

Università di Torino  
*Dipartimento di Informatica*  
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

---

Corso  
“*Algoritmi e Complessità*”

DOMANDE D'ESAME

Luca Roversi

12 aprile 2022 (A.A.21/22)

# Indice

<i>A Brute-Force e Certificazione</i>	<b>2</b>
<i>B Backtrack</i>	<b>4</b>
<i>C Branch&amp;Bound</i>	<b>6</b>
D Domande relative a Problemi computazionali	<b>11</b>
Bibliografia	<b>12</b>

L'esame consisterà nell'argomentare la risposta ad un paio di domande, scelte a caso tra quelle elencate di seguito. L'idea è di non superare i 30/45 minuti di colloquio, globalmente.

**Perché domande note a priori?**

- Per prepararsi all'esame si suggerisce di confezionare materiale da illustrare durante l'orale come guida per argomentare le risposte.

È certamente possibile usare il materiale fornito durante il corso (porzioni di testo, figure, codice, etc. ).

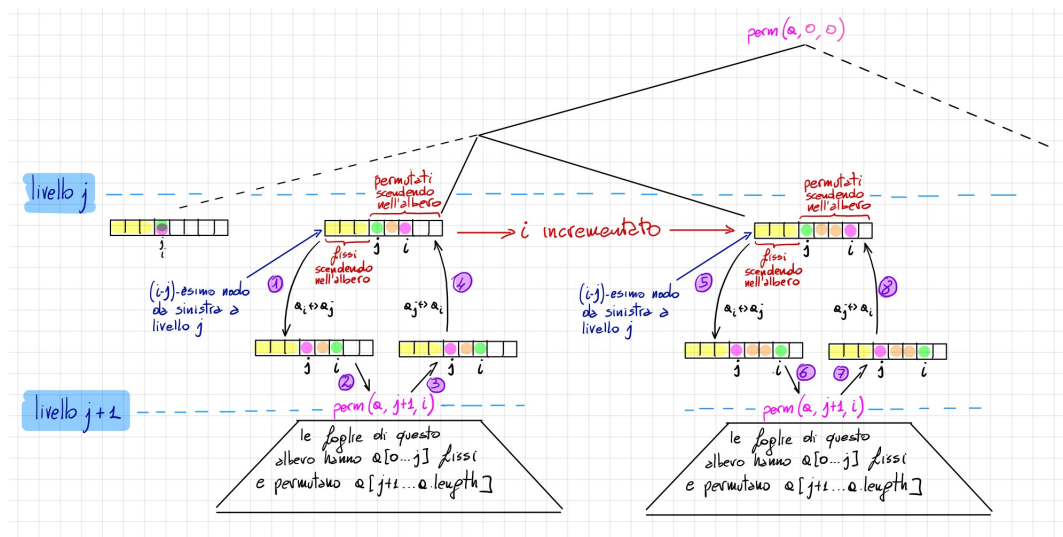
- Se si prevede produrre il materiale di cui al punto precedente, esso dovrà essere consegnato al docente entro la scadenza di iscrizione all'appello cui si intende partecipare.

Se non si consegnerà alcun materiale sarà possibile usare solo la lavagna e l'orale si svolgerà in presenza.

**Criteri di valutazione.**

- Riguardo all'esposizione, saranno apprezzate: sintesi, correttezza, chiarezza, ampiezza delle argomentazioni.
- Riguardo al materiale consegnato, verrà valutata la completezza, rispetto agli argomenti trattati durante il programma didattico, e la correttezza.

**Domanda A.1** La figura:



riassume graficamente la configurazione generica di un algoritmo che genera le *permutazioni* di una lista di elementi.

**Punto 1.** Argomentare a livello intuitivo sul perché la configurazione generica data in figura può essere usata per sintetizzare un algoritmo completo e corretto per la generazione delle permutazioni di una lista di elementi.

**Punto 2.** Modificare la figura precedente in modo da ottenere la configurazione generica di un algoritmo che genera *tutti i sottoinsiemi di elementi* di un array, giustificando correttezza e completezza a livello intuitivo.

**Punto 1** e **Punto 2** sono da considerarsi come domande alternative. □

---

**Domanda A.2** Fissare una nozione formale di *permutazione* e dimostrare formalmente che essa è riflessiva e simmetrica. ☐

**Domanda A.3** Dimostrare formalmente la completezza di un algoritmo di generazione di permutazioni di una lista di elementi. ☐

**Domanda A.4** La figura:

d	Voto max	Valutazione
1	2	2
2	5	4.6
3	4	3.9
4	2	2

Voto massimo ammesso: 9  
Votazione migliore: 8.6

riassume un'istanza del problema Valutazione.

**Punto 1** Descrivere una ricerca *Brute-Force* di una risposta in uno spazio degli stati organizzato come permutazioni.

**Punto 2** Descrivere una ricerca *Brute-Force* della risposta in uno spazio degli stati organizzato a sottoinsiemi.

**Punto 1 e Punto 2** sono da considerarsi come domande alternative. ☐

**Domanda B.1** Illustrare uno pseudo-algoritmo a piacere che implementa la tecnica *Backtrack*, giustificando le varie parti.  $\square$

**Domanda B.2** Illustrare la tecnica algoritmica *Backtrack* usando i problemi:

- Ordinamento di un insieme di numeri,
- Cammino Hamiltoniano,
- Colorazione di un grafo,
- Subsetsum.

Per ciascuno argomentare una *funzione bound*, prestando attenzione alla terminologia usata per descrivere le parti dello spazio degli stati.

Inoltre, relativamente al problema dell'ordinamento, argomentare sul perché è possibile evitare soluzioni basate su *Backtrack*.  $\square$

**Domanda B.3** La figure seguenti:

---

```

1  Algorithm Backtrack( $k$ )
2  // This schema describes the backtracking process using
3  // recursion. On entering, the first  $k - 1$  values
4  //  $x[1], x[2], \dots, x[k - 1]$  of the solution vector
5  //  $x[1 : n]$  have been assigned.  $x[ ]$  and  $n$  are global.
6  {
7      for (each  $x[k] \in T(x[1], \dots, x[k - 1])$ ) do
8      {
9          if ( $B_k(x[1], x[2], \dots, x[k]) \neq 0$ ) then
10         {
11             if ( $x[1], x[2], \dots, x[k]$  is a path to an answer node)
12             then write ( $x[1 : k]$ );
13             if ( $k < n$ ) then Backtrack( $k + 1$ );
14         }
15     }
16 }

```

$$T(x[1..k]) = \{v_1, v_{n_k} \mid v \text{ valori per } x[k]\}$$

$$B_k(x[1..k]) = \begin{cases} 1 & \text{se } x[1..k][k+1..n] \text{ può essere soluzione} \\ 0 & \text{altrimenti} \end{cases}$$

```

1  Algorithm !Backtrack( $n$ )
2  // This schema describes the backtracking process.
3  // All solutions are generated in  $x[1 : n]$  and printed
4  // as soon as they are determined.
5  {
6       $k := 1$ ;
7      while ( $k \neq 0$ ) do
8      {
9          if (there remains an untried  $x[k] \in T(x[1], x[2], \dots,$ 
10              $x[k - 1])$  and  $B_k(x[1], \dots, x[k])$  is true) then
11          {
12              if ( $x[1], \dots, x[k]$  is a path to an answer node)
13              then write ( $x[1 : k]$ );
14               $k := k + 1$ ; // Consider the next set.
15          }
16          else  $k := k - 1$ ; // Backtrack to the previous set.
17      }
18 }

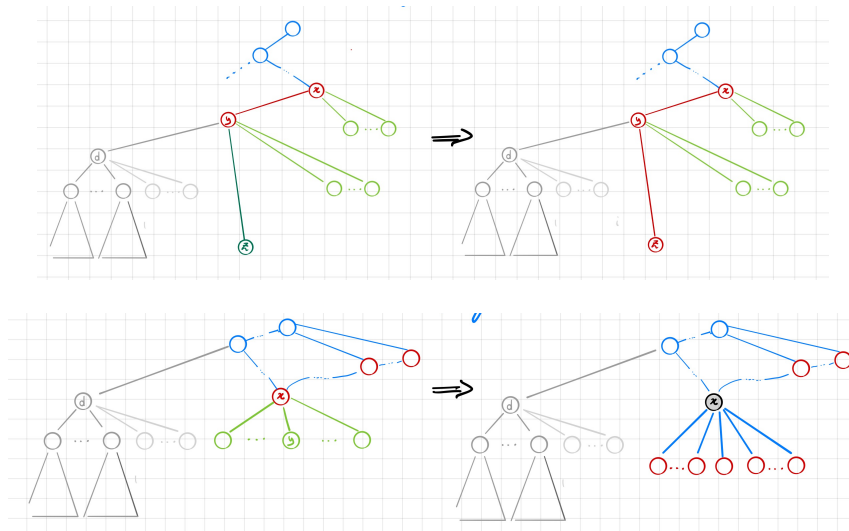
```

$$T(x[1..k]) = \{v_1, v_{n_k} \mid v \text{ valori per } x[k]\}$$

$$B_k(x[1..k]) = \begin{cases} 1 & \text{se } x[1..k][k+1..n] \text{ può essere soluzione} \\ 0 & \text{altrimenti} \end{cases}$$

riportano proposte in [HSR07] di pseudo-algoritmi di *Backtrack*. Commentare la funzionalità delle loro parti in base ad una qualche struttura dati fissata per la rappresentazione dello spazio degli stati su cui opera.  $\square$

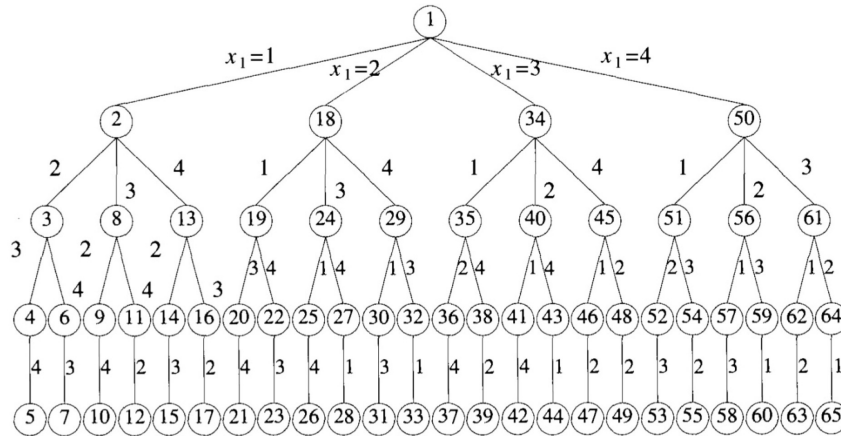
**Domanda C.1** Ad esempio, anche sfruttando almeno una delle seguenti immagini:



definire le nozioni *dead node*, *live node*, *E-node*, descrivere l'impatto delle politiche di generazione di *live node* e di assegnazione dello stato di *E-node* sulla visita dello spazio degli stati □

**Domanda C.2** L'albero degli stati del problema 4Regine è riportato in:





che è la Figura 7.2 de [HSR07].

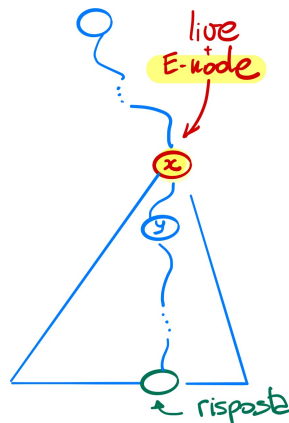
**Punto 1** Illustrare a grandi linee come si sviluppa la visita dello spazio degli stati qui sopra illustrato, che possiamo definire in profondità ed in ampiezza.

**Punto 2** Eventualmente usando lo spazio degli stati illustrato:

1. fornire un riscontro del fatto che visite in ampiezza non sono necessariamente più efficaci di quelle in profondità per individuare una risposta,
2. argomentare sulla inapplicabilità di criteri “ovvi” di *ranking* dei *live node*, per realizzare strategie di visita di uno spazio degli stati, né in ampiezza, né in profondità.

**Punto 1** e **Punto 2** sono da considerarsi come domande alternative. □

**Domanda C.3** Considerare, ad esempio, il seguente schema riassuntivo:



**Punto 1** Argomentare sul significato di una funzione costo per individuare una risposta, e ricostruire la struttura di una funzione costo approssimata.

---

**Punto 2** Dopo aver ricordato la struttura di una funzione costo approssimata che quantifichi sia il costo del lavoro di ricerca della risposta già effettuato, e che stimi quello ancora da fare, argomentare le conseguenze di considerare *nullo* il costo del lavoro già svolto.

**Punto 3** Dopo aver ricordato la struttura di una funzione costo approssimata che quantifichi sia il costo del lavoro di ricerca della risposta già effettuato, e che stimi quello ancora da fare, argomentare le conseguenze di considerare *non nullo* il costo del lavoro già svolto.

**Punto 1, Punto 2 e Punto 3** sono da considerarsi come domande alternative.  $\square$

**Domanda C.4** Inquadrare e descrivere la relazione tra le componenti della funzione costo  $\hat{c}(\cdot)$  e le strategie *Breadth-search* e *D-search* in uno spazio degli stati.  $\square$

**Domanda C.5** La seguente immagine:

```
listnode = record {
    listnode *next, *parent; float cost;
}

1  Algorithm LCSearch(t)
2  // Search t for an answer node.
3  {
4      if *t is an answer node then output *t and return;
5      E := t; // E-node.
6      Initialize the list of live nodes to be empty;
7      repeat
8      {
9          for each child x of E do
10         {
11             if x is an answer node then output the path
12                 from x to t and return;
13             Add(x); // x is a new live node.
14             (x → parent) := E; // Pointer for path to root
15         }
16         if there are no more live nodes then
17         {
18             write ("No answer node"); return;
19         }
20         E := Least();
21     } until (false);
22 }
```

riporta l'**Algoritmo 8.1** in [HSR07]. Commentarne il funzionamento in base ad una qualche struttura dati fissata per la rappresentazione dello spazio degli stati su cui opera.  $\square$

**Domanda C.6** Definire una *funzione costo* per il problema Subsetsum e discuterne le possibili interazioni con meccanismi FIFO/LIFO di accumulo dei *live node* e con criteri di *pruning*.  $\square$

---

**Domanda C.7** Definire una *funzione costo* per il problema **4Regine** e discuterne le possibili interazioni con meccanismi FIFO/LIFO di accumulo dei *live node* e con meccanismi di *pruning*.  $\square$

**Domanda C.8** Illustrare gli algoritmi **Greedy** e **Greedy-split**. Discutere la qualità della soluzione offerta da **Greedy** al variare del valore  $M$ , se applicato all'istanza **KP**:

$$(\vec{w} = (1, M), \vec{p} = (3, M), 2M) .$$

$\square$

**Domanda C.9** Illustrare gli algoritmi **Greedy-split** e **Ext-Greedy**. Discutere la qualità della soluzione prodotta da entrambi gli algoritmi, se applicati all'istanza **KP**:

$$(\vec{w} = (1, M), \vec{p} = (2, M), M) .$$

$\square$

**Domanda C.10** Illustrare l'algoritmo **Ext-Greedy**, inquadrare il significato della classe di algoritmi *k-approximation* e dimostrare che **Ext-Greedy** appartiene alla classe  $\frac{1}{2}$ -*approximation*. Argomentare sul perché **Ext-Greedy** non può appartenere a una classe di migliore di  $\frac{1}{2}$ -*approximation*.  $\square$

**Domanda C.11** Illustrare l'algoritmo **Greedy-LKP**, illustrandone il funzionamento su una istanza di **KP**, come, ad esempio:

$$(\vec{w} = (2, 3, 6, 7, 5, 4), \vec{p} = (6, 5, 8, 9, 6, 3), 10) , \quad (\text{C.1})$$

o anche più semplice, ed enunciarne la proprietà principale.  $\square$

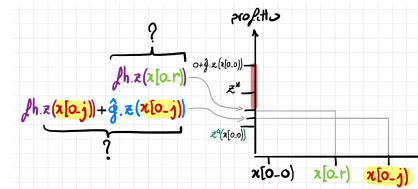
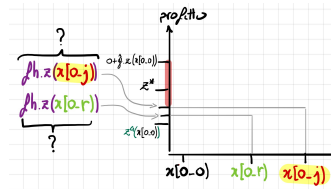
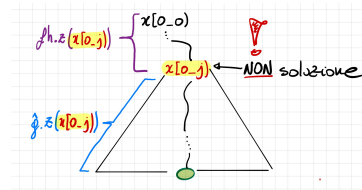
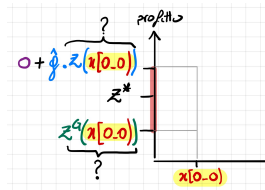
**Domanda C.12** Dimostrare almeno un caso dell'enunciato: “**Greedy-LKP** gode della proprietà *Greedy Choice Property*”.  $\square$

**Domanda C.13** Illustrare significato ed origine della sequenza di inclusioni tra profitti seguenti:

$$\hat{p} \leq z^G \leq z^* \leq \lfloor z^{\text{LP}} \rfloor \leq z^{\text{LP}} \leq \hat{p} + p_{\text{split}} \leq z^G + p_{\text{split}} .$$

$\square$

**Domanda C.14** Discutere la sintesi di un algoritmo *Branch&Bound* per **KP**, eventualmente sfruttando i seguenti diagrammi:



□

**Domanda D.1** Definire il problema decisionale **Subsetsum**, per un qualsiasi insieme di numeri positivi  $X$  ed un numero positivo  $s$ . Dimostrare che esso è un caso specifico del problema KP.  $\square$

**Domanda D.2** Definire i problemi KP e Bounded-KP. Dimostrare che sono equivalenti.  $\square$

- 
- [HSR07] E. Horowitz, S. Sahni e S. Rajasekaran. *Computer Algorithms*. 2nd. Disponibile per il *download* su <https://www.pdfdrive.com>. Summit, NJ, USA: Silicon Press, 2007.
- [KPP04] H. Kellerer, U. Pferschy e D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.