stato implicitamente ordinato attraverso l'ordine delle colonne della matrice \rightarrow sfruttiamo questo ordinamento
- Il linguaggio a cui siamo interessati contiene solo stringhe internamente alfabeticamente ordinate e prive di ripetizioni (perché ogni stringa rappresenta un insieme): ciascuna di esse è identificata dalla rappresentazione binaria adottata
- Ad es. se $|M|=4$, la rappresentazione binaria della stringa che contiene solo l'elemento di M (idealmente è un carattere dell'alfabeto) relativo alla prima colonna è 1000, quella relativa all'elemento di cui alla seconda colonna è 0100, quella relativa all'elemento di cui alla terza è 0010 e infine quella relativa all'elemento di cui alla quarta è 0001 \rightarrow l'ordinamento del primo livello è $\langle 1000, 0100, 0010, 0001 \rangle$

Precedenza

- Date le rappresentazioni binarie di due ipotesi appartenenti allo stesso livello (e quindi contenenti lo stesso numero di 1), l'ipotesi il cui 1 più significativo occupa una posizione più a sx rispetto a quella del bit di valore 1 più significativo dell'altra precede l'altra; a parità di posizione del bit di valore 1 più significativo, la precedenza va controllata ricorsivamente come sopra sulla rappresentazione binaria di entrambe le ipotesi decurtata di tutti i bit fino a quello di valore 1 più significativo


Precedenza (cont.)

- Considerando il livello 4 di H con $|M|=7$, 1100011 precede 1011100 (addirittura non esistono altre rappresentazioni che, secondo l'ordine considerato cadono fra le due date, pertanto nel livello 4, 1011100 è il vicino di destra di 1100011)
- Considerando la rappresentazione binaria come un valore binario naturale, un'ipotesi h_1 precede un'ipotesi h_2 posta sullo stesso livello se $\text{bin}(h_1) > \text{bin}(h_2)$

Partizionamento dei successori immediati

- Si indica con $\text{succ}_L(h) \subseteq \text{succ}(h)$ l'insieme di successori immediati di h che possono essere successori immediati anche di ipotesi che sono a sx di h sullo stesso livello ma che non sono successori immediati di alcuna ipotesi a dx di h . Detto altrimenti, $\text{succ}_L(h)$ è l'insieme dei successori immediati di h per ciascuno dei quali h è il predecessore immediato posto più dx
- Per definizione, $\text{succ}_R(h) = \text{succ}(h) \setminus \text{succ}_L(h)$
- Nell'illustrazione grafica, la relazione fra h e ciascuna ipotesi appartenente a $\text{succ}_L(h)$ è visualizzata mediante un arco con linea continua (mentre le relazioni con le ipotesi appartenenti a $\text{succ}_R(h)$ sono raffigurate come archi in linea tratteggiata)

Partizionamento dei successori immediati (cont.)

- Per ogni ipotesi h' , esiste solo un'ipotesi h (appartenente al livello adiacente superiore) tale che $h' \in \text{succ}_L(h)$ 
- Questa proprietà è molto importante al fine di garantire che un'ipotesi h' venga generata (al più) una sola volta (a partire dall'unico predecessore immediato h tale $h' \in \text{succ}_L(h)$)
- Sia $h' \in \text{succ}_L(h)$. Il numero di predecessori immediati di h' non coincidenti con h è pari a $\text{card}(h') - 1$, cioè è pari a $\text{card}(h)$ (questa proprietà discende dall'ultimo corollario di pag. 20)

Teorema

- *La rappresentazione binaria di ciascuna ipotesi appartenente a $\text{succ}_L(h)$ si ottiene complementando uno 0 nei bit a sx del bit di valore 1 più significativo di $\text{bin}(h)$*
- Ad es. se $\text{bin}(h)=0001$, allora
 $\text{succ}_L(h)=\{h_1', h_2', h_3'\}$, dove (da sx a dx)
 $h_1'=1001$
 $h_2'=0101$
 $h_3'=0011$

Partizionamento dei successori immediati

- La rappresentazione binaria di ciascuna ipotesi h'' a sx di h che condivide un successore immediato h' appartenente a $\text{succ}_L(h)$ si ottiene complementando una delle occorrenze di 0 a sx del bit di valore 1 più significativo di $\text{bin}(h)$ – in questo modo si identifica $\text{bin}(h')$ – e quindi complementando una occorrenza di 1 a dx del bit appena complementato, fino a raggiungere il bit di valore 1 posto più a dx in $\text{bin}(h)$
- Ad es. se $\text{bin}(h)=0110$, allora esiste un unico $h' \in \text{succ}_L(h)$, dove $\text{bin}(h')=1110$, e due h'' , le cui rappresentazioni binarie sono 1010 e 1100



Terminologia



- Data un'ipotesi $h \neq \emptyset$ tale che $\text{bin}(h)[1] = 0$, chiamiamo **globalInitial(h)** l'ipotesi posta più a sx, sullo stesso livello di h , fra tutte quelle che condividono con h un successore immediato appartenente a $\text{succ}_L(h)$. La sua rappresentazione binaria si ottiene complementando, entro $\text{bin}(h)$, l'occorrenza di 0 che occupa la posizione di indice 1 e quindi complementando l'occorrenza meno significativa di 1
- Data un'ipotesi h e un'ipotesi $h' \in \text{succ}_L(h)$, con **initial(h,h')** e **final(h,h')** si indicano rispettivamente il predecessore immediato di h' più a sx e più a dx fra tutti i predecessori immediati di h' distinti da h (posti necessariamente a sx di h). Se h appartiene al primo livello di H , allora $\text{initial}(h,h') = \text{final}(h,h')$



Esempio

- Sia $\text{bin}(h)=0011$; allora
 $\text{succ}_L(h)=\{h_1', h_2'\}$, dove (da sx verso dx)
 $h_1'=1011$
 $h_2'=0111$
Le ipotesi poste sullo stesso livello di h , a sx di h , che condividono con h rispettivamente il successore immediato h_1' e h_2' sono (da sx verso dx)
 $h_{11}''=1010$ (questo è $\text{initial}(h, h_1')$)
 $h_{12}''=1001$ (questo è $\text{final}(h, h_1')$)
e
 $h_{21}''=0110$ (questo è $\text{initial}(h, h_2')$)
 $h_{22}''=0101$ (questo è $\text{final}(h, h_2')$)
- Si noti che h_{11}'' è anche $\text{globalInitial}(h)$

Su globalInitial



- Date due ipotesi h_1 e h_2 tali che $h_1 \neq \emptyset$, $h_2 \neq \emptyset$ e $\text{bin}(h_1)[1] = \text{bin}(h_2)[1] = 0$, appartenenti allo stesso livello, con $\text{bin}(h_1) > \text{bin}(h_2)$, allora $\text{globalInitial}(h_1) \geq \text{globalInitial}(h_2)$
- Due ipotesi h_1 e h_2 appartenenti allo stesso livello possono essere tali che $\text{globalInitial}(h_1) = \text{globalInitial}(h_2)$. In realtà, le ipotesi per cui vale suddetta uguaglianza possono essere più di due: sono tutte le t ipotesi h appartenenti allo stesso livello che condividono i $(\text{card}(h) - t)$ bit più significativi della loro rappresentazione binaria, dove i rimanenti t bit contengono un solo 1
- Esempio: date h_1 e h_2 , con $\text{bin}(h_1) = 00110$ e $\text{bin}(h_2) = 00101$ (le due ipotesi hanno il bit più significativo uguale a zero, appartengono allo stesso livello, $\text{bin}(h_1) > \text{bin}(h_2)$ e $t = 2$), è vero che $\text{globalInitial}(h_1) = \text{globalInitial}(h_2) = 10100$








CALCOLO DEI MINIMAL HITTING SET

Esplorazione di H

- La pagina che segue propone lo pseudocodice di un algoritmo che esplora lo spazio H cercando le soluzioni (per un qualsivoglia problema il cui dominio delle soluzioni sia H) per cardinalità non decrescente, evitando di esplorare ogni ipotesi che sia meno preferibile di una soluzione già trovata, assumendo come criterio di preferenza la relazione \subseteq

Algoritmo (template) principale

```

1: procedure MAIN
2:    $h_0 \leftarrow \emptyset$  ▷ top level hypothesis in  $H$ 
      ▷ the assignment operator sets the key field of  $h_0$ , typically  $bin(h_0)$ 
3:   SET_FIELDS( $h_0$ ) ▷ sets the additional fields of  $h_0$ 
4:    $current \leftarrow \langle h_0 \rangle$  ▷  $current$  is a sequence
5:    $\Delta \leftarrow \langle \rangle$  ▷ initially empty sequence containing all the solutions found so far
6:   repeat
7:      $next \leftarrow \langle \rangle$  ▷ empty sequence
8:     for each  $h$  in  $current$  (from left to right) do
9:       if CHECK( $h$ ) then ▷ checks whether  $h$  is a solution
10:        APPEND( $\Delta$ ,  $h$ ) ▷ appends  $h$  to sequence  $\Delta$ 
11:        remove  $h$  from  $current$  
12:      else if  $h = \emptyset$  then 
13:        APPEND( $next$ , GENERATE_CHILDREN( $h$ ))
14:      else if  $LM1(h) \neq 1$  then ▷  $LM1(h)$  is the index of the leftmost
      ▷ occurrence of 1 in  $bin(h)$ 
      ▷ a hypothesis whose most significant bit in  $bin(h)$  equals 1 has no children
      
15:         $h_s'' \leftarrow globalInitial(h)$  
16:        remove from  $current$  all hypotheses  $h_r$  s.t.  $bin(h_r) > bin(h_s'')$  
17:         $h_p \leftarrow$  first hypothesis in  $current$ 
18:        if  $h_p \neq h$  then 
19:          MERGE( $next$ , GENERATE_CHILDREN( $h$ )) 
20:         $current \leftarrow next$ 
21:   until  $current = \langle \rangle$ 
22:   return  $\Delta$ 

```

Moduli APPEND e MERGE

- APPEND “appende” il secondo parametro al primo, che è una sequenza, cioè inserisce l’elemento indicato come primo parametro in fondo (in coda) alla sequenza stessa
- MERGE “fonde” il secondo parametro, che è una sequenza ordinata, entro la sequenza ordinata indicata come primo parametro, cosicché al termine dell’operazione essa sia ordinata. Entrambe i parametri sono sequenze di ipotesi, ordinate per valori decrescenti delle rappresentazioni binarie delle ipotesi stesse

Generazione dei successori immediati


- Il modulo `GENERATE_CHILDREN(h)` genera solo ciascun successore immediato h' appartenente a $\text{succ}_L(h)$ i cui predecessori immediati si trovino tutti in *current* (se non tutti i predecessori immediati di h' sono in *current*, h' è successore di una soluzione già trovata e quindi non viene generato; in questo modo, il modulo effettua la potatura dello spazio di ricerca H)
- Alla riga 11 dello pseudocodice dell'algoritmo principale, l'eliminazione da *current* dell'ipotesi h (che è una soluzione) è tesa appunto alla potatura (non verranno generati i successori di h)

Cancellazioni entro *current*





- L'eliminazione delle ipotesi h_r di cui alla riga 16 del medesimo algoritmo è invece tesa alla sola riduzione dello spazio occupato da *current*. Vengono infatti eliminate ipotesi che si trovano all'inizio di *current*, quindi a sx dell'ipotesi h attualmente sotto analisi e pertanto già esaminate, che non sono predecessori immediati di alcun successore immediato di h (né delle ipotesi che seguono h). Questa cancellazione fa sì che, nello pseudocodice dell'algoritmo GENERATE_CHILDREN(h), di cui alla pagina successiva, l'esplorazione delle ipotesi che precedono h parta sempre dalla prima ipotesi in *current*
- La cancellazione progressiva di ipotesi contenute in *current* entro il ciclo *for* dell'algoritmo principale complica la codifica del ciclo stesso, dal momento che esso deve scandire (da sx a dx) gli elementi contenuti in *current*

Generazione dei successori immediati

- Lo pseudocodice di GENERATE_CHILDREN(h) prosegue nella pagina successiva

```
1: procedure GENERATE_CHILDREN( $h$ )
2:    $children \leftarrow \langle \rangle$ 
3:   if  $h = \emptyset$  then                                     ▷ top level hypothesis in  $H$ 
4:     for  $i \leftarrow 1$  to  $|M|$  do
5:        $bin(h') \leftarrow bin(h)$ 
6:        $bin(h')[i] \leftarrow 1$  
7:       SET_FIELDS( $h'$ )
8:       APPEND( $children, h'$ )
9:   return  $children$ 
```


Generazione dei successori immediati (cont.)

```
10:   $h_p \leftarrow$  first hypothesis in current
       $\triangleright$  the following loop scans (from left to right) every  $h' \in succ_L(h)$ 
11:  for  $i \leftarrow 1$  to  $(LM1(h) - 1)$  do
12:     $bin(h') \leftarrow bin(h)$  
13:     $bin(h')[i] \leftarrow 1$ 
14:    SET_FIELDS( $h'$ )
15:    PROPAGATE( $h, h'$ )   $\triangleright$  the setting of  $h'$  is progressively updated
       $\triangleright$  by propagating to  $h'$  some characteristics of its immediate predecessors
16:     $h''_i \leftarrow initial(h, h')$ 
17:     $h''_f \leftarrow final(h, h')$ 
18:     $cont \leftarrow 0$ 
       $\triangleright$   $cont$  is the counter of the immediate predecessors of  $h'$  in current,  $h$  excluded
19:    while  $bin(h_p) \leq bin(h''_i) \wedge bin(h_p) \geq bin(h''_f)$  do  $\triangleright$  comparisons between
       $\triangleright$  binary values;  $\wedge$  is a logical AND
20:       if  $dist(h_p, h') = 1 \wedge dist(h_p, h) = 2$  then
21:        PROPAGATE( $h_p, h'$ )
22:         $cont \leftarrow cont + 1$ 
23:         $h_p \leftarrow prox(h_p)$   $\triangleright$   $prox(h_p)$  is the hypothesis that
       $\triangleright$  comes immediately after  $h_p$  in current
24:      if  $cont = card(h)$  then 
25:        APPEND(children,  $h'$ )
26:  return children
```

Calcolo dei MHS

- L'algoritmo principale, unitamente al modulo `GENERATE_CHILDREN`, può essere adottato per portare a termine svariati compiti che richiedono l'esplorazione di H tenendo conto del criterio di preferenza scelto (\subseteq). La specializzazione relativa al compito da svolgere avviene assegnando un corpo opportuno ai moduli invocati (`PROPAGATE(h,h')`, `CHECK(h)` e `SET_FIELDS(h')`). Nel presente documento l'algoritmo viene sfruttato per il calcolo dei MHS

Vantaggi e svantaggi

- L'algoritmo proposto è anytime
- I MHS sono prodotti in ordine di cardinalità non decrescente
- Nello pseudocodice dell'algoritmo si è privilegiata la generalizzazione (cioè si è creato un algoritmo template sfruttabile per svolgere tanti compiti), a (minimo) discapito dell'efficienza (ad es. l'algoritmo esplora anche l'ipotesi \emptyset che nel calcolo dei MHS è inutile considerare)

Campo aggiuntivo

- (Ai soli fini del calcolo dei MHS) ogni ipotesi h è dotata di un campo aggiuntivo, indicato con $\text{vector}(h)$ che è un vettore di lunghezza $|N|$ rappresentante
 - la colonna j della matrice, se h è un singoletto che contiene solo l'elemento di M di indice j
 - il bitwise OR delle colonne della matrice relative a tutti gli elementi contenuti in h , altrimenti

(in entrambi i casi, il bit presente nella riga k del vettore vale 1 sse h ha una intersezione non vuota con l'insieme rappresentato dalla riga k della matrice)

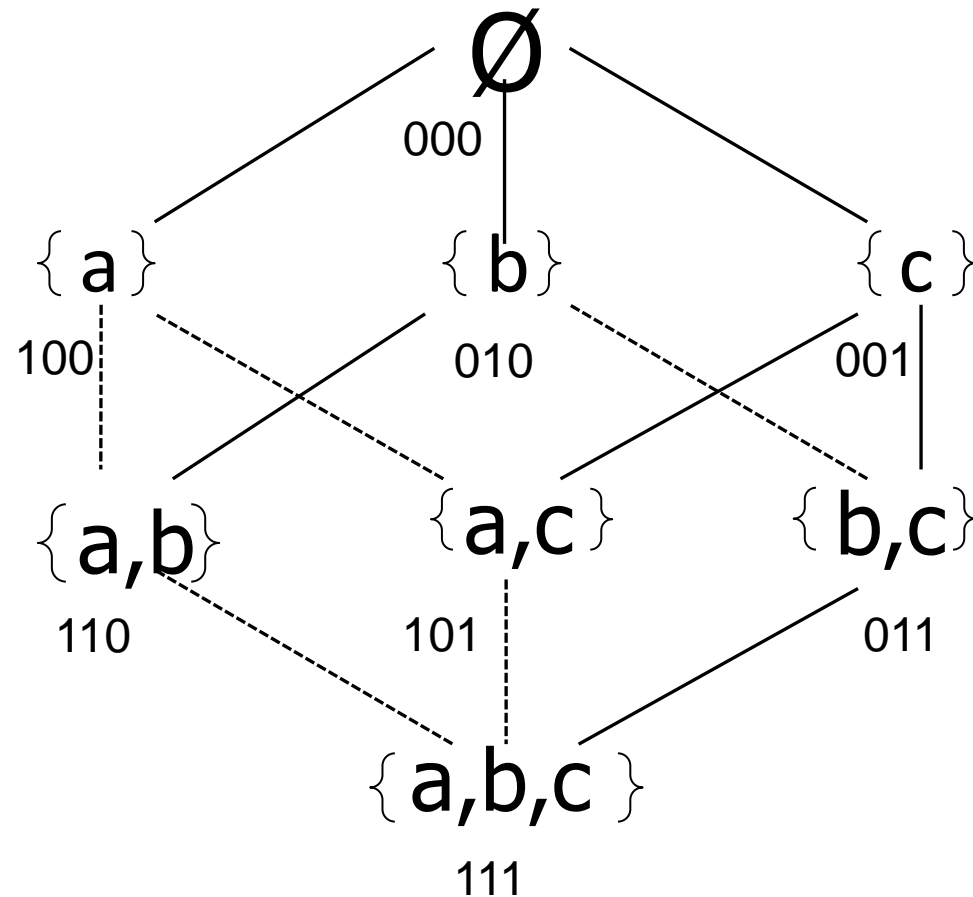
Moduli invocati

```
1: procedure SET_FIELDS( $h$ )
2:   if  $h$  is an immediate successor of hypothesis  $\emptyset$  then
3:      $vector(h) \leftarrow$  matrix column relevant to singleton  $h$ 
4:   else
5:      $vector(h) \leftarrow$  vector where all the  $|N|$  values are zeros
```

```
1: procedure PROPAGATE( $h, h'$ )
2:    $vector(h') \leftarrow vector(h') \vee vector(h)$   ▷  $h$  is an immediate predecessor of  $h'$   
▷ bitwise OR
```

```
1: procedure CHECK( $h$ )
2:   if  $vector(h)$  does not include any zero then
3:     return true
4:   else
5:     return false
```

Esempio di calcolo



	a	b	c
A	1		
B	1		1
C		1	
D	1	1	
E	1		1
F		1	
G	1		

Livello 1

- `current: <100, 010, 001>`
- `next: <>`


100: {a}

- Check: false



a
1
1
1
1
1

010: {b}

- Check: false
- Generated 110 : {a,b}
- next: $\langle 110 \rangle$ 



001: {c}

- Check: false
- Generated {a,c} and {b,c}
- next: $\langle 110, 101, 011 \rangle$

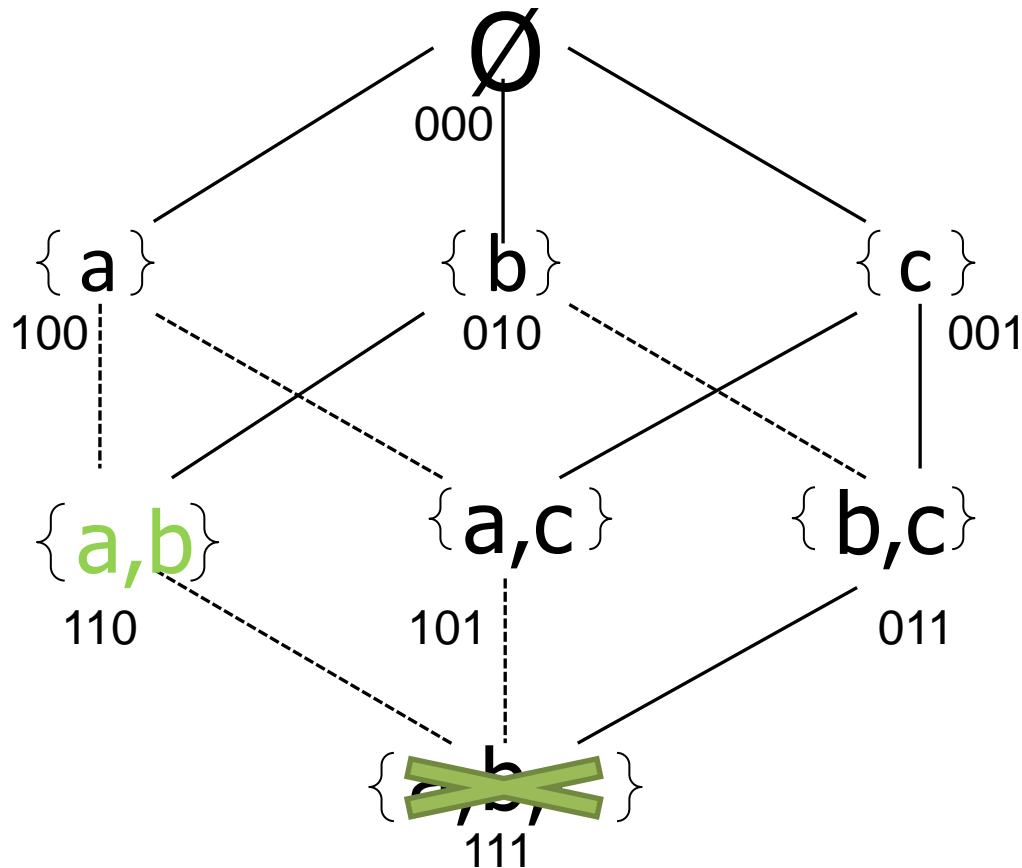


Livello 2

- current: $\langle 110, 101, 011 \rangle$
- next: $\langle \rangle$


110: {a,b}

- Check: true
- Insert 110 in Δ and remove it from current



a	b	$a \vee b$
1		1
1		1
	1	1
1	1	1
1		1
	1	1
1		1

101: {a,c}

- Check: false
- Non vengono inserite ipotesi in `next` perché non esistono successori immediati generabili a partire da 101 

a	c	avc
1		1
1	1	1
1		1
1	1	1
1		1

011: {b,c}

- Check: false
- Non vengono inserite ipotesi in `next` perché l'unico successore immediato generabile a partire da 011, cioè 111, è anche successore immediato di 110 e quest'ultima ipotesi non si trova in `current`, pertanto `next` resta vuoto

b	c	bvc
	1	1
1		1
1		1
	1	1
1		1
		1

Livello 3

- `A current` viene assegnato il contenuto di `next`
- `current` è vuoto, pertanto il processo termina

Compito 1

- Realizzare un'applicazione software che effettui il calcolo dei MHS secondo gli algoritmi proposti nelle pagine precedenti e rispetti i requisiti, funzionali e non, espressi nelle pagine seguenti
- Si noti che, nello pseudocodice fornito, il concetto di ipotesi e di rappresentazione binaria della stessa sono stati tenuti separati. Nel codice ogni ipotesi deve invece essere univocamente definita solo dalla sua rappresentazione binaria (cioè un'ipotesi coincide con la sua rappresentazione binaria)

Requisiti funzionali

- L'applicazione per il calcolo dei MHS deve:
 - acquisire in ingresso la matrice
 - produrre in uscita l'insieme di tutti i MHS relativi all'istanza di problema considerata (secondo l'ordine che è implicito nel metodo), unitamente a un **riassunto** che compendi le dimensioni della matrice d'ingresso, il numero totale dei MHS trovati e la loro cardinalità minima e massima
 - consentire all'utente di **interrompere il calcolo** dei MHS prima che esso sia concluso (o eventualmente di fissare una durata massima per l'elaborazione), fornendo in uscita tutti i MHS già calcolati (sempre secondo l'ordine di cui sopra) e un riassunto analogo a quello del punto precedente, esplicitando però che il calcolo non è stato completato e l'indice del livello di H in corrispondenza del quale l'esplorazione è stata interrotta (N.B. per indice di un livello di H si intende la cardinalità di tutte le ipotesi appartenenti a quel livello, anche se l'esplorazione del livello non è stata portata a termine)

Implementazione

- Il modulo SET_FIELDS considera come **parametro d'ingresso** (aggiuntivo **implicito**) una matrice che rappresenta una collezione di insiemi. Come sarà discusso nella sezione relativa alla sperimentazione, tale matrice è descritta per righe (entro un file di testo), mentre l'elaborazione compiuta da SET_FIELDS richiede di conoscere la matrice per colonne. È opportuno prendere in considerazione l'opzione di una **preelaborazione**. Inoltre nell'implementazione non è necessario che si manipoli effettivamente una matrice, intesa come array di array. Pertanto, al di là della struttura dati adottata, della «matrice» potrebbero eventualmente essere di volta in volta caricate da file le parti che servono. Il numero di accessi al file aumenterebbe il tempo di esecuzione, riducendo però l'occupazione di memoria centrale. Queste considerazioni sono da intendere solo come un invito a valutare più alternative, non come indicazioni (prescrittive) per lo sviluppo

Requisiti non funzionali

- La matrice (che è l'unico input del calcolo) è fornita, riga per riga, attraverso un **file di testo** avente il **formato** dei file resi disponibili per la sperimentazione
- L'**estensione** del nome del **file d'ingresso** è **matrix**

Requisiti non funzionali (cont.)

- L'uscita principale è costituita dall'insieme dei MHS calcolati, che deve essere fornito, un MHS dopo l'altro, attraverso un **file di testo** avente lo stesso **formato** del file d'ingresso (un MHS può infatti essere indicato esattamente come una riga di una matrice avente $|M|$ colonne)
- Il nome del **file di uscita** deve essere identico a quello del file d'ingresso ma la sua **estensione** deve essere mhs (anziché matrix)
- Il **riassunto** relativo all'uscita deve essere incluso come commento nel file che contiene i MHS

SPERIMENTAZIONE

Benchmark

- Sono resi disponibili due benchmark, ciascuno contenente file di testo (rispettivamente 1371 e 1400) di tipo matrix (attenzione: numerosi file sono condivisi fra i due)
- Ogni file rappresenta una matrice d'ingresso (le cui dimensioni possono arrivare fino a 3512 colonne)
- Ogni file .matrix è univocamente individuato dal suo nome (ad es. 74L85.026): la descrizione delle prove sperimentali condotte acquisendo in ingresso tale file deve riportare questo nome

Formato interno dei file .matrix

commenti

- ```
;;; Host = zelda6, Version = 26.1, date = 2010-01-13-17-08-09
;;; Source = /tilde/dekleer/projects/GDE/DXC/dxc-09-syn-
benchmark-1.1/74l85/74L85.000.scn
;;; Error status nil
;;; Injected fault:32(o2)
;;; Map 1(z1) 2(z2) 3(z3) 4(z4) 5(z5) 6(z6) 7(z7) 8(z8) 9(z9) 10(z10)
11(z11) 12(z12) 13(z13) 14(z14) 15(z15) 16(z16) 17(z17) 18(z18)
19(z19) 20(z20) 21(z21) 22(z22) 23(z23) 24(z24) 25(z25) 26(z26)
27(z27) 28(z28) 29(z29) 30(z30) 31(o1) 32(o2) 33(o3)
```


```
0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 -
0 1 0 1 0 -
```

righe della  
matrice; qui  
 $|N| = 2$

indice intero assegnato a  
ciascun elemento di M (fra  
parentesi); in questo  
esempio  $|M| = 33$

separatori  
delle righe  
della matrice

## Compito 2


- Sfruttare i benchmark forniti per condurre una sperimentazione, ovvero effettuare delle prove di funzionamento dell'applicazione per il calcolo dei MHS sviluppata (Compito 1) acquisendo in ingresso delle matrici contenute nei benchmark, nel rispetto dei requisiti funzionali e non indicati nelle pagine seguenti 
- Il primo fine della sperimentazione è quello di registrare e valutare criticamente le prestazioni spaziali e temporali delle prove condotte



# Requisiti non funzionali

- Si assuma che il file d'ingresso `.matrix` non indichi esplicitamente né il numero di righe né il numero di colonne (queste informazioni possono eventualmente comparire come commento nel file ma non possono essere sfruttate come parametri d'ingresso)

# Requisiti funzionali

- Nei file .matrix, il dominio  $M$  è spesso un superinsieme dell'unione degli insiemi della collezione  $N$  considerata (cioè una o più colonne della matrice in ingresso possono essere vuote). Al fine di ridurre le dimensioni dello spazio di ricerca, **si restringa il dominio da considerare nell'esplorazione a  $M' \subseteq M$** , dove  $M'$  contiene i soli elementi presenti negli insiemi della collezione  $N$  (ovvero si riconduca la matrice in ingresso a una matrice priva di colonne vuote), avendo cura tuttavia di rappresentare i MHS in uscita mediante una matrice a  $|M|$  colonne (secondo lo stesso ordine delle colonne della matrice in ingresso) 
- Nel riassunto che deve sempre essere presente nel file d'uscita, indicare anche il valore di  $|M'|$ ; in aggiunta, eventualmente indicare i numeri d'ordine delle colonne vuote 'soppresse'
- La cardinalità di un MHS non può mai superare il valore  $\max\{|N|, |M'|\}$ , pertanto l'esplorazione dello spazio  $H$  non deve mai oltrepassare il livello di indice pari a tale valore

## Requisiti funzionali (cont.)

- L'applicazione sviluppata deve registrare, nel riassunto di cui si è già parlato, anche i dati prestazionali relativi a ogni prova di esecuzione
- È positivo che l'applicazione sviluppata produca eventuali **informazioni intermedie** (ad es. il numero di ipotesi generate per ogni livello) che siano di interesse per la valutazione sperimentale

# Permutazioni

- Permutando le righe e/o le colonne della matrice in ingresso si devono ottenere in uscita gli stessi MHS (ATTENZIONE: la permutazione delle colonne è la più significativa ma comporta qualche complicazione relativa all'identificazione delle colonne stesse)
- Ciascun file .matrix usato nella sperimentazione può dar luogo a più prove di esecuzione, condotte su permutazioni diverse delle righe e/o delle colonne della matrice in ingresso (chiamate semplicemente permutazioni della matrice)

# Compito 3

- Realizzare un programma software che produca delle permutazioni di una matrice contenuta in un file .matrix
- Realizzare un programma software che confronti le uscite prodotte da prove di esecuzione dell'applicazione per il calcolo dei MHS sviluppata (Compito 1) condotte su una matrice contenuta nei benchmark e una sua permutazione o su due permutazioni diverse della medesima matrice in ingresso. Eventuali discrepanze nei valori delle uscite denunciano la presenza di errori (logici o di programmazione) nel risolutore e/o nello pseudocodice fornito in queste specifiche
- Si conduca una sperimentazione relativa al calcolo dei MHS di più permutazioni della medesima matrice, confrontando non solo le uscite prodotte ma anche le prestazioni (spaziali e temporali) riscontrate

# Riflessioni

- Come già sottolineato, i MHS relativi alle diverse permutazioni della stessa matrice d'ingresso sono gli stessi ma le prestazioni spaziali e temporali dell'applicazione che li calcola possono essere diversi. Identificare dei criteri di generazione delle permutazioni volti a **valutare l'andamento delle prestazioni al variare dell'ordinamento di righe e colonne** (in particolare, di queste ultime, visto che l'applicazione opera «per colonne») è un'interessante attività di ricerca scientifica

**RICHIESTE**

# Gruppi di lavoro

- Ogni gruppo, costituito da tre studenti, deve
  - svolgere i tre compiti descritti nelle sezioni precedenti. Nel caso eccezionale il lavoro venisse affrontato da un gruppo comprendente meno di tre studenti, non dovrà essere considerato il terzo compito
  - redigere una relazione scritta che illustri il lavoro svolto, le **scelte implementative** compiute, la **sperimentazione** condotta (su una selezione – a discrezione del gruppo – dei file .matrix forniti) e una **valutazione critica** degli esiti sperimentali



# Lavoro e relazione

- Particolare attenzione deve essere dedicata alla **scelta di strutture dati** volte a estendere nella misura maggiore possibile le dimensioni delle matrici che possono essere elaborate nonché di altri accorgimenti aventi lo stesso fine. La relazione deve documentare tali scelte e accorgimenti
- Sono naturalmente apprezzati gli sforzi tesi a ridurre il costo temporale della computazione, che devono anch'essi essere documentati
- È consentita qualsiasi forma di **riuso del software**
- La relazione deve contenere ogni indicazione ritenuta utile al fine di consentire l'utilizzo dei programmi realizzati e la conduzione di ulteriori sperimentazioni
- La relazione deve evidenziare tutte le limitazioni riscontrate nelle prove di esecuzione effettuate

# Requisiti non funzionali

- Non è richiesta la realizzazione di GUI: l'elaborazione dei programmi da realizzare deve essere batch (I/O solo da/per file)
- Non è richiesto il soddisfacimento di altri requisiti non funzionali oltre a quelli evidenziati nelle sezioni precedenti; in particolare, non è imposto un linguaggio di programmazione, né un ambiente di sviluppo né un ambiente di destinazione

# Consegna del materiale

- Ai fini del superamento della prova orale è necessario consegnare, entro i tempi indicati nelle note relative agli appelli, una cartella elettronica contenente relazione (in formato sia “sorgente”, sia pdf), codice sorgente ed eventuale codice eseguibile dei programmi software sviluppati, (l’indicazione di) tutti i file di ingresso su cui è stata condotta la sperimentazione unitamente ai corrispondenti file di uscita prodotti e ai relativi riassunti nonché eventuali file (ad es. Excel o Matlab) creati al fine di valutare gli andamenti delle prestazioni
- La consegna deve avvenire inviando via email il link a tale cartella, link creato usando un sistema di condivisione (ad es. dropbox, GDrive), oppure attraverso la piattaforma Moodle
- Il progetto descritto in questo documento deve essere presentato entro la sessione d’esame autunnale di Novembre 2025 (inclusa)