Algoritmi e Strutture Dati

Commesso Viaggiatore (`tsp`)

Testo del problema

Dato un grafo completo pesato non orientato di N nodi, trovare il percorso minimo che partendo dal nodo i, visiti tutti i nodi del grafo.

Formato dell'input

La prima linea contiene il numero di nodi N. Le $N-1$ linee successive contengono i pesi w degli archi: La prima riga contiene il peso dell'arco da 1 a 0; la seconda riga contiene i pesi degli archi da 2 a 0 e 1, la terza riga contiene i pesi degli archi da 3 a 0, 1 e 2, ecc. .

Formato dell'output

L'output può contenenere varie soluzioni, ognuna descritta da una riga terminata dal simbolo `#`. Il correttore prenderà **l'ultima riga terminata da `#`** come soluzione da valutare. La soluzione è costituita da $N+1$ interi, ovvero il percorso di nodi da visitare.

Assunzioni

- $3 < N \leq 50$
- $0 \leq w \leq 1000$

Istruzioni per l'output

Se scrivete una soluzione esponenziale (tipo branch and bound) importate `tsp.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.
Il correttore considererà l'ultima riga di output che finisce con `#`. Quindi, anche se non appendete soluzioni multiple, terminate l'output con `#`.

Requisiti tecnici

Il main va sempre dichiarato come `int main()` o `int main(void)`. Questo esercizio deve essere svolto in C++, non è possibile usare il C.

```
... include delle libreria di sistema ...
#include "tsp.h"

int main() {
...

  return 0;
}
```

è importante che il main termini correttamente con un'istruzione `return 0`.

Istruzioni di compilazione

Di seguito riportiamo le istruzioni per testare i vostri progammi su vari sistemi. Si suppone che il sorgente con il vostro codice si chiami file `got.cpp`. I file `got.cpp`, `grader.cpp` e `got.h` devo stare nella stessa cartella.

Sistemi GNU/Linux

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

Sistemi Mac OS X

Su sistemi Mac OS X usate il seguente comando di compilazione:

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

Se ottente un errore del tipo: `use of undeclared identifier quick_exit`, sostituite in `grader.cpp` l'istruzione `quick_exit(EXIT_SUCCESS);` con `exit(EXIT_SUCCESS);`.

Sistemi Windows

Per il sistema Windows 10 potete installare il "Windows Subsystem for Linux"[1]. Successivamente potete installare i tool necessari per usare Visual Studio Code[2] o Visual Studio 2017[3] seguendo le relative guide riportate nelle note. Usando questo sistema fate attenzione a dove salvate i file e a quale nome gli date in quanto potreste avere delle difficoltà con percorsi che contengano spazi e caratteri speciali.
In alternativa, o per sistemi precedenti a Windows 10 potete installare *Cygwin*[4], un ambiente completamente POSIX-compatibile per Windows. Anche in questo caso esistono guide per configurare i comuni editor disponibili su Windows di modo che utilizzino l'ambiente Cygwin, come per esempio Visual Studio[5].
Una volta installato Cygwin è possibile simulare quanto avviane su arena compilando ilproprio sorgente senza includere l'header `got.cpp` e il grader `grader.cpp`:

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

e lanciare il comando come:

```
timeout.exe 3 ./got
```

`timeout.exe` arresterò il programma dopo 3 secondi.

Esempi di input/output

File `input.txt`	File `output.txt`
4 1 1 3 2 4 1	0 1 3 2 0#

[1] https://docs.microsoft.com/en-us/windows/wsl/install-win10
[2] https://code.visualstudio.com/docs/cpp/config-wsl
[3] https://devblogs.microsoft.com/cppblog/targeting-windows-subsystem-for-linux-from-visual-studio/
[4] https://www.cygwin.com/
[5] https://devblogs.microsoft.com/cppblog/using-mingw-and-cygwin-with-visual-cpp-and-open-folder/