

Esame 20210907

Esercizio 3

(1) Esercizio 3 v1

ESSAY

marked out of 10

penalty 0

File picker

Un labirinto si può definire come una matrice `m` di dimensioni `NxL`. Ogni cella della matrice contiene il valore intero 1 (rappresentante uno spazio libero, quindi percorribile) oppure il valore intero 0 (rappresentante uno spazio chiuso, quindi un muro). Un labirinto ha una cella di partenza (per esempio, la cella `m[0][0]`) e una cella di arrivo (indicata generalmente con `m[x][y]`).

Scrivere nel file `esercizio3.cc` la corretta implementazione della procedura `risolviLabirinto` che prende come parametri formali una matrice di valori interi (0 oppure 1) `labirinto` di dimensioni `5x5`, un intero `x` e un intero `y`. Usando una pila come supporto, la procedura `risolviLabirinto` deve trovare e stampare a video una sequenza contigua di celle della matrice `labirinto` contenenti il valore 1 (un percorso, quindi) che colleghi la cella di partenza `labirinto[0][0]` e la cella di arrivo `labirinto[x][y]`. Una sequenza è "contigua" quando celle consecutive differiscono di un solo indice; vale a dire che nel labirinto ci si può muovere verticalmente e orizzontalmente ma non diagonalmente. Per esempio, la sequenza `labirinto[0][0]`, `labirinto[0][1]`, `labirinto[1][1]` è contigua, mentre la sequenza `labirinto[0][0]`, `labirinto[1][1]` non è contigua.

Questi sono due esempi di esecuzione (la cella di partenza è in alto a sinistra):

Esempio 1, `arrivo = [3, 3]`

```
1  1  1  0  0
0  1  0  0  0
0  1  1  1  0
0  0  0  1  0
1  0  0  0  0
```

```
computer > ./a.out
```

```
Percorso: [0,0], [0,1], [1,1],
[2,1], [2,2], [2,3], [3,3]
```

Esempio 2, `arrivo = [4, 2]`

```
1  0  0  0  0
1  0  0  1  0
1  1  1  0  1
0  0  0  0  1
1  1  1  0  1
```

```
computer > ./a.out
```

```
Percorso:
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta definizione e implementazione della procedura `risolviLabirinto` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;

- Scaricare anche i file `pila.cc` e `pila.h` i quali implementano le funzionalità di una pila. Il file `pila.h` contiene la definizione della struct `cella` utilizzata come oggetto della pila. Usare questi file nella risoluzione dell'esercizio. Ricordarsi di inizializzare e deallocare la pila;
- La cella di partenza contiene sempre un "1". Il labirinto può non essere risolvibile, i.e., è possibile che la sequenza di celle contigue fra partenza e arrivo non esista (in tal caso, la procedura `risolviLabirinto` non deve stampare a video nulla). Assumete che gli interi `x` e `y` siano entro i limiti della matrice;
- E' consentito definire e implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- All'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`.
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione.

Suggerimenti:

- E' consigliabile creare e utilizzare una variabile (per esempio, un'altra matrice) per tener traccia delle celle già visitate.
- Il percorso, se presente, può essere stampato in entrambe le direzioni (cioè sia dalla cella di partenza a quella di arrivo che viceversa).

Information for graders:

(2) Esercizio 3 v2

ESSAY

marked out of 10

penalty 0

File picker

Un labirinto si può definire come una matrice `m` di dimensioni `NxL`. Ogni cella della matrice contiene il valore intero 1 (rappresentante uno spazio libero, quindi percorribile) oppure il valore intero 0 (rappresentante uno spazio chiuso, quindi un muro). Un labirinto ha una cella di partenza (per esempio, la cella `m[0][0]`) e una cella di arrivo (indicata generalmente con `m[x][y]`).

Scrivere nel file `esercizio3.cc` la corretta implementazione della procedura `risolviLabirinto` che prende come parametri formali una matrice di valori interi (0 oppure 1) `labirinto` di dimensioni `6x6`, un intero `x` e un intero `y`. Usando una pila come supporto, la procedura `risolviLabirinto` deve trovare e stampare a video una sequenza contigua di celle della matrice `labirinto` contenenti il valore 1 (un percorso, quindi) che colleghi la cella di partenza `labirinto[0][0]` e la cella di arrivo `labirinto[x][y]`. Una sequenza è "contigua" quando celle consecutive differiscono di un solo indice; vale a dire che nel labirinto ci si può muovere verticalmente e orizzontalmente ma non diagonalmente. Per esempio, la sequenza `labirinto[0][0]`, `labirinto[0][1]`, `labirinto[1][1]` è contigua, mentre la sequenza `labirinto[0][0]`, `labirinto[1][1]` non è contigua.

Questi sono due esempi di esecuzione (la cella di partenza è in alto a sinistra):

Esempio 1, `arrivo = [3, 3]`

```
1  1  1  0  0
0  1  0  0  0
0  1  1  1  0
0  0  0  1  0
1  0  0  0  0
```

```
computer > ./a.out
```

```
Percorso: [0,0], [0,1], [1,1],
[2,1], [2,2], [2,3], [3,3]
```

Esempio 2, `arrivo = [4, 2]`

```
1  0  0  0  0
1  0  0  1  0
1  1  1  0  1
0  0  0  0  1
1  1  1  0  1
```

```
computer > ./a.out
```

```
Percorso:
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta definizione e implementazione della procedura `risolviLabirinto` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Scaricare anche i file `pila.cc` e `pila.h` i quali implementano le funzionalità di una pila. Il file `pila.h` contiene la definizione della struct `cella` utilizzata come oggetto della pila. Usare questi file nella risoluzione dell'esercizio. Ricordarsi di inizializzare e deallocare la pila;

- La cella di partenza contiene sempre un "1". Il labirinto può non essere risolvibile, i.e., è possibile che la sequenza di celle contigue fra partenza e arrivo non esista (in tal caso, la procedura `risolviLabirinto` non deve stampare a video nulla). Assumete che gli interi `x` e `y` siano entro i limiti della matrice;
- E' consentito definire e implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- All'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`.
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione.

Suggerimenti:

- E' consigliabile creare e utilizzare una variabile (per esempio, un'altra matrice) per tener traccia delle celle già visitate.
- Il percorso, se presente, può essere stampato in entrambe le direzioni (cioè sia dalla cella di partenza a quella di arrivo che viceversa).

Information for graders:

(3) Esercizio 3 v3

ESSAY

marked out of 10

penalty 0

File picker

Un labirinto si può definire come una matrice m di dimensioni $N \times L$. Ogni cella della matrice contiene il valore intero 1 (rappresentante uno spazio libero, quindi percorribile) oppure il valore intero 0 (rappresentante uno spazio chiuso, quindi un muro). Un labirinto ha una cella di partenza (per esempio, la cella $m[0][0]$) e una cella di arrivo (indicata generalmente con $m[j][k]$).

Scrivere nel file `esercizio3.cc` la corretta implementazione della procedura `risolviLabirinto` che prende come parametri formali una matrice di valori interi (0 oppure 1) `labirinto` di dimensioni 5×5 , un intero j e un intero k . Usando una pila come supporto, la procedura `risolviLabirinto` deve trovare e stampare a video una sequenza contigua di celle della matrice `labirinto` contenenti il valore 1 (un percorso, quindi) che colleghi la cella di partenza `labirinto[0][0]` e la cella di arrivo `labirinto[j][k]`. Una sequenza è "contigua" quando celle consecutive differiscono di un solo indice; vale a dire che nel labirinto ci si può muovere verticalmente e orizzontalmente ma non diagonalmente. Per esempio, la sequenza `labirinto[0][0]`, `labirinto[0][1]`, `labirinto[1][1]` è contigua, mentre la sequenza `labirinto[0][0]`, `labirinto[1][1]` non è contigua.

Questi sono due esempi di esecuzione (la cella di partenza è in alto a sinistra):

Esempio 1, $arrivo = [3, 3]$

```
1  1  1  0  0
0  1  0  0  0
0  1  1  1  0
0  0  0  1  0
1  0  0  0  0
```

```
computer > ./a.out
```

```
Percorso: [0,0], [0,1], [1,1],
[2,1], [2,2], [2,3], [3,3]
```

Esempio 2, $arrivo = [4, 2]$

```
1  0  0  0  0
1  0  0  1  0
1  1  1  0  1
0  0  0  0  1
1  1  1  0  1
```

```
computer > ./a.out
```

```
Percorso:
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta definizione e implementazione della procedura `risolviLabirinto` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Scaricare anche i file `pila.cc` e `pila.h` i quali implementano le funzionalità di una pila. Il file `pila.h` contiene la definizione della struct `cella` utilizzata come oggetto della pila. Usare questi file nella risoluzione dell'esercizio. Ricordarsi di inizializzare e deallocare la pila;

- La cella di partenza contiene sempre un "1". Il labirinto può non essere risolvibile, i.e., è possibile che la sequenza di celle contigue fra partenza e arrivo non esista (in tal caso, la procedura `risolviLabirinto` non deve stampare a video nulla). Assumete che gli interi `j` e `k` siano entro i limiti della matrice;
- E' consentito definire e implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- All'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`.
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione.

Suggerimenti:

- E' consigliabile creare e utilizzare una variabile (per esempio, un'altra matrice) per tener traccia delle celle già visitate.
- Il percorso, se presente, può essere stampato in entrambe le direzioni (cioè sia dalla cella di partenza a quella di arrivo che viceversa).

Information for graders:

(4) Esercizio 3 v4

ESSAY

marked out of 10

penalty 0

File picker

Un labirinto si può definire come una matrice m di dimensioni $N \times L$. Ogni cella della matrice contiene il valore intero 1 (rappresentante uno spazio libero, quindi percorribile) oppure il valore intero 0 (rappresentante uno spazio chiuso, quindi un muro). Un labirinto ha una cella di partenza (per esempio, la cella $m[0][0]$) e una cella di arrivo (indicata generalmente con $m[j][k]$).

Scrivere nel file `esercizio3.cc` la corretta implementazione della procedura `risolviLabirinto` che prende come parametri formali una matrice di valori interi (0 oppure 1) `labirinto` di dimensioni 6×6 , un intero j e un intero k . Usando una pila come supporto, la procedura `risolviLabirinto` deve trovare e stampare a video una sequenza contigua di celle della matrice `labirinto` contenenti il valore 1 (un percorso, quindi) che colleghi la cella di partenza `labirinto[0][0]` e la cella di arrivo `labirinto[j][k]`. Una sequenza è "contigua" quando celle consecutive differiscono di un solo indice; vale a dire che nel labirinto ci si può muovere verticalmente e orizzontalmente ma non diagonalmente. Per esempio, la sequenza `labirinto[0][0]`, `labirinto[0][1]`, `labirinto[1][1]` è contigua, mentre la sequenza `labirinto[0][0]`, `labirinto[1][1]` non è contigua.

Questi sono due esempi di esecuzione (la cella di partenza è in alto a sinistra):

Esempio 1, $arrivo = [3, 3]$

```
1  1  1  0  0
0  1  0  0  0
0  1  1  1  0
0  0  0  1  0
1  0  0  0  0
```

```
computer > ./a.out
```

```
Percorso: [0,0], [0,1], [1,1],
[2,1], [2,2], [2,3], [3,3]
```

Esempio 2, $arrivo = [4, 2]$

```
1  0  0  0  0
1  0  0  1  0
1  1  1  0  1
0  0  0  0  1
1  1  1  0  1
```

```
computer > ./a.out
```

```
Percorso:
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta definizione e implementazione della procedura `risolviLabirinto` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Scaricare anche i file `pila.cc` e `pila.h` i quali implementano le funzionalità di una pila. Il file `pila.h` contiene la definizione della struct `cella` utilizzata come oggetto della pila. Usare questi file nella risoluzione dell'esercizio. Ricordarsi di inizializzare e deallocare la pila;

- La cella di partenza contiene sempre un "1". Il labirinto può non essere risolvibile, i.e., è possibile che la sequenza di celle contigue fra partenza e arrivo non esista (in tal caso, la procedura `risolviLabirinto` non deve stampare a video nulla). Assumete che gli interi `j` e `k` siano entro i limiti della matrice;
- E' consentito definire e implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- All'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`.
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione.

Suggerimenti:

- E' consigliabile creare e utilizzare una variabile (per esempio, un'altra matrice) per tener traccia delle celle già visitate.
- Il percorso, se presente, può essere stampato in entrambe le direzioni (cioè sia dalla cella di partenza a quella di arrivo che viceversa).

Information for graders:

Total of marks: 40