Cognome: **Nome**: **# Matricola**: **Riga**: **Col**:

Algoritmi e Strutture Dati – 18/07/2019 – Parte A

A1 – Complessità – Punti ≥ 8

Trovare i limiti superiore e inferiori più stretti possibili per la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + 2T(\lfloor n/4 \rfloor) + 3T(\lfloor n/6 \rfloor) + n^2 & n \geq 6 \\ 1 & n < 6 \end{cases}$$

A2 – Somma di quattro – Punti ≥ 10

Dato un vettore A contenente n interi non ordinati e un valore k, scrivere un algoritmo che restituisca **true** se esistono quattro elementi del vettore (anche ripetuti) che sommati insieme danno origine al valore k.

boolean four(**int**[] A, **int** n, **int** k)

Assumete che gli interi siano compresi nell'intervallo $[-2^{60}, 2^{60} - 1]$ (memorizzabili in interi a 64 bit), che n sia compreso fra 1.000 e 10.000, di aver a disposizione 1 GB di RAM per questo compito.

Discutere informalmente la correttezza dell'algoritmo e calcolare la sua complessità computazionale. Notate che soluzioni corrette ma di complessità $\Theta(n^4)$ o superiore otterranno una valutazione pari al 30%.

Ad esempio, dati un vettore $A = [7, 1, 13, 12, -3, 7]$ e un intero $k = 35$, la risposta è **true** in quanto $13 + 13 + 12 - 3 = 35$.

A3 – Nurikabe – Punti ≥ 12

Nurikabe[1] è uno yōkai, uno spirito del folklore giapponese che costruisce muri invisibili per confondere i viaggiatori durante la notte. È anche il nome di un puzzle della Nikoli[2], la stessa casa editrice di Fillomino (detto anche Games of (Approximated) Thrones).
Si gioca su una griglia G di dimensione $n \times n$, dove le celle sono considerate adiacenti se si toccano orizzontalmente o verticalmente. Nella griglia sono piazzati alcuni numeri interi positivi. Compito del giocatore è di colorare le celle di bianco o di nero, rispettando le regole seguenti:

1. Le celle colorate di bianco connesse fra loro formano una *stanza*.
2. Le celle colorate di nero connesse fra loro formano un *muro*.
3. Esiste un solo muro.
4. Ogni cella contenente un numero è colorata di bianco.
5. Ogni stanza contiene esattamente un numero, che deve corrispondere al numero di celle che formano la stanza.
6. Il muro non può contenere aree nere di dimensione 2×2.

Risolvere questo problema è complesso, in quanto NP-completo. Il vostro compito, tuttavia, è molto più semplice. Scrivere un algoritmo:

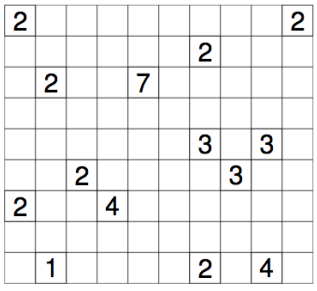
boolean isSolution(**int**[][] G, **int**[][] S, **int** n)

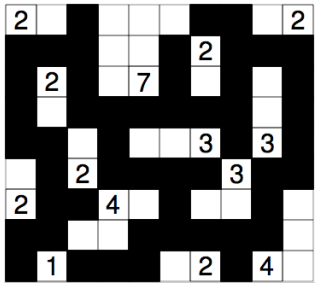
che prenda in input:

- una matrice di input G, dove ogni elemento può essere 0 (cella non numerata) oppure > 0 (cella numerata)
- una matrice soluzione S, dove ogni elemento può essere 0 (bianco) oppure 1 (nero)

e restituisca **true** se e solo se la matrice S è una soluzione corretta del problema Nurikabe G.

Le due figure seguenti mostrano un problema di Nurikabe e la sua soluzione. Credits: Wikipedia





[1] https://en.wikipedia.org/wiki/Nurikabe
[2] https://en.wikipedia.org/wiki/Nurikabe_(puzzle)

Cognome: **Nome**: **# Matricola**: **Riga**: **Col**:

Algoritmi e Strutture Dati – 18/07/2019 – Parte B

B1 – Rete elettrica trentina – Punti ≥ 8

La rete elettrica del Trentino è modellata da un grafo $G = (V, E)$. L'insieme V è costituito da tre sottoinsiemi disgiunti:

- un insieme P di *centrali di produzione*;
- un insieme T di *nodi di trasformazione*;
- un insieme D di *nodi di distribuzione*.

Ogni centrale $p \in P$ può produrre una potenza $power[p] \in \mathbb{R}^+$ in MegaWatt (MW). Ogni nodo di trasformazione $t \in T$ necessita di $consume[t] \in \mathbb{R}^+$ MW per rifornire le case a cui è collegata.

Ogni arco $(u, v) \in E$ può trasmettere fino a k MW (quantità fissa); si noti che $u, v \in V$, ovvero un arco può mettere in contatto qualsiasi tipologia di nodi.

Siete un addetto corrotto alla manutenzione dei nodi di distribuzione della rete. Potete installare un deviatore in ognuno dei nodi di distribuzione $d \in D$ in grado di "rubare" esattamente r MW da ogni nodo, per un totale di $r \cdot |D|$ MW. Ma attenzione: se rubando tale potenza qualche nodo di trasformazione $t \in T$ riceve una potenza inferiore a $consume[t]$ MW, questo causerà un blackout e verrete scoperti.

Descrivere un algoritmo che restituisca **true** se e solo se riuscite a rubare esattamente $r \cdot |D|$ MW senza essere scoperti.

Discutere informalmente la correttezza dell'algoritmo e calcolare la sua complessità computazionale.

B2 – k-alternato – Punti ≥ 10

Un vettore di interi distinti è *k-alternato*, con $k \geq 2$, se tutti i sottovettori crescenti/decrescenti contigui contenuti nel vettore hanno lunghezza k o inferiore.

Ad esempio, $[2, 4, 6, 8, 7, 5, 3, 10, 12]$ è 4-alternato, perché i sottovettori crescenti/decrescenti contigui $[2, 4, 6, 8]$, $[8, 7, 5, 3]$, $[3, 10, 12]$ contengono al più 4 elementi.

Sia A un insieme contenente n interi distinti. Scrivere un algoritmo

int kAlternate(SET A, **int** k)

che restituisca il numero di sequenze k-alternate che possono essere ottenute posizionando i valori di A in un vettore.

Discutere informalmente la correttezza dell'algoritmo e calcolare la sua complessità computazionale. Si noti che in questo caso si terranno in considerazione anche di eventuali miglioramenti nei fattori moltiplicativi.

Ad esempio, i vettori 2-alternati ottenibili da $C = \{2, 7, 5\}$ sono i seguenti:

$$[2, 7, 5]$$
$$[5, 7, 2]$$
$$[5, 2, 7]$$
$$[7, 2, 5]$$

B3 – Max Increasing – Punti ≥ 12

Sia A un vettore contenente n interi positivi.

- Una *sottosequenza crescente* di A è un sottosequenza di A i cui valori sono ordinati.
- Il *valore di una sottosequenza* è pari alla somma dei suoi valori.
- Una sottosequenza crescente di A è *massimale* se ha valore massimo fra tutte le sottosequenze crescenti di A.

Scrivere un algoritmo

int maxIncreasing(**int**[] A, **int** n)

che restituisca il *valore* di una sottosequenza crescente massimale contenuta in A.

Discutere informalmente la correttezza dell'algoritmo e calcolare la sua complessità computazionale.

Se $A = [\underline{\mathbf{1}}, \underline{8}, \mathbf{4}, \underline{12}, 2, 10, \mathbf{6}, \underline{14}, 1, \mathbf{9}, 5, \mathbf{13}, 3, 11]$, una delle sottosequenze crescenti è $[1, 4, 6, 9, 13]$ con valore 33; una sottosequenza crescente massimale è tuttavia $[1, 8, 12, 14]$, con valore 35. L'algoritmo deve quindi restituire 35. Si noti anche che $[1, 1]$ non è una sottosequenza crescente.