

# SECONDO PROGETTO ASD 2018/2019

# GAME OF (APPROXIMATED) THRONES



# GAME OF (APPROXIMATED) THRONES

Le casate di Westeros sono  
ormai in guerra da anni;  
numerosi Lord proclamano  
di essere l'unico vero Re.

# GAME OF (APPROXIMATED) THRONES

Le casate di Westeros sono  
ormai in guerra da anni;  
numerosi Lord proclamano  
di essere l'unico vero Re.

Fra di essi, un Lord  
sconosciuto riesce a farsi  
strada fra tutti e a  
conquistare il Trono di  
Spade!

# GAME OF (APPROXIMATED) THRONES

Le casate di Westeros sono ormai in guerra da anni; numerosi Lord proclamano di essere l'unico vero Re.

Fra di essi, un Lord sconosciuto riesce a farsi strada fra tutti e a conquistare il Trono di Spade!

Il suo nome è Albert di casa Montron, Primo del suo Nome, Re dei sandali, Padre dei problemi, Protettore degli algoritmi, Distruttore di medie.



# Dopo la battaglia

Dopo la vittoriosa battaglia per il trono, il nuovo Re deve dividere il territorio fra le varie casate, premiando chi si è alleato con lui e punendo chi si è defilato.



# Dopo la battaglia

Dopo la vittoriosa battaglia per il trono, il nuovo Re deve dividere il territorio fra le varie casate, premiando chi si è alleato con lui e punendo chi si è defilato. Purtroppo molte delle antiche mappe sono andate bruciate - questo succede quando si prende un drago come animale da compagnia invece di un gatto.



# LA SUDDIVISIONE DEI TERRITORI (I)

Grazie a grandi ricerche, il maestro Criswell Tarly è riuscito a ricostruire una mappa del territorio e ad individuare alcune zone strategiche, contraddistinte dalla presenza dei **castelli**.



# LA SUDDIVISIONE DEI TERRITORI (I)

Grazie a grandi ricerche, il maestro Criswell Tarly è riuscito a ricostruire una mappa del territorio e ad individuare alcune zone strategiche, contraddistinte dalla presenza dei **castelli**.

Dopo importanti discussioni con il primo cavaliere Martya Stark, si è decisa la dimensione delle aree da assegnare per ogni castello. Pur di riuscire ad assegnare più castelli possibili, rispettando le dimensioni scelte, potrà accadere che più castelli vengano assegnati ad una stessa suddivisione. Inoltre, il Re vuole evitare che due suddivisioni della stessa dimensione si trovino vicine. Tra loro non ci sarebbe alcuna rivalità e potrebbero allearsi contro di lui.



## LA SUDDIVISIONE DEI TERRITORI (II)

Ora non resta che suddividere il territorio fra le casate, rispettando le seguenti regole:

- Ogni suddivisione deve contenere uno o più castelli.
  - Ogni suddivisione deve essere **esattamente** della dimensione richiesta, data dai suoi castelli.
  - Due suddivisioni della stessa dimensione non possono toccarsi.
  - Alcune parti del territorio possono rimanere non assegnate.
- ⇒ Se non si riesce a assegnare una suddivisione a un castello viene raso al suolo.
- ⇒ È importante riuscire a fare in modo che **tanti più castelli possibili** facciano parte di una suddivisione.

Aiutate Re Albert, Criswell Tarly e Martya Stark a creare una giusta suddivisione dei territori!

## IL VOSTRO COMPITO

Il vostro compito è fornire una mappa con delle suddivisioni valide associate ai castelli dati in input, tale che **quanti più castelli possibili** facciano parte di una suddivisione valida.

# ESEMPIO: SOLUZIONE PARZIALE



⇒  $N_v : 9, N_c : 12, N_1 : 1$

⇒ punti: 3 . 64 / 5



# ESEMPIO: SOLUZIONE OTTIMA



⇒  $N_v : 12$ ,  $N_c : 12$ ,  $N_1 : 1$

⇒ punti: 5.00/5

## SUDDIVISIONI VALIDE (I)

Una suddivisione valida è definita nel seguente modo:

- Due celle confinano se hanno un lato in comune, ovvero le celle confinano orizzontalmente o verticalmente. Celle con un vertice in comune (in diagonale) non sono confinanti.
- Se due celle confinanti contengono lo stesso valore  $k > 0$ , allora appartengono alla stessa suddivisione.
- La dimensione di una suddivisione è data dal numero di celle che la compongono.
- Una suddivisione di dimensione  $k$  è **valida** se e solo se le sue celle contengono il valore  $k$ .

## SUDDIVISIONI CONFINANTI

È impossibile che due suddivisioni valide contenenti valori  $k$  siano confinanti; infatti, la loro unione sarebbe vista come una suddivisione **non valida**, in quanto contenente il valore  $k$  e di dimensione  $2k$ .

## SUDDIVISIONI VALIDE (II)

Le seguenti regole definiscono il concetto di associazione fra castelli e suddivisioni. L'output rispetta la suddivisione dei castelli se:

- Contiene solo suddivisioni valide.
- Per ogni suddivisione valida di dimensione  $k$ :
  - ▶ almeno una delle sue celle contiene un castello di valore  $k$  nella corrispondente cella dell'input;
  - ▶ nessuna delle sue celle contiene un castello di valore  $k' \neq k$  nella corrispondente cella dell'input, non potete "sostituire" un castello con un altro;
- tutte le celle dell'output che non fanno parte di una suddivisione devono contenere il valore 0, anche se nella corrispondente cella dell'input c'era un castello.

### CASTELLI SENZA TERRENO

Se non si riesce a trovare una suddivisione valida per un castello viene raso al suolo, mettendo uno 0 nella cella corrispondente.

# INPUT/OUTPUT

**Input:** La mappa del territorio, con i rispettivi castelli e la dimensione della suddivisione di cui potrebbero far parte. Tutti i numeri sono separati sono separati da **tabulazioni**.

- La prima riga riporta 2 numeri,  $N$  ed  $M$ , rispettivamente il numero di righe e di colonne della mappa.
- Le successive  $N$  righe sono composte da  $M$  interi che descrivono la mappa:
  - ▶ Se una cella della mappa è marcata con un numero  $k > 0$ , questa rappresenta un **castello**, che deve essere posizionato in una **suddivisione di dimensione k**. Tutte le altre celle sono marcate con uno 0.

**Output:** le vostre proposte di suddivisione della mappa, ogni soluzione è così composta:

- $N$  righe, ciascuna contenente  $M$  colonne, ovvero la mappa delle suddivisioni;
- 1 riga contenente i caratteri  $\ast \ast \ast$ .

## ESEMPI: SOLUZIONE PARZIALE

INPUT	OUTPUT
9 5	7 7 0 3 3
0 0 0 0 0	7 2 2 0 3
0 2 0 0 3	7 7 0 0 9
0 0 0 0 0	0 7 7 9 9
0 0 7 0 0	2 0 1 9 0
0 6 1 0 0	2 9 9 9 9
2 0 0 0 9	6 6 6 4 9
0 0 6 0 0	0 6 0 4 4
5 0 0 0 0	0 6 6 4 0
5 0 6 4 0	* * *

⇒  $N_v : 9$ ,  $N_c : 12$ ,  $N_1 : 1$

⇒ punti: 3.64/5

## ESEMPI: SOLUZIONE OTTIMA

INPUT					OUTPUT				
9	5				7	7	7	7	3
0	0	0	0	0	6	2	7	3	3
0	2	0	0	3	6	2	7	9	9
0	0	0	0	0	6	6	7	9	9
0	0	7	0	0	6	6	1	9	9
0	6	1	0	0	2	2	6	6	9
2	0	0	0	9	5	6	6	9	9
0	0	6	0	0	5	5	6	4	4
5	0	0	0	0	5	5	6	4	4
5	0	6	4	0	***				

⇒  $N_v : 12, N_c : 12, N_1 : 1$

⇒ punti: 5.00 / 5

## ESEMPI: SOLUZIONE NON VALIDA (I)

INPUT	OUTPUT
9 5	7 7 0 3 3
0 0 0 0 0	7 2 2 0 3
0 2 0 0 3	7 7 0 0 9
0 0 0 0 0	0 7 7 9 9
0 0 7 0 0	2 0 1 9 0
0 6 1 0 0	2 9 9 9 9
2 0 6 0 9	6 6 6 0 9
0 0 6 0 0	0 6 0 4 4
5 0 6 0 0	0 6 6 4 0
5 0 6 4 0	* * *

⇒ **Errore!** Dimensione della suddivisione (3) diversa da quella dichiarata (4)

## ESEMPI: SOLUZIONE NON VALIDA (II)

INPUT	OUTPUT
9 5	0 0 0 0 0
0 0 0 0 0	0 0 0 0 0
0 2 0 0 3	0 0 0 0 0
0 0 0 0 0	0 0 0 0 0
0 0 7 0 0	0 0 0 9 0
0 6 1 0 0	0 0 0 9 9
2 0 6 0 9	0 0 0 9 9
0 0 6 0 0	0 0 0 9 9
5 0 6 0 0	0 0 0 9 9
5 0 6 4 0	* * *

⇒ **Errore!** Output non corrispondente all'input, il castello nella cella (9, 4) di valore 4 è stato sostituito con 9.

# NOTE SU INPUT

## ASSUNZIONI GENERALI

- $0 < N, M \leq 200$
- $0 < k < NM$

## CASI DI TEST

- Ci sono 20 casi di test in totale.
- I casi di test hanno tutti almeno una soluzione ottima.
- In almeno 11 casi c'è un solo castello per suddivisione.

## DATASET DI ESEMPIO

**Attenzione!** Per gli input forniti nel dataset di esempio non è stata calcolata una soluzione ottima. Per questo motivo il dataset non contiene anche i relativi output, solitamente messi a disposizione.

# OUTPUT

Se scrivete una soluzione esponenziale:

- Importate `got.h` (scaricabile da judge).
- Man mano che migliorate la soluzione, scrivetela in output terminando la riga con `***`.
- La libreria arresterà il programma prima del timeout.

```
... include delle librerie di sistema ...
#include "got.h"
int main() {
    ...
}
```

Note:

- Il `main` va sempre dichiarato come `int main()` o `int main(void)`.
- Il correttore considererà l'ultima riga di output che finisce con `***` quindi, anche se non stampate soluzioni multiple, terminate l'output con `***`.

# COMPILARE IN LOCALE

Per testare le vostre soluzioni in locale (supponiamo che il vostro file si chiami `got.cpp`):

- Scaricate `grader.cpp`
- Il comando di compilazione è il seguente

```
/usr/bin/g++ -DEVAL -std=c++11 -O2 -pipe -static -s -o  
got grader.cpp got.cpp
```

I file `got.cpp`, `grader.cpp` e `got.h` devono essere nella stessa cartella.

Per sistemi Mac OS X e Windows vedere la nota nel testo.

**Nota:** Per questo esercizio è necessario usare il C++, non è possibile usare il C.

# PUNTEGGIO

- Ci sono 20 casi di test: ogni test assegna un punteggio (max 5 punti), che considera fino ai centesimi;
- Una soluzione proposta è valida se rispetta tutti le regole richieste. Soluzioni non valide fanno **zero** punti!
- Le risposte valide invece ottengono questo punteggio:

$$\max \left( 0.0, \frac{N_v - N_1}{N_c - N_1} \right) \cdot 5$$

Dove  $N_v$  è il numero di castelli inseriti in suddivisioni valide,  $N_c$  è il numero di castelli dati in input,  $N_1$  è il numero di castelli che richiedono dimensione 1.

- ⇒ La **sufficienza è posta a 40 punti**.
- ⇒ C'è un limite di 40 sottoposizioni per gruppo.

# PUNTI BONUS PER L'ESAME

L'assegnazione punti avviene in maniera competitiva, come per il primo progetto:

- **3 punti** ai gruppi nel primo terzile della classifica (primo terzo della classifica, punteggio maggiore a quello fatto da almeno 2/3 dei gruppi);
- **2 punti** ai gruppi nel secondo terzile della classifica (secondo terzo della classifica);
- **1 punto** ai gruppi nel terzo terzile della classifica (ultimo terzo della classifica);

# CONSEGNA

**Consegna: 30 maggio ore 20:00**

Per caricare il vostro codice, recatevi su

<https://judge.science.unitn.it/arena>

## SUGGERIMENTI

Cominciate subito a lavorare al progetto per presentarvi al prossimo laboratorio (mercoledì 29 maggio) con tutte le domande che vorrete fare.

Come al solito sappiate che:

- potete venire a ricevimento,
- risponderemo alle vostre mail

...ma non durante il week-end!

## È PERMESSO:

- ① Discutere all'interno del gruppo
- ② Chiedere chiarimenti sul testo
- ③ Chiedere opinioni su soluzioni
- ④ Sfruttare codice fornito nei laboratori
- ⑤ Utilizzare pseudocodice da libri o Wikipedia
- ⑥ Richiedere aiuto (anche pesante) per la soluzione "minima"
- ⑦ Venire a ricevimento

# DONT'S

## È VIETATO:

- ① Discutere con altri gruppi
- ② Mettere il proprio codice su repository pubblici
- ③ Utilizzare codice scritto da altri
- ④ Condividere codice (abbiamo potenti mezzi!)

## DATE E ORARI

- lunedì 27 maggio 2019 dalle 15:30 alle 17:30 in aula Cornisello (Povo 1, piano 1);
  - martedì 28 maggio 2019 dalle 14:30 alle 16:30 in aula Cornisello (Povo 1, piano 1);
  - mercoledì 29 maggio 2019 dalle 15:30 alle 17:30 in A101 (ricevimento in lab);
  - giovedì 30 maggio 2019 dalle 14:30 alle 16:30 in aula Cornisello (Povo 1, piano 1);
- ⇒ Prima di venire a ricevimento, è obbligatorio richiedere appuntamento via mail ([cristian.consonni@unitn.it](mailto:cristian.consonni@unitn.it)) e ([marta.fornasier@studenti.unitn.it](mailto:marta.fornasier@studenti.unitn.it)).