Esame 20230728

Esercizio 3

(1) **Esercizio 3 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

Un *grafo* è un insieme di elementi detti *nodi* o *vertici* che possono essere collegati fra loro da *archi*. Più formalmente, si dice grafo una coppia ordinata $G = (V, E)$ di insiemi, con V insieme dei nodi ed $E \subseteq V \times V$ insieme degli archi, tali che gli elementi di E siano coppie di elementi di V. I grafi sono molto usati in diversi contesti (e.g., informatica, reti di telecomunicazioni, ...).

Assumendo che ci siano N nodi in un grafo, questo può essere rappresentato con una matrice di booleani, detta *matrice delle adiacenze* $A[i][j]$ con $i, j < N$ tale che $A[i][j]$ ha valore True se esiste un arco tra i nodi i e j, False altrimenti.

Scrivere nel file `esercizio3.cpp` una funzione `visita` che prende come argomento una matrice di booleani A, la dimensione N (un intero positivo) della matrice delle adiacenze, e un nodo $i < N$ che rappresenta un nodo del grafo e costruisce la lista concatenata `list *` dei nodi che si possono raggiungere dal nodo i seguendo gli archi uscenti dal nodo e dai nodi raggiunti, sfruttando un algoritmo il cui pseudocodice è il seguente.

- Creo un array `visited` di booleani di dimensione `N` che identifica se il nodo i-esimo è già stato visitato, ed inizialmente tutti i nodi non sono stati visitati (ogni elemento è False).
- Creo una lista `result` inizialmente vuota.
- Creo uno stack e vi inserisco il nodo i ricevuto come argomento.
- Fintanto che lo stack non è vuoto , prendo un elemento s dallo stack , se s non è stato ancora visitato, modifico `visited` per tenere traccia che il nodo è stato visitato (metto a true la posizione s), lo aggiungo in testa alla lista `result`, e poi considero tutti i nodi j adiacenti al nodo s (sono tutti quelli per cui $A[s][j]$ è true) che non sono stati ancora visitati e li aggiungo allo stack .
- Quando ho svuotato lo stack , ritorno la lista `result`.

La funzione `visita` è inserita in un `main` che genera un grafo random come matrice, lo stampa, chiama la funzione `visita`, stampa la lista ottenuta, la dealloca, e dealloca il grafo.

Di seguito è riportato un esempio di esecuzione.

```
marco > ./a.out
Grafo G[16][16]
0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0
1 1 0 1 1 0 1 1 1 0 1 0 0 0 0 0
1 1 0 1 1 1 0 0 0 1 1 0 0 1 1 1
0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1
1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 1
0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1
1 0 0 0 1 1 1 0 1 1 0 0 1 1 0 1
0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1
1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0
0 0 1 0 1 1 0 1 1 0 0 1 1 1 0 0
1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0
1 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0
```

```
1 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0
1 0 1 0 1 0 1 1 0 1 1 0 1 0 0 1
0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0
1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0
Lista: 6 0 2 9 3 7 4 5 8 11 14 12 15 13 10 1
```

Note:

- Scaricare i file `esercizio3.cpp`, `stack.cpp`, e `stack.h`, modificare `esercizio3.cpp` per inserire la dichiarazione e la definizione della funzione `visita`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- 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`, `cstdlib`, e `ctime`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma.

esercizio3.cpp

stack.cpp

stack.h

Information for graders:

(2) **Esercizio 3 v2**

ESSAY | marked out of 10 | penalty 0 | File picker

Un *grafo* è un insieme di elementi detti *nodi* o *vertici* che possono essere collegati fra loro da *archi*. Più formalmente, si dice grafo una coppia ordinata $G = (V, E)$ di insiemi, con V insieme dei nodi ed $E \subseteq V \times V$ insieme degli archi, tali che gli elementi di E siano coppie di elementi di V. I grafi sono molto usati in diversi contesti (e.g., informatica, reti di telecomunicazioni, ...).

Assumendo che ci siano N nodi in un grafo, questo può essere rappresentato con una matrice di booleani, detta *matrice delle adiacenze* $A[i][j]$ con $i, j < N$ tale che $A[i][j]$ ha valore True se esiste un arco tra i nodi i e j, False altrimenti.

Scrivere nel file `esercizio3.cpp` una funzione `visita` che prende come argomento una matrice di booleani A, la dimensione N (un intero positivo) della matrice delle adiacenze, e un nodo $i < N$ che rappresenta un nodo del grafo e costruisce la lista concatenata `list *` dei nodi che si possono raggiungere dal nodo i seguendo gli archi uscenti dal nodo e dai nodi raggiunti, sfruttando un algoritmo il cui pseudocodice è il seguente.

- Creo un array `visited` di booleani di dimensione `N` che identifica se il nodo i-esimo è già stato visitato, ed inizialmente tutti i nodi non sono stati visitati (ogni elemento è False).
- Creo una lista `result` inizialmente vuota.
- Creo uno coda e vi inserisco il nodo i ricevuto come argomento.
- Fintanto che la coda non è vuota , prendo un elemento s dalla coda , se s non è stato ancora visitato, modifico `visited` per tenere traccia che il nodo è stato visitato (metto a true la posizione s), lo aggiungo in testa alla lista `result`, e poi considero tutti i nodi j adiacenti al nodo s (sono tutti quelli per cui $A[s][j]$ è true) che non sono stati ancora visitati e li aggiungo alla coda .
- Quando ho svuotato la coda , ritorno la lista `result`.

La funzione `visita` è inserita in un `main` che genera un grafo random come matrice, lo stampa, chiama la funzione `visita`, stampa la lista ottenuta, la dealloca, e dealloca il grafo.

Di seguito è riportato un esempio di esecuzione.

```
marco > ./a.out
Grafo G[16][16]
0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0
1 1 0 1 1 0 1 1 1 0 1 0 0 0 0 0
1 1 0 1 1 1 0 0 0 1 1 0 0 1 1 1
0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1
1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 1
0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1
1 0 0 0 1 1 1 0 1 1 0 0 1 1 0 1
0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1
1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0
0 0 1 0 1 1 0 1 1 0 0 1 1 1 0 0
1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0
1 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0
1 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0
1 0 1 0 1 0 1 1 0 1 1 0 1 0 0 1
0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0
```

```
1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0
Lista: 12 15 14 5 13 11 9 2 10 8 7 6 4 3 0 1
```

Note:

- Scaricare i file `esercizio3.cpp`, `queue.cpp`, e `queue.h`, modificare `esercizio3.cpp` per inserire la dichiarazione e la definizione della funzione `visita`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- 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`, `cstdlib`, e `ctime`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma.

esercizio3.cpp

queue.cpp

queue.h

Information for graders:

Total of marks: 20