1. feladat – Benzinkutak **100 pont**

Gigel egy olyan városban lakik, melyben N útkereszteződés található, 1-től N-ig számozva. A kereszteződéseket M kétirányú utca köti össze. N-ből P kereszteződésben benzinkút épült. Gigel szeretné tudni, hogy az egyes útkereszteződésekhez mely benzinkutak vannak a legközelebb. Amennyiben egy kereszteződés két benzinkúttól is ugyanolyan távolságra található, Gigel a kisebb sorszámmal rendelkező benzinkutat fogja választani.

Követelmény

Írj programot amely megkeresi az egyes útkereszteződésekhez legközelebb eső benzinkutakat.

Bemeneti adatok

A *benzinarii.in* bemeneti állomány első sora az `N`, `M` és `P` értékeket tartalmazza. A következő `M` sorban az X és Y útkereszteződéseket összekötő kétirányú útaknak megfelelő X, Y számpárok jelennek meg. Az utolsó sorban megjelenő `P` szám a benzinkúttal rendelkező útkereszteződéseket jelzi.

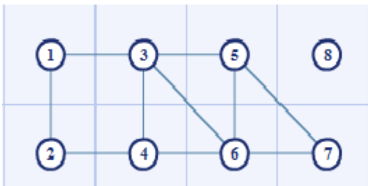
Kimeneti adatok

A *benzinarii.out* kimeneti állomány egyetlen sorbol fog állni, amely *N* számot tartalmaz. Az *i*-edik szám értéke -1, ha az *i*-edik útkereszteződésből egyetlen benzinkúthoz sem lehet eljutni. Ellenkező esetben az *i*-edik szám értéke a legközelebbi benzinkúttal rendelkező útkereszteződés száma lesz.

Megkötések

- Két útkereszteződés közötti távolság alatt azon utcák számát értjük, melyeken át kell haladnunk ahhoz, hogy egyik kereszteződésből a másikba érjünk
- $1 \leq N \leq 100\ 000$; $1 \leq M \leq 200\ 000$; $1 \leq P \leq 1\ 000$
- A tesztek 50%-a esetében: $1 \leq N \leq 1000$; $1 \leq M \leq 2000$; $1 \leq P \leq 100$

Példa

benzinarii.in	benzinarii.out	Magyarázat
8 10 2 1 2 1 3 2 4 3 4 3 5 3 6 4 6 5 6 5 7 6 7 7 1	1 1 1 1 7 7 7 -1	 2 benzinkút van: egy az 1-es és egy a 7-es útkereszteződésnél. Az 1-es, 2-es, 3-es és 4-es útkereszteződések az 1-es kereszteződésnél található benzinkúthoz vannak közelebb. Az 5-ös, 6-os és 7-es kereszteződések közelebb esnek a 7-es benzinkúthoz. A 4-es kereszteződés mindkét benzinkúttól egyforma távolságra található, viszont az 1-es benzinkutat tekintjük közelebbinek, mivel 1 < 7.

Maximális lefutási idő: 1 másodperc
Memóriakorlátozás: 32 MB, melyből 8 MB a verem számára **Forráskód maximális mérete**: 10 KB.

2. feladat – Hashtag **100 pont**

Yolo és Swag egy új, Hashtag nevű játékot játszanak. A játékhoz N, egyenként 1-től N-ig számozott négyzet és két bábu szükséges. Kezdetben Yolo bábuja az 1-es négyzeten, míg Swag bábuja a K-adik négyzetben található. A játékosok minden kör elején egy pénzérmét dobnak fel. Ha a címer kerül felül, akkor Yolo bábuja lép egyet előre, ellenkező esetben viszont Swag bábuja halad. Yolo akkor nyer, hogyha a bábujával utoléri Swag bábuját, míg Swag akkor nyerheti meg a játékot, hogyha sikerül eljuttatnia bábuját az N-edik négyzetig.

Követelmény

Swag szeretné tudni, hogy adott `N` és `K` értékek esetén hány lehetséges módon nyerhet, azaz hány különböző olyan játékmenet képzelhető el, melyből végül Swag kerül ki győztesként. Két játékmenetet akkor tekintünk különbözőnek, ha azok legkevesebb egy lépésben különböznek egymástól.

Bemeneti adatok

A *hashtag.in* bemeneti állomány első sora 2 szóközzel elválasztott természetes számot tartalmaz (a leírásban említett `N` és `K`).

Kimeneti adatok

Mivel a lehetséges játékmenetek száma nagyon nagy, a *hashtag.out* kimeneti fájlba egyetlen természetes szám kerül: a Swag által nyerhető játékok száma modulo `666013` (az eredmény 666013-mal történő osztási maradéka).

Megkötések

- $1 \le K \le N \le 2000$
- A tesztek 40%-a esetében: $1 \le K \le N \le 15$
- A tesztek 35%-a esetében a lehetséges játékmenetek száma kisebb mint 666013
- Swag felhívja a figyelmeteket arra, hogy (A+B) % 666013 = (A%666013+B%666013) % 666013

Példa

hashtag.in	hashtag.out	Explicații
4 2	2	Kezdetben Yolo bábuja az 1., míg Swag bábuja a 2. négyzetben található. Két olyan játékmenet létezik, mely Swag győzelmével zárul: az első játékban Swag egymás után kétszer lép (SS) és a 3., majd a 4. négyzetbe ér; a második játékban Swag lépését Yolo lépése követi, majd végül ismét Swag lép (SYS), azaz Swag bábuja a 3. négyzetbe kerül, majd Yolo bábuja a 2. négyzetbe lép, végül pedig Swag a 4. négyzetbe lépve megnyeri a játékot.
5 2	5	Swag 5 lehetséges módon nyerhet: SSS, SYSS, SYSYS, SSYS, SSYYS.
7 4	28	Swag 28 különböző módon nyerheti meg a játékot.

Maximális lefutási idő: 0.3 másodperc
Memóriakorlátozás: 32 MB, melyből 16 MB a verem számára
Forráskód maximális mérete: 10 KB.

Benzinării - Descrierea soluției

prof. Lupșe-Turpan Mircea,
Liceul Teoretic Grigore Moisil Timișoara

Varianta 1 (100 puncte) - Complexitate O(N+M+P log P)

Se utilizează un algoritm BFS cu noduri de plecare multiple. Toate intersecțiile în care sunt construite benzinării sunt adăugate inițial în coada BFS. Se reține în memorie un vector Sol[i] cu semnificația "benzinăria cea mai apropiată de intersecția i". Se observă că pentru a respecta precizarea cu determinarea benzinăriei cu număr minim este suficientă o sortare inițială a benzinăriilor.

Varianta 2 (100 puncte) - Complexitate O(N+M)

Aceași rezolvare ca și la varianta 1, dar benzinăriile nu se sortează inițial, urmând ca la fiecare pas din BFS să se facă o verificare în plus și eventual o actualizare.

Varianta 3 (70 puncte) - Complexitate O((N+M)*P)

Se aplică un algoritm BFS pentru fiecare benzinărie în parte.

Varianta 4 (50 puncte) - Complexitate O((N+M)*N)

Se aplică un algoritm BFS din fiecare intersecție.

O soluție care implementează graful cu ajutorul matricei de adiacență poate obține cel mult 50 puncte.

O soluție care nu ține cont de precizarea cu determinarea benzinăriei care se află în intersecția cu număr minim poate obține cel mult 30 puncte.

Hashtag - Descrierea soluției

stud. Okros Alexandru,
Universitatea Politehnica Timișoara

Varianta 1 (35-40 puncte) - Complexitate $O(2^N)$ (Mircea Lupșe-Turpan)

Se utilizează un algoritm backtracking. Se generează un șir binar, în care valoarea 0 corespunde mutării pionului lui Yolo, iar valoarea 1 corespunde mutării pionului lui Swag. O soluție parțială este validă dacă diferența dintre numărul de mutări al lui Yolo (numărul de zerouri) și numărul de mutări al lui Swag (numărul de valori de 1) este mai mică decât K-1, unde K-1 reprezintă diferența inițială de căsuțe dintre Yolo și Swag. O soluție parțială este și soluție dacă numărul de mutări al lui Swag este N-K (diferența dintre poziția N și poziția inițială a lui Swag), adică numărul de mutări necesare lui Swag pentru a ajunge pe poziția N.

Varianta 2 (35-40 puncte) - Complexitate $O(2^N)$ (Mircea Lupșe-Turpan)

Se poate optimiza algoritmul de mai sus prin renunțarea la generarea șirului binar și prin utilizarea a două variabile YoloMoves și SwagMoves care rețin în orice moment numărul de mutări al lui Yolo, respectiv numărul de mutări al lui Swag. În acest fel verificarea dacă o soluție parțială este validă, respectiv verificarea dacă o soluție parțială este soluție se face prin compararea acestor 2 variabile.

Varianta 3 (40 puncte) - Complexitate $O(2^N)$

Se bazează pe programare dinamică. Se scrie funcția L(A, B) care calculează numărul de jocuri câștigate de Swag dacă Yolo începe pe poziția A și Swag pe poziția B.

Recurența este L(A,B) = L(A+1,B) + L(A, B+1). Trebuie sa calculăm L(1, K), deoarece Yolo începe pe poziția 1, iar Swag pe poziția K.

Varianta 4 (100 puncte) - Complexitate $O(N^2)$

Este optimizarea soluției precedente folosind memoizare. Valorile deja calculate se rețin într-o matrice, astfel că se evită calcularea unor valori de doua ori.

Complexitatea de memorie $O(N^2)$.

Varianta 5 (100 puncte) - Complexitate $O(N^2)$

Folosim programare dinamica. DP[i][j] = numărul de jocuri câștigate de Swag dacă el e pe poziția j, iar jocul se termină pe poziția i. Recurența: DP[i][j] = DP[i][j+1] + DP[i-1][j-1] (primul caz Swag se mută o căsuță la dreapta, cazul doi Yolo se mută o căsuță și astfel swag și finish-ul se aproprie cu o căsuță, de unde "-1" la parametri).

Complexitatea de memorie $O(N^2)$.

Varianta 6 (100 puncte) - Complexitate $O(N^2)$

La soluția precedentă folosim doar ultimile două linii ale matricei și observăm că putem obține număr de operații N*(N-K) deoarece nu trebuie calculate toate valorile din matrice (rezultă din recurență).

Prin utilizarea unei matrici cu 2 linii si N coloane, complexitatea de memorie scade la O(N).