Szinte már közismert, hogy egy gráfban Hamilon-kör keresése, vagy egyáltalán a létezésének megállapítása NP-teljes feladat, csak $O(c^N)$ idejű algoritmussal oldható meg determinisztikuan. Kétségkívül a legjobb (legkevesebb kereszteződésből álló) megoldás a gráf egy Hamilton-köre lenne, de mivel ennek keresése — ha egyáltalán létezik — exponenciális idejű algoritmust igényelne, ezért megelégszünk egy egyszerűbb módszerrel is, ami $O(N)$ helyett legfeljebb $O(N^2)$ elemű megoldást ad.

Ez a módszer pedig a "mindig a legközelebbi bejáratlan" elve. Az algoritmus végtelenül egyszerű: a pillanatnyi csúcsból, ahol éppen állunk, a legközelebbi olyan csúcsra megyünk, ahol még nem jártunk, természetesen figyelembe véve a megfordulási tilalmat.

A csúcsok mindegyikéről logikai változókkal jelezhetjük, hogy azokat már bejártuk (kiírtuk) vagy sem. Az algoritmus végét az a pillanat jelenti, mikor minden kereszteződés bejárttá válik. A legközelebbi megfelelő csúcs keresése szélességi keresés segítségével történik; súlyozatlan és irányítatlan gráfokban ez is megteszi, nem kell Dijkstra algoritmusához folyamodnunk.

Azt könnyen bizonyíthatjuk, hogy az algoritmus kimenete legfeljebb $O(N^2)$ számot tartalmaz. (Minden bejárás legfeljebb N csúcsot érint, és legfeljebb N bejárásra van szükség.)

Azt azonban már nehezebb belátni, hogy a módszer tényleg működik, és a tolatási kritérium mellett is helyes megoldást ad. Ehhez először is be kell látnunk, hogy igazak a következő lemmák:

1. lemma: Legyen az utolsó bejárt él (u, v). Ha ez egy kör része, akkor a v-ből indulva a gráf minden csúcsa bejárható.

Bizonyítás: Tegyük fel, hogy a v csúcsból indulva szeretnénk a gráf valamelyik w csúcsába eljutni. Ezt megtehetjük az (u, v) él használata nélkül, mert a gráf ennek eltávolításával is összefüggő maradt, ugyanis ha egy él része egy körnek, akkor nem híd.

2. lemma: A város gráfjában bármely híd törlése a gráfot két olyan komponensre bonja, melyekben van legalább egy kör.

Bizonyítás: Tegyük fel, hogy nem így van. Különböztessünk meg két esetet. Ha az egyik komponens egyetlen csúcs, akkor ennek fokszáma a híd törlése előtt is csak 1 volt, ez pedig ellentmond a feladatnak. Ha a komponens több csúcsból áll, akkor — mivel egyik sincs benne egyetlen körben sem — ezek között az élek mind hidak, ebből következően a komponens fagráf. Egy irányítatlan fának pedig legalább 2 levele (1 fokszámú csúcsa van). Ha az egyik ilyen levél úgy keletkezett egy 2 fokszámú csúcsból, hogy töröltük a hidat, a másiknak akkor is 1 fokszámúnak kellett lennie; ez pedig megint ellentmond a feladat leírásának.

3. lemma: Legyen az utolsó bejárt él (u, v), ami híd. A v-t tartalmazó komponensnek van olyan bejáratlan csúcsa, ami benne van legalább egy körben.

Bizonyítás: A legutolsó bejárásnak nyilvánvalóan az u-t tartalmazó komponensből kellett indulnia, egyébként a $v, u, ..., u, v$ csúcssorozatot v-re cserélve egy rövidebb utat kapnánk. Annak, hogy az (u, v) hídon áthaladtunk, az az oka, hogy a v-t tartalmazó komponensben van bejáratlan csúcs. Tegyük fel, hogy a lemma következménye nem igaz, ekkor a komponens minden körének minden csúcsát bejártuk. Ehhez azonban az oda vezető összes hídon át kellett már haladnunk (különben nem kerülhettünk volna át a másik komponensbe), tehát azok sem lehetnek bejáratlan csúcsok. Következik ebből, hogy a komponens egyik csúcsa sem bejáratlan, ekkor viszont a keresés adott volna találatot komponensben; ellentmondás miatt a lemma igaz.

A fenti lemmák azt igazolják, hogy mindig találhatunk olyan bejáratlan csúcsot, ahová eljuthatunk úgy, hogy az utolsóként bejárt élet egyáltalán nem használjuk.

A programkód talán egyetlen különösebb magyarázatra szoruló elemei a `lastBFS` és `currentBFS` változók. A szélességi keresés (Breadth-First Search, BFS) fontos eleme, hogy jelzi a csúcsokról, hogy azokat a keresés érintette-e már. Ezt logikai változókkal szokás megoldani, de azokat minden keresés előtt törölni kell. Ha azonban a kereséseket megszámozzuk, akkor elég azt tárolni egy csúcsról, hogy melyik a legutolsó olyan bejárás, amiben részt vett; ezzel megspórolunk $O(N) * N$ inicializációt.

Az UTF-8 kódolású forráskód a „`cl s56.c`" paranccsal fordítható, vagy Visual C++ 2008/2010-zel egy Win32 konzol projektben.