Úvod

všetky 3 podúlohy som riešil pomocou sat solveru, ktorý sme spomenuli ako prvý, tj picosat, ale neskôr sa ukázalo, že potrebeujem pri Hamiltonovskej kružnici vypísať všetky riešenia, tak som potreboval relsat. Dĺžky výpočtov pre všetky prípady boli vskutku krátke tak cca do 5 sekund, ale ako na konci spomínam, pre f-1 farieb to bolo v niektorých prípadoch aj 5 minut.

Podúloha A

Tu stačí dodať pen podmienku, že nejaká dáma musí byť na políčku 47,47, tj premenná 4748 musí byť true. Teda trebárs na koniec súboru vygenerovaného naším python programčekom zo zadania dopíšeme podmienku 4748 0 a pustíme picosat a vypluje nám riešenie, ktore ešte preženieme cez niečo čo nám z toho vyrobí späť súradnice a dostávame

(0,7), (1,19), (2,90), (3,44), (4,60), (5,80), (6,78), (7,73), (8,75), (9,48), (10,65), (11,51), (12,59), (13,61), (14,29), (15,45), (16,54), (17,34), (18,77), (19,40), (20,98), (21,2), (23,99), (23,49), (24,53), (25,23), (26,16), (27,79), (28,25), (29,9), (30,1), (31,18), (32,95), (33,36), (34,69), (35,67), (36,37), (37,97), (38,17), (39,96), (40,28), (41,26), (42,70), (43,86), (44,38), (45,11), (46,15), **(47,47)**, (48,94), (49,82), (50,64), (51,88), (52,41), (53,13), (54,4), (55,10), (56,92), (57,43), (58,62), (59,22), (60,3), (61,72), (62,57), (63,35), (64,76), (65,32), (66,24), (67,58), (68,21), (69,74), (70,31), (71,20), (72,6), (73,50), (74,42), (75,81), (76,89), (77,55), (78,8), (79,0), (80,56), (81,83), (82,91), (83,5), (84,12), (85,68), (86,85), (87,39), (88,63), (89,71), (90,27), (91,33), (92,84), (93,52), (94,87), (95,46), (96,66), (97,93), (98,30), (99,14)

Podúloha B

Fu asi najťahšia z tých 3 podúloh. Tu si budeme reprezentovať hrany ako premenné od 1 po h - počet hrán. Chceme, aby po prejdení sat solverom boli hrany po ktorých prejdeme true a ostatné hrany false.

Myšielka je jednoduchá: Každý vrchol musí mať práve 2 hrany true - len vtedy budú hrany tvoriť Hamiltonovskú kružnicu. No pre sat je to trochu zložitejšie. Tuto tú podmienku vieme formulovať nasledovne: Neexistuje žiadka trojica hrán, ktoré majú spoločný vrchol a sú true a súčasne existuje aspoň jedna dvojica hrán, ktoré majú spoločný vrchol a sú true. Vieme, že táto podmienka vlastne dokopy s predcháduajúcou znie, že existuje práve jedna dvojica, ktorá má spoločný vrchol a sú true, pretože keby týchto dvojíc bolo viac, tak potom určite by sme našli trojicu. Takže toto platí.

Podmienky typu nesmie existovať trojica medzi hranami h_1,h_2,h_3 môžme pre sat formulovať ako $-h_1$ $-h_2$ $-h_3$

Horšie je to pre podmienky kde chceme aby bola aspoň jedna dvojica tj h_1 a h_2, pretože nech z vrchola vychádza h hrán, tak musí plaiť (h_1 a h_2) alebo (h_1 a h_3) alebo ... alebo (h_{n-1} a h_n)
Nanešťastie cnf pozná len podmienky tvaru (X_1 alebo Y_1) a(X_2 alebo Y_2)...a(X_n alebo Y_n)
Ale keďže každá tvar sa dá zapísať do CNF tak aj tento nieje výnimkou. Takýto tvar vieme prehodiť do CNF tak, že budeme mať 2^n podmienok tvaru (p_1 alebo p_2... p_n) kde pi je bud X_i alebo Y_i. Tento návod sa dá ľahko nájsť na internete.

Takže už vieme zapísať podmienky v cnf, preto ak zostávame na Slovensku tak tu sa nikomu chceme ručne spracovávať graf a prepisovať ho do cnf, preto si poďme spraviť nejaký program. V pythone ho robiť nebudem - je moc pomalý na vypisovanie 150MB súborov, preto ho radšej napíšem v C++.

```
#include<iostream>
```

```
#include<vector>
using namespace std;
vector<vector<int>>V;

int main(){
        int n,h;
        cin>>n>>h;
        V.resize(n+47);

        for(int i=1; i<=h; i++){
                int a,b;
                cin>>a>>b;
                V[a].push_back(i);
                V[b].push_back(i);
        }

        int poc=0;
        for(int i=0; i<n; i++){
                poc+=(V[i].size()*(V[i].size()-1)*(V[i].size()-2))/6;
                poc+=1<<((V[i].size()*(V[i].size()-1))/2);
        }

        cout<<"p␣cnf␣"<<h<<"␣"<<poc<<endl;

        for(int i=0; i<n; i++){
                vector<pair<int,int>>P;
                for(auto it=V[i].begin(); it!=V[i].end(); it++){
                        for(auto it1=it+1; it1!=V[i].end(); it1++){
                        P.push_back(make_pair(*it,*it1));
                        for(auto it2=it1+1; it2!=V[i].end(); it2++)
                        cout<<-*it<<"␣"<<-*it1<<"␣"<<-*it2<<"␣0\n";
                        }
                }
                for(int s=0; s<(1<<P.size()); s++){
                        int p=s;
                        for(int j=0; j<P.size(); j++){
                                if(p%2==1)cout<<P[j].second<<"␣";
                                else cout<<P[j].first<<"␣";
                                p/=2;
                        }
                        cout<<"0\n";
                }
        }
}
```

Ako som pri písaní tohoto zistil, že podmienky niesú správne, aby sat solver vyhodil len tie hrany ktoré tvoria hamiltonovskú kružnicu, ale vyhodí riešenie, ktoré pozostáva z niekoľkých kružníc, čo znamená nesprávne riešenie. Ja som si však povedal, že už sa mi nechce vymýšľať niečo lepšie, preto som si radšej stiahol relsat - ktorý mi vyhodí všetky riešenia a už len overím či niektoré je z nich správne.

```
#include<iostream>
```

```
#include<vector>
using namespace std;
vector<pair<int,int>>V;
int main(){

int n,h;
cin>>n>>h;
V.resize(h+47);

for(int i=1; i<=h; i++)
cin>>V[i].first>>V[i].second;

string s;
while(cin>>s){
cin>>s;
vector<pair<int,int>>T;
for(int i=1; i<=n; i++){
int t;
cin>>t;
T.push_back(V[t]);
}
int start=T[0].second;
int pos=0;
int last=T[0].first;
int poc=1;
cout<<last<<"␣";
while(last!=start){
poc++;
for(int j=0; j<T.size(); j++)
    if((T[j].first==last || T[j].second==last) && j!=pos){
    if(T[j].first==last){pos=j;last=T[j].second; break;}
    else {pos=j;last=T[j].first; break;}
    }
cout<<last<<"␣";
}
if(poc==n)cout<<"dobre"<<endl;
else cout<<"zle"<<endl;
}

}
```

Funguje to asi tak, že najskôr mu dám na vstup graf zo súboru a potom mu hádžem riešenia zo satsolveru. Pre každé mi vypíše, či je správne alebo nie a vypíše mi aj cestovanie po vrcholoch, takže tie sem hneď môžem napísať.

ham25:

15 3 22 14 16 11 21 8 7 13 19 2 20 23 9 17 12 18 5 10 1 4 24 0 6

ham30:

5 11 26 23 3 4 14 25 17 9 10 18 28 6 22 29 2 0 21 19 13 1 20 7 27 24 12 8 15 16

ham35:

31 1 6 34 2 15 28 7 30 32 23 4 14 20 10 18 22 13 29 17 9 11 24 19 0 3 25 16 26 21 5 12 33 8 27

ham40:
9 11 8 28 33 36 1 16 32 25 18 20 17 13 4 0 27 37 29 12 30 3 38 14 31 24 10 7 34 2 21 26 35 23 19 39 15 5 6 22

Podúloha C

V tejto úlohe chceme nájsť počet farieb. každý sat slver nám však vráti true, alebo false či riešenie extistuje, takže nám rozhodne nevráti najmenší počet farieb. My však vieme, že ak riešenie existuje pre f farieb, tak bude existovať aj pre $f+1$ farieb. A navyše ak riešenie pre f farieb neextistuje, tak nebude existovať ani pre $f-1$ farieb. Ako správych informatikov nám hned napadne používať binárne vyhľadávanie. Vieme, že pre n farieb to ide, kde n je počet vrcholov a pre 1 frabu nie. Riešenie potom bude počet farieb f takých, že pre f farieb riešenie existuje, ale pre $f-1$ nie.
Ako však zapíšeme graf pomocou cnf ?? No pre každý vrchol si vytvoríme f premenných, kde každá premenná prestavuje nejakú z f farieb, ktoré momentálne skúšame. Potom musí platiť, že medzi týmito f farbami jedného vrchola musí byť práve 1 true - vrchol nemôže byť ofarbený žiadnou, alebo aspoň 2 farbami. Potom ešte platí, že ak existuje hrana medzi dvoma vrcholmi, tak potom niesú rovnaké farby týchto dvoch vrcholov. Toto vieme jednoducho zapísať v cnf a použijem na to môj program:

```
#include<iostream>

using namespace std;

int main(){

int n,h,f;
cin>>n>>h;
f=11;
cout<<"p␣cnf␣"<<n*f<<"␣"<<n*(f+((f-2)*(f-1))/2)+h*f<<endl;

for(int i=0; i<n; i++){

for (int j=i*f+1; j<=i*f+f; j++)
    cout<<j<<"␣";
cout<<"0"<<endl;

for(int j=i*f+1; j<=i*f+f; j++){

for(int k=j+1; k<=i*f+f; k++)
    cout<<-j<<"␣"<<-k<<"␣0\n";
}

}


for(int i=1; i<=h; i++){
int a,b;
cin>>a>>b;
for(int j=1; j<=f; j++)
    cout<<-(f*a+j)<<"␣"<<-(f*b+j)<<"␣0\n";
}
```

```
}
```

Toto je napr. príkald pre počet farieb 11 - výsledok najmenšieho vstupu.

Takže pekne binárne vyhľadávame a dostávame výstupy:

col020:11
col100:6
col120:9
col121:6
col138:11
col999:3

Poznámka: pre testovanie f-1 farieb, kde f je riešenie, to trvalo hodne dlho v niektorých prípadoch niektoré zbehli hned, ale na jedno som čakal cca 30 min.