

V tomto příkladu procházím zadaný orientovaný graf z prvního bodu všemi možnými cestami, dokud nenaleznu okružní cestu vyhovující zadání, nebo dokud nezjistím, že taková cesta neexistuje.

Křižovatky jsou uloženy jako struktury (záznamy) v poli, jehož index je zároveň číslem křižovatky. Tato struktura obsahuje množinu čísel křižovatek, na které z této křižovatky vedou cesty, množinu, která je podmnožinou skupiny první a která obsahuje čísla křižovatek na které jsme již v této “jždě” na této křižovatce odbočili a dvojici čísel, která udává zakázanou odbočku.

V algoritmu hrají zásadní roli tři datové struktury: zásobník, na který si ukládáme čísla projetých křižovatek, iterátor množiny neprojetých odboček křižovatky, který je pro každé “projetí” křižovatky lokální a proto je použit druhý zásobník, na který hodnotu tohoto iterátoru při každém projetí křižovatky ukládáme. Pokud by bylo místo řešení pomocí cyklu zvoleno řešení za pomoci rekurze, stačilo by danou proměnou mít jako lokální proměnou funkce a nemuseli bychom používat tyto zásobníky, ovšem domnívám se, že by v tomto případě byla rekurze méně vhodná. Ačkoli hovořím o zásobníku (stack), používám ve svém řešení STL kontejner typu vector, aby bylo jednodušší získat hodnotu která je pod vrcholem zásobníku, případně obsah zásobníku vypsát.

Na počátku je ukazatel momentální křižovatky nastaven na první křižovatku, její číslo uloženo na zásobník křižovatek a iterátor nepoužitých výjezdů je nastaven na první výjezd z křižovatky. Potom se začne provádět hlavní cyklus. V tom se nejprve zjišťuje, zda-li je možné jet dále, tzn. zda-li existuje neprojetý výjezd z křižovatky a zda náhodou není v tomto případě zakázaný. Pokud ano, uložíme do množiny použitých odboček této křižovatky číslo cílové křižovatky tohoto výjezdu, uložíme jej také na zásobník projetých křižovatek, na zásobník iterátorů uložíme náš iterátor množiny neprojetých křižovatek, iterátor nastavíme na první prvek množiny cest cílové křižovatky a cyklus opakujeme znovu. V druhém případě, kdy již nemůžeme pokračovat dále, zjistíme počet křižovatek uložených na zásobníku projetých křižovatek. V případě že je roven počtu cest v našem městě jsme našli řešení, opustíme tedy cyklus a vypíšeme obsah zásobníku. Pokud je roven jedné,

jedná se o případ kdy jsme se octli zpátky na počáteční křižovatce a nemůžeme pokračovat dále – okružní cesta neexistuje. V ostatních případech se musíme vrátit na předchozí křižovatku: obnovíme ze zásobníku iterátor předchozí křižovatky, snížíme zásobník projetých křižovatek a začneme cyklus znovu pro předchozí křižovatku. Cyklus je prováděn tak dlouho, dokud nenastane případ že je cesta nalezena, nebo že je zjištěno že neexistuje.

Čas provádění algoritmu roste s počtem cest a počtem křižovatek.

Paměťová složitost je lineární vzhledem k počtu křižovatek a počtu cest.

```
#include <iostream>
#include <fstream>

#include <set>
#include <vector>

#define NO_PATH_FOUND "Okruzni jizda neexistuje.\n"

using namespace std;

class city
{
private:
    struct xroad {
        set<unsigned> unused;    // mnozina nepouzitych - povolenych vyjezdu
        set<unsigned> used;      // mnozina pouzitych - nepovolenych vyjezdu
        unsigned bad_from, bad_to; // dvojice zakazaneho odboceni
    } *cross;                  // jednotlivé křižovatky
    unsigned nroads;           // počet cest
    vector<unsigned> path;      // projita cesta
public:
    void add_road(unsigned from, unsigned to)
        { cross[from].unused.insert(to); nroads++; };
    void set_noturn(unsigned xroadno, unsigned from, unsigned to)
        { cross[xroadno].bad_from = from; cross[xroadno].bad_to = to; };
    bool find_path(vector<unsigned> **pathptr);
    city(unsigned ncross) { cross = new xroad[ncross]; nroads = 0; };
    ~city() { delete [] cross; };
};
```

```

bool city::find_path(vector<unsigned> **pathptr)
{
    bool done = false;
    bool path_exists = false;
    vector< set<unsigned>::iterator > path_iters;
    set<unsigned>::iterator turn;

    xroad *cur = &cross[0];
    path.push_back(0);
    turn = cur->unused.begin();

    while(!done){
        while(true){
            // iterator je na konci, nemame kam jet
            if(turn == cur->unused.end()){
                // nasli jsme cestu
                if(path.size() == nroads+1){
                    path_exists = true;
                    done = true;
                    break;
                }
                // nelze nalezt cestu
            }else if(path.size() == 1){
                path_exists = false;
                done = true;
                break;
            }
            // vratime se o krizovatku zpet
        }else{
            unsigned ournum = path.back();
            path.pop_back();
            // o krizovatku zpet
            cur = &cross[path.back()];
            // oznacime nasi cestu jako nepouzitou
            cur->used.erase(ournum);
            // obnovime iterator ze zasobniku
            turn = path_iters.back();
            //snizime zasobnik
            path_iters.pop_back();
            break;
        }
    }
}

```

```

// iterator není na konci
}else{
    // získáme předposlední hodnotu ze zásobníku, pokud možné
    unsigned previous;
    if(path.size() > 1){
        previous = *(path.end()-2);
    }
    //pokud cesta není použita ani zakázána, pokračujeme po ní
    if( (path.size() == 1
        && cur->used.find(*turn) == cur->used.end())
        ||
        ((cur->used.find(*turn) == cur->used.end())
        && !(previous == cur->bad_from
        && *turn == cur->bad_to))
        ){

        cur->used.insert(*turn);
        // dame na zásobník cílovou křižovatku
        path.push_back(*turn);
        // křižovatka pro další cyklus
        cur = &cross[*turn];
        // inkrementujeme iterator pro případ návratu
        turn++;
        // uložíme náš iterator na zásobník
        path_iters.push_back(turn);
        // vytvoříme nový iterator
        turn = cur->unused.begin();
        // přejdeme do dalšího cyklu
        break;
    }
}
turn++;
}
}

if(pathptr != NULL){
    *pathptr = (path_exists ? new vector<unsigned>(path) : NULL);
}
return path_exists;
}

```

```

int main(void)
{
    ifstream infile("okruh.in");
    ofstream outfile("okruh.out");

    unsigned xroads, roads;
    if(!(infile >> xroads >> roads)){
        cerr << "Neplatny vstup\n";
        exit(EXIT_FAILURE);
    }

    city my_city(xroads);

    //nacteme cesty
    for(unsigned i=0; i<roads; i++){
        unsigned from, to;
        if(!(infile >> from >> to)){
            cerr << "Neplatny vstup\n";
            exit(EXIT_FAILURE);
        }
        my_city.add_road(from-1,to-1);
    }

    //nacteme zakazane odbocky
    for(unsigned i=0; i<xroads; i++){
        unsigned from, to;
        if(!(infile >> from >> to)){
            cerr << "Neplatny vstup\n";
            exit(EXIT_FAILURE);
        }
        my_city.set_noturn(i,from-1,to-1);
    }

    vector<unsigned> *my_path;

```

```
if(my_city.find_path(&my_path)){
    vector<unsigned>::iterator i = my_path->begin();
    outfile << (*i)+1;
    for(i++; i != (my_path->end()-1); i++){
        outfile << ' ' << (*i)+1;
    }
    outfile << '\n';
}else{
    outfile << NO_PATH_FOUND;
}
}
```