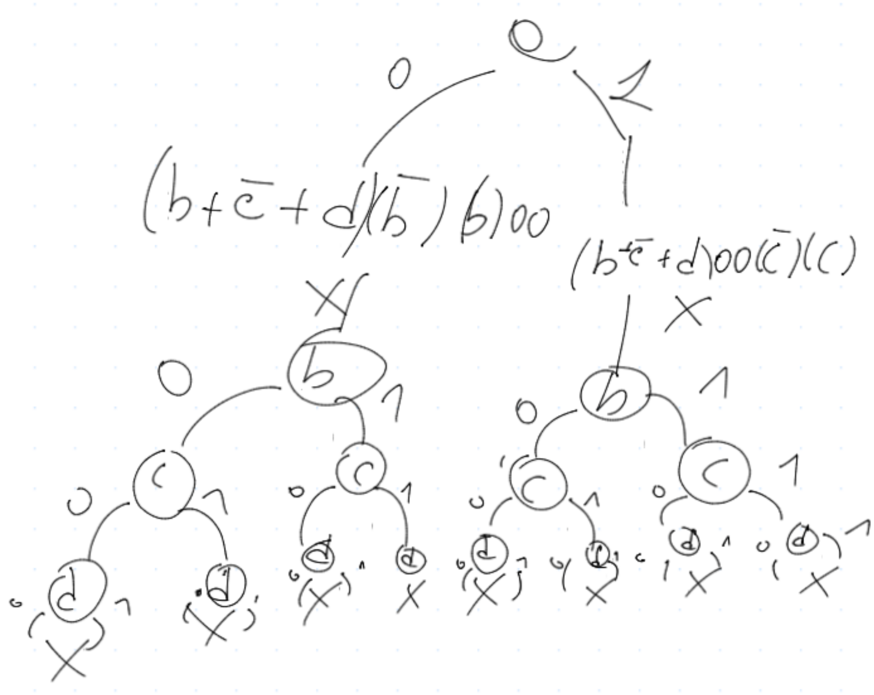
Esercizi2

Es. 1

Il vantaggio derivante da tale accorgimento è che viene dimezzato il tempo di esecuzione dell'algoritmo. D'altronde basta aumentare di 1 bit il numero da fattorizzare e il vantaggio sarebbe perso.

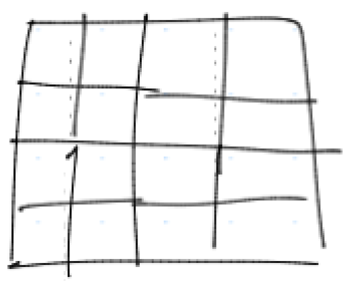
Es. 2

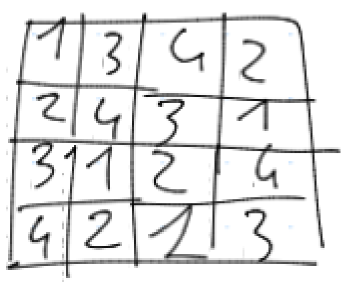
$$(\bar{a}+b+\bar{c}+d)(\bar{a}+\bar{b})(\bar{a}+b)(a+\bar{c})(a+c)$$



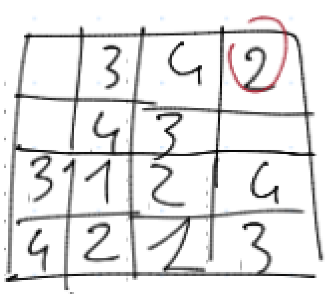
Es. 3

Partiamo dalla soluzione iniziale che otteniamo su un Sudoku $n^2 * n^2$ con $n = 2$ senza nessuna casella iniziale preimpostata. Ciò ci permette di riempire il Sudoku in modo efficiente e con complessità polinomiale.
Bisogna riempire la prima colonna con i numeri da 1 ad n, la seconda da $1 + n$ e applicando uno shift ciclico quando arriviamo ad n^2 e questo per le prime n colonne. Per il resto bisogna applicare uno shift ciclico diagonale.





Ora bisogna individuare nelle sottomatrici da 1 ad n le colonne con solo i numeri compresi tra 1 ed n. (In questo caso c_1 e c_4).
Cancelliamo i valori in quelle colonne e li sostituiamo con i valori contenuti nel problema LatinSquare iniziale.



Ora nelle caselle vuote possono esserci solamente i numeri compresi tra 1 ed n quindi risolvere questo Sudoku è equivalente a risolvere il problema per il LatinSquare.

Es. 4

Una possibile soluzione è quella di posizionare sulla diagonale principale tutti 1. A questo punto possiamo riempire le righe partendo da 1 fino ad arrivare ad n applicando shift ciclici con complessità polinomiale $O(n^2)$.

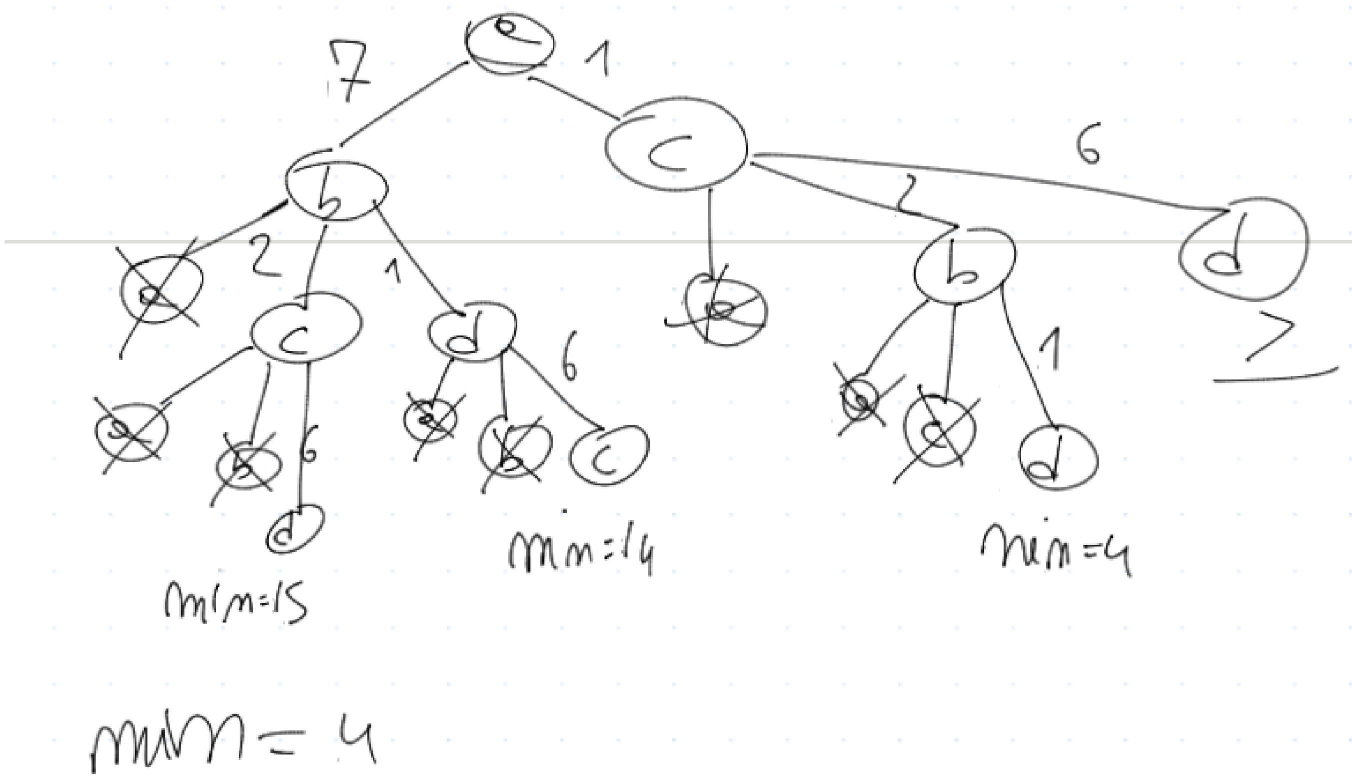
Es. $n = 3$

1		
	1	
		1

1	2	3
3	1	2
2	3	1

Es. 5

Partiamo dal nodo a. Se un nodo all'interno dell'albero di BackTracking è contrassegnato con una X allora l'inserimento di quel nodo all'interno di S (con S soluzione del TSP) creerebbe un ciclo, rendendo S una soluzione non accettabile.
Se un nodo invece ha una soluzione parziale $\geq$ del massimo trovato fino a quel momento non ne vale la pena continuare ad esplorare quel sotto-albero.



Es. 6

Il problema IndipendentSet è un problema di ottimizzazione (parlare dell' IndipendentSet se vuoi). Possiamo applicare l'idea del BackTracking in linea generale, ma siccome abbiamo un problema di ottimizzazione utilizzeremo la tecnica del Branch-And-Bound per valutare anche eventuali soluzioni parziali.

All'interno dell'albero i nodi saranno i nodi del grafo $G = (V, E)$ fornito in input.
Gli archi invece stanno ad indicare se inserire il nodo v all'interno dell'insieme S che indicherà una soluzione del problema IndipendentSet.

I nodi marchiati con X sta ad indicare che l'inserimento del nodo v all'interno di S porta ad una situazione in cui $\exists u \in S | (u, v) \in E$.
Smette di esplorare un sottoalbero se il numero di nodi ancora da esplorare $+$ il numero di nodi attualmente in S è $\leq$ del massimo fin ora trovato.

Es. 7

Sia $V = (v_1, \ldots, v_n)$ l'insieme delle variabili e $C = (c_1, \ldots, c_n)$ l'insieme delle clausole.
Costruiamo il grafo $G = (V', E)$ dove
$V' = (v_1, \ldots, v_n, c_1, \ldots, c_n)$
l'arco $(c_i, v_j) \in E \iff c_i$ contiene la variabile v_j.

Date le condizioni del problema, ogni clausola e ogni variabile avrà esattamente 3 archi.

Consideriamo $C' \subseteq C$ e $N(C')$ tutte le variabili contenute in almeno una clausola in C'.

Il numero di archi (c_i, v_j) t.c. $c_i \in C'$ è esattamente (per come è disposto il problema) uguale a $3|C'|$ (e per ogni clausola).
Al massimo questo valore è $3|N(C')|$ (3 per ogni variabile).

Allora $|C'| \leq |N(C)|$ poiché può capitare che alcune clausole condividono qualche variabile.

Per il teorema di $Hall$ è possibile stabilire per ogni clausola c_i una variabile $h(c_i)$ se ovviamente $(c_i, h(c_i)) \in E$.
È possibile quindi creare un perfect matching.

Se il letterale l di c_i corrispondente alla variabile $h(c_i)$ è in forma vera, imposta il valore della variabile a $True$
$False$ altrimenti.

https://cs.stackexchange.com/questions/105539/3sat-instance-with-exactly-3-instances-of-each-literal
https://en.wikipedia.org/wiki/Hall%27s_marriage_theorem#Constructive_proof_of_the_hard_direction

Es. 8

Sfruttare i fati del problema

$t_j \leq \frac{1}{2}\sum_{i=1}^{n} t_i$

Dobbiamo dimostrare

$\frac{1}{2}T_1 \leq T_2 \leq 2T_1$

Dato $T_1 \geq T_2$

Dimostriamo $T_2 \leq 2T_1$: banale

Dimostriamo $\frac{1}{2}T_1 \leq T_2$

$\frac{1}{2}T_1 \leq T_2 = T_1 \leq 2T_2$

Se l'algoritmo si è fermato significa che

$t_{min} > \Delta = T_1 - T_2$

$t_{min} + T_2 > T_1$

$T1 < t_{min} + T_2 < 2T_2$

$T1 < 2T_2$

$\frac{1}{2}T_1 \leq T_2$