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

Corso
"*Algoritmi e Complessità*"

Domande d'esame

Luca Roversi

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

Indice

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.

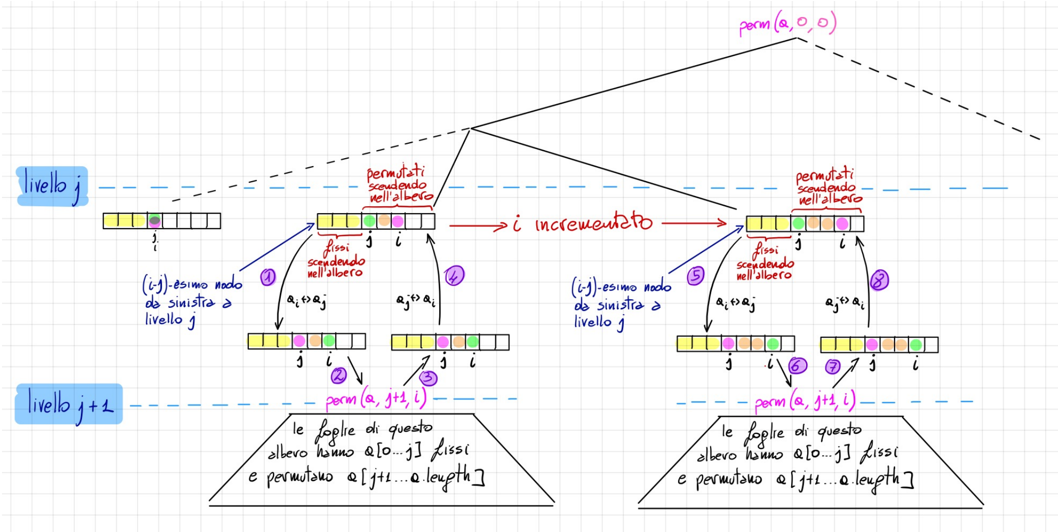
  Se non si consegnerà alcun materiale sarà possibile usare solo la lavagna e l'orale si svolgerà in presenza.

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.

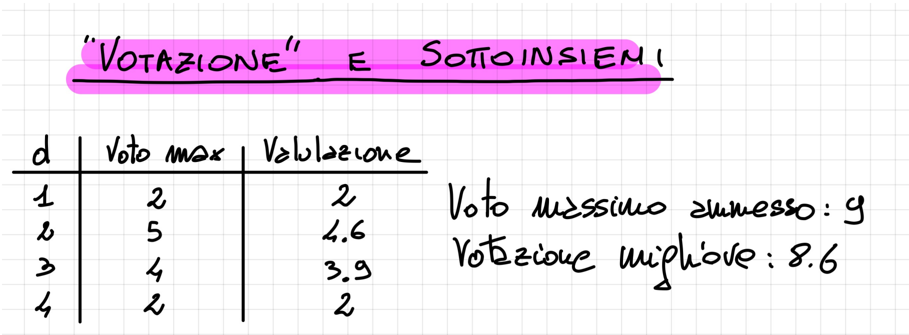
Punto 2. Modificare la figura precedente in modo da ottenere la configurazione generica di un algoritmo che genera *tutti i sottoinsiemi di elementi* di un *array*, giustificando correttezza e completezza a livello intuitivo.

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:



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. □

Backtrack B

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

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*. □

Domanda B.3 La figure seguenti:

```
Algorithm Backtrack(k)
// This schema describes the backtracking process using
// recursion. On entering, the first k − 1 values
// x[1], x[2], ..., x[k − 1] of the solution vector
// x[1 : n] have been assigned. x[ ] and n are global.
{
    for (each x[k] ∈ T(x[1], ..., x[k − 1]) do
    {
        if (B_k(x[1], x[2], ..., x[k]) ≠ 0) then
        {
            if (x[1], x[2], ..., x[k] is a path to an answer node)
                then write (x[1 : k]);
            if (k < n) then Backtrack(k + 1);
        }
    }
}
```

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

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

```
Algorithm IBacktrack(n)
// This schema describes the backtracking process.
// All solutions are generated in x[1 : n] and printed
// as soon as they are determined.
{
    k := 1;
    while (k ≠ 0) do
    {
        if (there remains an untried x[k] ∈ T(x[1], x[2], ...,
            x[k − 1]) and B_k(x[1], ..., x[k]) is true) then
        {
                if (x[1], ..., x[k] is a path to an answer node)
                    then write (x[1 : k]);
                k := k + 1; // Consider the next set.
        }
        else k := k − 1; // Backtrack to the previous set.
    }
}
```

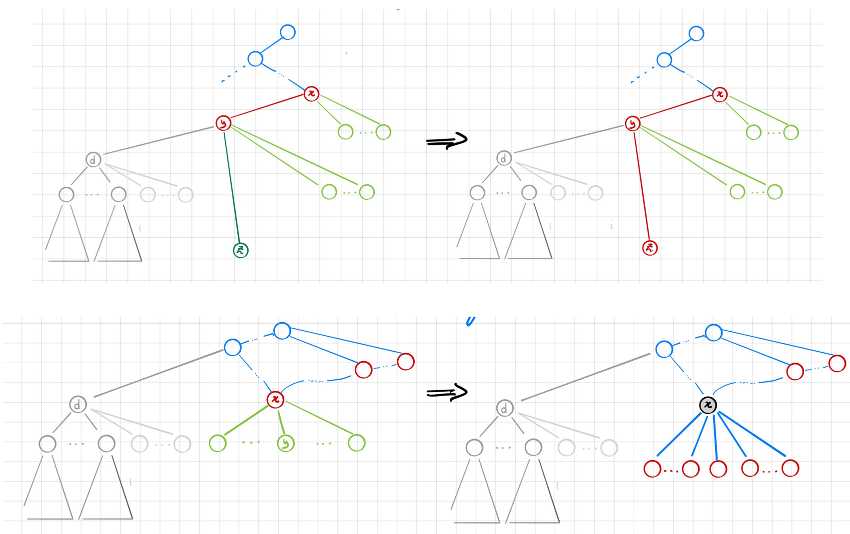
$T(x[1\dots k]) = \{v_1 \dots v_{n_k} \mid \text{valori per } x[k]\}$

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

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. □

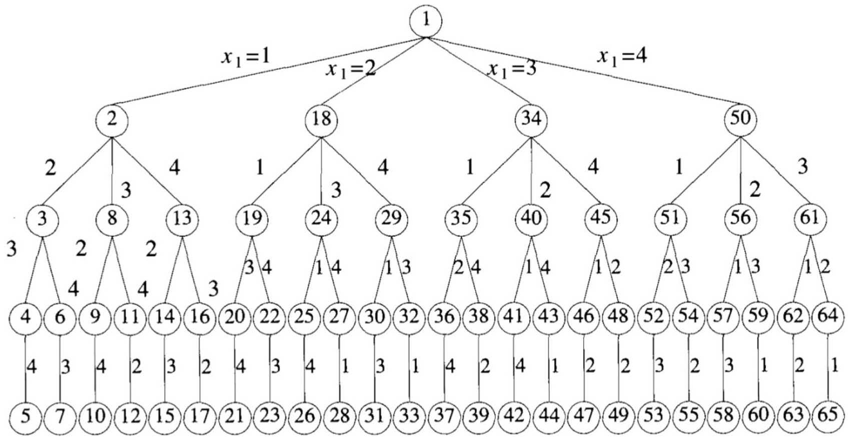
Branch&Bound C

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 assegnazinoe 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].

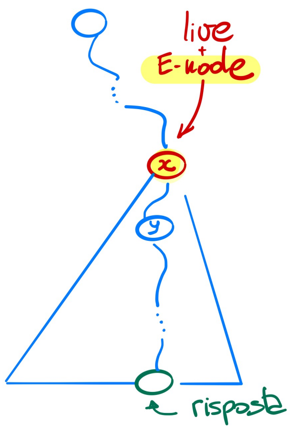
Punto 1 Illustrare a grandi linee come si sviluppa la visita dello spazio degli stati qui sopra illustrato, che possiamo definire in profondità ed in ampiezza.

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. □

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. □

Domanda C.5 La seguente immagine:

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

Algorithm LCSearch(t)
// Search t for an answer node.
{
    if *t is an answer node then output *t and return;
    E := t; // E-node.
    Initialize the list of live nodes to be empty;
    repeat
    {
        for each child x of E do
        {
            if x is an answer node then output the path
                from x to t and return;
            Add(x); // x is a new live node.
            (x -> parent) := E; // Pointer for path to root
        }
        if there are no more live nodes then
        {
            write ("No answer node"); return;
        }
        E := Least();
    } until (false);
}
```

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. □

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*. □

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.* □

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:

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

□

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:

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

□

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 $\frac{1}{2}$-*approxiamtion.* Argomentare sul perché Ext-Greedy non può appartenere a una classe di migliore di $\frac{1}{2}$-*approximation.* □

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

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

o anche più semplice, ed enunciarne la proprietà principale. □

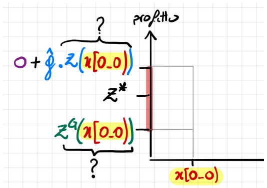
Domanda C.12 Dimostrare almeno un caso dell'enunciato: "Greedy-LKP gode della proprità *Greedy Choice Property*". □

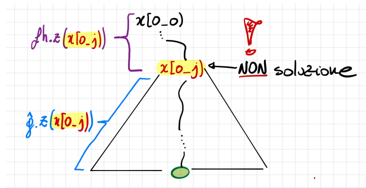
Domanda C.13 Illustrare significato ed origine della sequenza di inclusioni tra profitti seguenti:

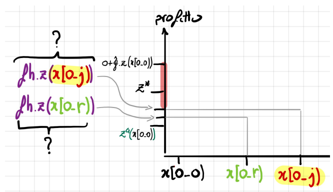
$$\hat{p} \leq z^{\mathsf{G}} \leq z^{*} \leq \lfloor z^{\mathsf{LP}} \rfloor \leq z^{\mathsf{LP}} \leq \hat{p} + p_{split} \leq z^{\mathsf{G}} + p_{split} \ .$$

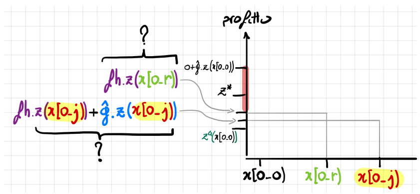
□

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


profitto
0 + g.z(x[0_0])
z*
z^q(x[0_0])
x[0_0]


fh.z(x[0_j])
x[0_0]
x[0_j]
NON soluzione
g.z(x[0_j])


profitto
fh.z(x[0_j])
fh.z(x[0_r])
z*
x[0_0]
x[0_r]
x[0_j]


profitto
fh.z(x[0_r])
fh.z(x[0_j]) + g.z(x[0_j])
z*
x[0_0]
x[0_r]
x[0_j]

□

Programmazione dinamica D

Domanda D.1 Illustrare una soluzione ricorsiva del problema KP, impostata in accordo con la tecnica "Programmazione Dinamica" con accenni alla sua complessità pseudo-polinomiale. □

Domanda D.2 Illustrare vari algoritmi iterativi che implementano soluzioni al problema KP, impostati in accordo con la tecnica "Programmazione Dinamica". □

Complessità computazionale E

Domanda E.1 Illustrare le classi PTime ed NPTime in termini classici. □

Domanda E.2 Illustrare le classi PTime ed NPTime in termini di possibili linguaggi imperativi paradigmatici. □

Domanda E.3 Argomentare sul perché la definizione della classe PTime può essere basata indifferentemente sulla definizione di Macchine di Turing o su un ragionevole linguaggio di programmazione paradigmatico. □

Domanda E.4 Argomentare sul perché proprietà sulla complessità computazionale di un problema di ottimizzazione sono trasferibili al corrispondente problema decisionale e viceversa. □

Domanda E.5 Ricordando il concetto "Riduzione", dimostrare che KP è intrattabile, assumendo che SAT lo sia. □

Domanda E.6 Illustrare un qualche aspetto fondamentale della dimostrazione che SAT $\leq_{\mathsf{P}}$ NNF. □

Domanda E.7 Nell'ambito della dimostrazione di NNF $\leq_{\mathsf{P}}$ CNF, illustrare intuizione sulla e definizione della trasformazione di Tseytin, dimostrando la sua proprietà principale. □

Domanda E.8 Dimostrare NNF $\leq_{\mathsf{P}}$ CNF, supponendo di avere a disposizione la trasformazione di Tseytin ed il suo lemma principale. □

Domanda E.9 Illustrare un qualche aspetto fondamentale della dimostrazione che CNF $\leq_{\mathsf{P}}$ 3CNF. □

Domanda E.10 Nell'ambito della dimostrazione 3CNF $\leq_{\mathsf{P}}$ 3COL, definire i tratti fondamentali della riduzione L da formule a grafi e l'intuizione che ne guida la definizione. □

Domanda E.11 Nell'ambito della dimostrazione $\mathsf{3COL} \leq_{\mathsf{P}} \mathsf{EXCO}$, definire i tratti fondamentali della riduzione E_x da grafi ad insiemi e l'intuizione che ne guida la definizione. $\square$

Domanda E.12 Nell'ambito della dimostrazione $\mathsf{EXCO} \leq_{\mathsf{P}} \mathsf{Subsetsum}$, definire i tratti fondamentali della riduzione E_s da insiemi a insiemi e l'intuizione che ne guida la definizione. $\square$

Problemi computazionali F

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

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

Bibliografia F

[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.