Tema 1 – Proiectarea algoritmilor (Seria CD)

Deadline: 19.05.2020

David Iancu – davidiancudvd@gmail.com, Mihai Nan – mihai.nan.cti@gmail.com

Anul universitar 2019 – 2020

1 Holograme – 50 de puncte

1.1 Enunț

Gigica locuiește într-un oraș în care fiecare locuință e notată cu un număr de la 1 la N. Având în vedere că e pandemie și oamenii nu vor să iasă din casă, și-au instalat noile telefoane cu holograme cu care se pot vedea între ei. Hologramele funcționează într-un mod interesant: dacă persoana A își cumpără o hologramă cu care poate vorbi cu persoana B și persoana B are o hologramă cu care poate vorbi cu persoana C, și persoana A poate vorbi cu persoana C, în schimb dacă persoana B dorește să discute cu persoana A, nu se poate baza doar pe faptul că are o hologramă pentru B, și că A are o hologramă pentru el, trebuie să aibă și el o hologramă pentru A. Altfel spus, dacă A vorbește cu B, nu înseamnă că și B poate vorbi cu A, în schimb dacă A vorbește cu B și B vorbește cu C, atunci și A vorbește cu C. Se cunoaște o listă de perechi de case care doresc să comunice una cu cealaltă. Pentru că hologramele sunt scumpe, dorim să minimizăm numărul de holograme cumpărate pentru tot orașul, astfel încât fiecare pereche din listă să poată vorbi una cu alta.

1.2 Precizări

Fișierul de intrare `holograme.in` conține pe prima linie numărul de orașe (N) și numărul de perechi (M). Pe următoarele M linii sunt date perechile.

Fișierul de ieșire `holograme.out` va conține pe prima linie numărul minim de holograme cumpărate.

1.3 Restricții

- $1 \leq N \leq 10000$
- $1 \leq M \leq 20000$
- Timp maxim de execuție: **1 secundă / test**
- Limbajul acceptat: **C / C++**

1.4 Exemple

Input

```
4 6
1 2
2 3
3 4
1 3
1 4
2 4
```

Output

```
3
```

E suficient să luam o hologramă de la 1 la 2, o hologramă de la 2 la 3 și una de la 3 la 4. În acest fel, 1 poate comunica și cu 3 și cu 4, iar 2 poate comunica și cu 4.

Input

```
4 12
1 2
2 3
3 4
1 3
1 4
2 4
4 1
4 2
4 3
3 1
3 2
2 1
```

Output

```
4
```

Dacă luăm în plus față de exemplul trecut o hologramа de la 4 la 1, putem acoperi toate perechile de noduri posibile.

2 Scurt și la obiect – 50 de puncte

2.1 Enunț

Pentru că știm că studenților nu le plac cerințele lungi și, în plus, tocmai au învățat despre arbori minimi de acoperire, li se dă următoarea cerință: dându-se un graf cu N noduri și M muchii, ce muchii apar în toți arborii minimi de acoperire care se pot forma și ce muchii apar doar în o parte din arborii minimi de acoperire?

2.2 Precizări

Fișierul de intrare `scurt.in` conține pe prima linie numerele N și M, apoi cele M muchii, date sub forma (x, y, c), semnificând că avem o muchie bidirecțională de la x la y, de cost c.

Fișierul de ieșire `scurt.out` va conține pe primul rând 2 numere, N_1 și N_2, apoi N_1 linii cu perechi de muchii care apar în toți arborii de acoperire, apoi N_2 perechi cu muchii care apar cel puțin într-un arbore minim de acoperire, dar nu în toți. Muchiile trebuie afișate în ordinea citită din fișier, **unele muchii pot lipsi (cele care nu apar în niciun ciclu)**.

2.3 Restricții

- $1 \leq N \leq 5000$
- $1 \leq m \leq 1000000$
- $1 \leq c \leq 1000$
- Timp maxim de execuție: **1 secundă / test**
- Limbajul acceptat: **C / C++**

2.4 Exemple

Input	*Output*
5 7 1 2 1 2 3 1 3 4 2 4 1 2 1 5 3 4 5 3 2 5 6	2 4 1 2 2 3 3 4 4 1 1 5 4 5

Se pot forma 4 arbori minimi posibili de acoperire:

1. folosind muchiile $(1, 2), (2, 3), (3, 4), (1, 5)$;
2. folosind muchiile $(1, 2), (2, 3), (4, 1), (1, 5)$;
3. folosind muchiile $(1, 2), (2, 3), (3, 4), (4, 5)$;
4. folosind muchiile $(1, 2), (2, 3), (4, 1), (4, 5)$;

3 Umbre – 50 de puncte

3.1 Enunț

Se știe că mafia face contrabandă cu diverse tipuri de substanțe. Prietenii noștri **Relu** și **Sabin** au următoarea misiune: pornesc împreuna din orașul 1 și trebuie să ajungă la destinație, în orașul N, cumpărând toate tipurile de substanțe de care are nevoie **Capitanul**. Cei doi pot merge separat, dar important e ca la final să ajungă în orașul N. Ei cunosc drumurile și distanțele dintre orașe, exprimate în ore, dar și toate tipurile de substanțe pe care le deține un oraș. Cei doi doresc să ajungă în orașul final, N, cumpărând toate cele X tipuri de marfă de care au nevoie, într-un timp minim. Se dorește aflarea acestui timp minim, știind că nu există restricții în privința traseului pe care îl fac cei 2, atâta timp cât ajung la final la nodul N (nu trebuie să ajungă neapărat în același timp).

3.2 Precizări

Fișierul de intrare `umbre.in` conține pe prima linie trei numere naturale: N, M, X. Apoi pe următoarele N linii se află o listă cu numărul de substanțe și indecșii corespunzători substanțelor care se află în fiecare oraș (de exemplu 5 1 2 3 4 5 înseamnă că avem 5 substanțe cu indicii $1, 2, 3, 4$ și 5). Pe ultimele M linii avem drumurile dintre orașe, date sub forma (a, b, c), semnificând că avem un drum bidirecțional de la orașul a la orașul b cu costul c.

Fișierul de ieșire `umbre.out` va conține un număr semnificând timpul minim în care cei doi ajung în orașul N. Dacă ei ajung în orașul N în timpi diferiți, se consideră timpul în care a ajuns ultimul (cel mai mare timp dintre cele două).

3.3 Restricții

- $2 \leq N \leq 1000$
- $2 \leq M \leq 5000$
- $1 \leq K \leq 10$
- Fiecare oraș poate vinde între 0 și K tipuri de marfă.
- Lista pentru fiecare oraș nu conține duplicate.
- Timp maxim de execuție: **1 secundă / test**
- Limbajul acceptat: **C / C++**

3.4 Exemple

Input

```
7 6 7
1 1
1 2
1 3
1 4
1 5
1 6
1 7
1 2 1
2 3 2
3 4 3
4 7 4
1 4 5
4 5 6
5 6 7
6 7 8
```

Output

```
26
```

Sabin va merge pe calea mai scurtă, $1-2-3-4-7$, cu timp total 10. Relu va lua orașele mai greu de parcurs și va ajunge la destinație după 26 de ore fără somn (*it's a tough guy*).

4 Informații importante

- Temele vor fi testate automat folosind platforma vmchecker.
- Singurele limbaje acceptate pentru realizarea temei sunt C/C++.
- Va trebui să încărcați o arhivă având formatul **zip** care să conțină următoarele fișiere:
  - Fișierul / fișierele sursă cu implementările problemelor
  - Fișierul `Makefile` (vezi indicațiile)
  - Fișierul `README` (vezi indicațiile)
- Fișierul `Makefile` trebuie să conțină următoarele reguli:
  - `build` – care va compila sursele și va genera executabilele;
  - `run-p1` – care va rula executabilul pentru problema 1;
  - `run-p2` – care va rula executabilul pentru problema 2;
  - `run-p3` – care va rula executabilul pentru problema 3;
  - `clean` – care va șterge executabilele generate;
- Temele o să fie verificate doar folosind sistemul de operare Linux!
- Numele regulilor, din fișierul `Makefile`, trebuie să fie exact cele de mai sus. Absența sau denumirea diferită a acestora va avea drept consecință obținerea unui punctaj de **0 puncte** pe testele asociate problemei rezolvate de regula greșită.
- Fișierul `README` trebuie să conțină pentru fiecare problemă rezolvată următoarele:
  - explicația implementării realizate, prin evidențierea tehnicii de programare folosită;
  - estimarea complexității pentru soluția propusă.
- Absența fișierului `README` se penalizează cu o depunctare de **-10 puncte**.

Important

- **Nu este permisă compilarea cu opțiuni de optimizare a codului (`-O1`, `O2`, etc.).**
- **Tema este individuală! Toate soluţiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.**
- **Temele vor fi punctate doar pentru testele care sunt trecute pe `vmchecker`.**