PROHLEDÁVÁNÍ GRAFŮ

Karel Horák, Petr Ryšavý

16. března 2016

Katedra počítačů, FEL, ČVUT

Příklad 1

Nad frontou *(queue)* byly provedeny následující operace:

```
push(1)
push(2)
print(poll())
print(peek())
print(peek())
push(3)
print(poll())
print(poll())
print(peek())
push(4)
push(5)
print(poll())
print(poll())
```

Co bude obsahovat fronta po každém z volání a jaké hodnoty budou vypsány?

Řešení 1

Obsah fronty bude následující

```
1
1,2
2
2
2
2,3
3
-
-
4
4,5
5
-
```

Vytisklé hodnoty budou: `1,2,2,2,3,null,4,5`.

Příklad 2

Nad zásobníkem *(stack)* byly provedeny následující operace:

```
push(1)
push(2)
print(pop())
print(peek())
print(peek())
push(3)
print(pop())
print(pop())
print(peek())
push(4)
push(5)
print(pop())
print(pop())
```

Co bude obsahovat zásobník po každém z volání a jaké hodnoty budou vypsány?

Řešení 2

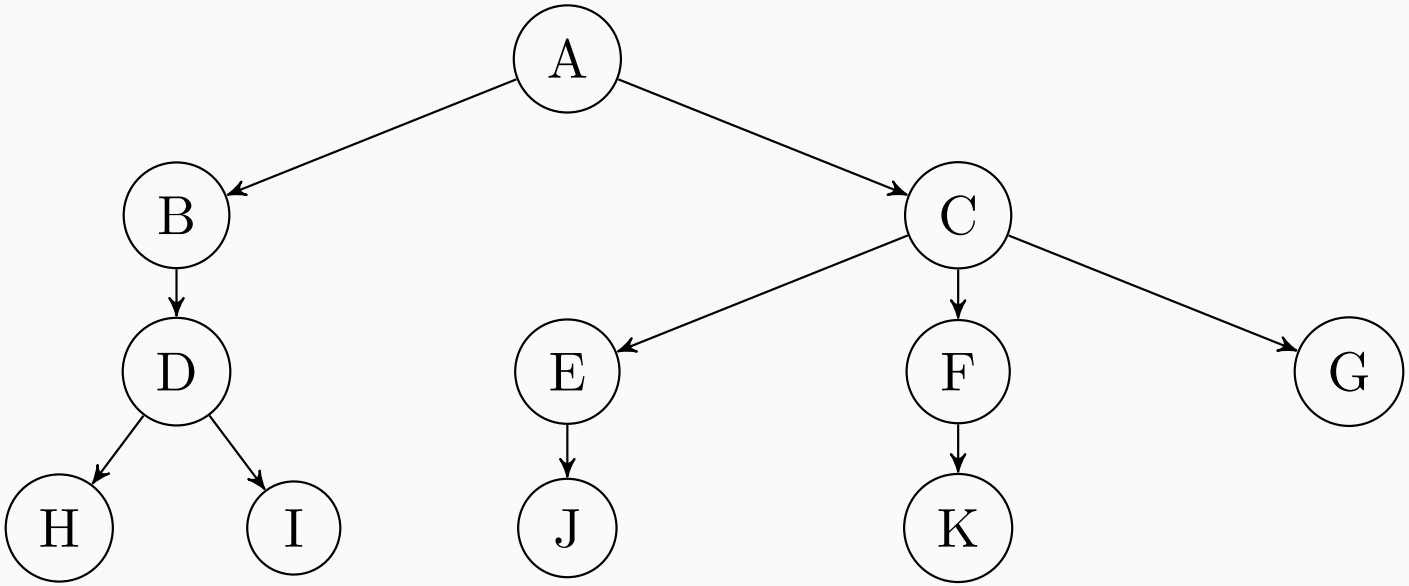
Obsah zásobníku bude následující

```
1
2,1
1
1
1
3,1
1
-
-
4
5,4
4
-
```

Vytisklé hodnoty budou: `2,1,1,3,1,null,5,4`.

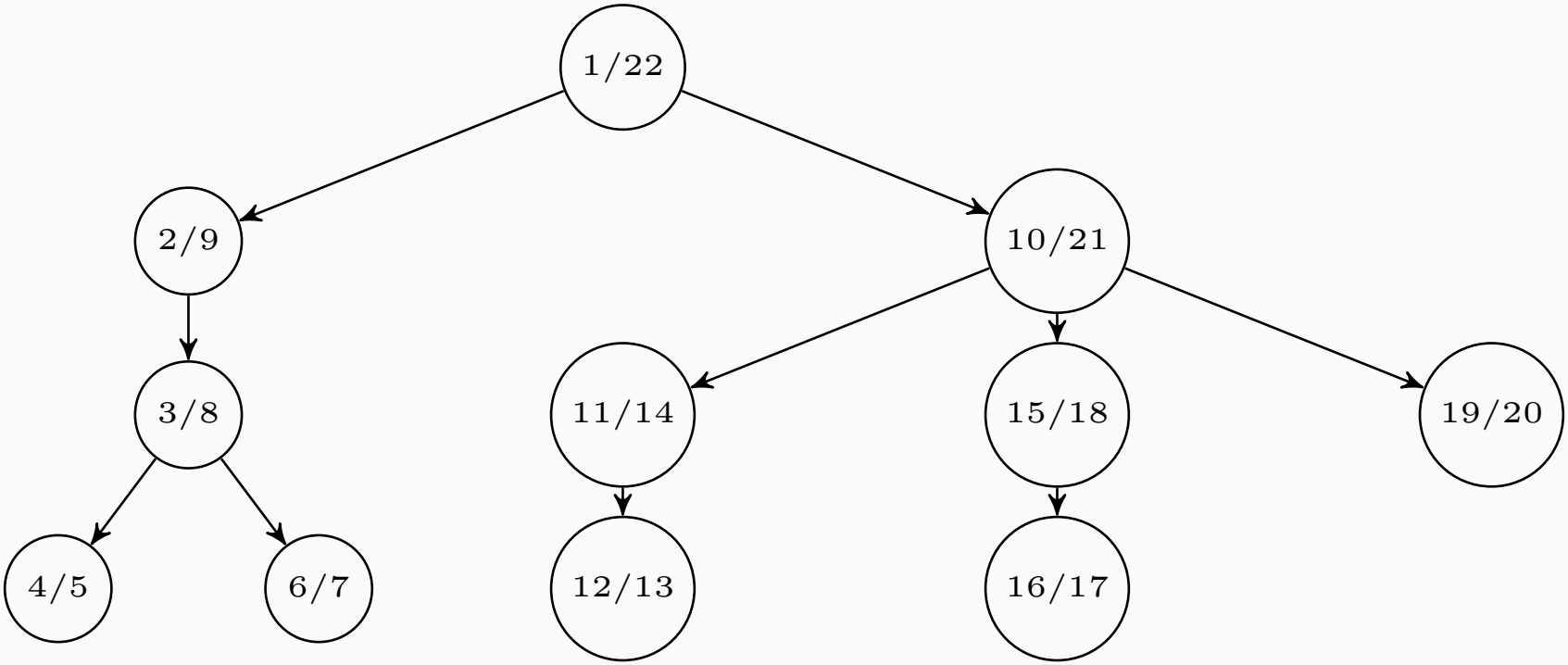
Příklad 3

Navrhněte algoritmus, který projde strom do hloubky (nepoužívejte rekurzi).



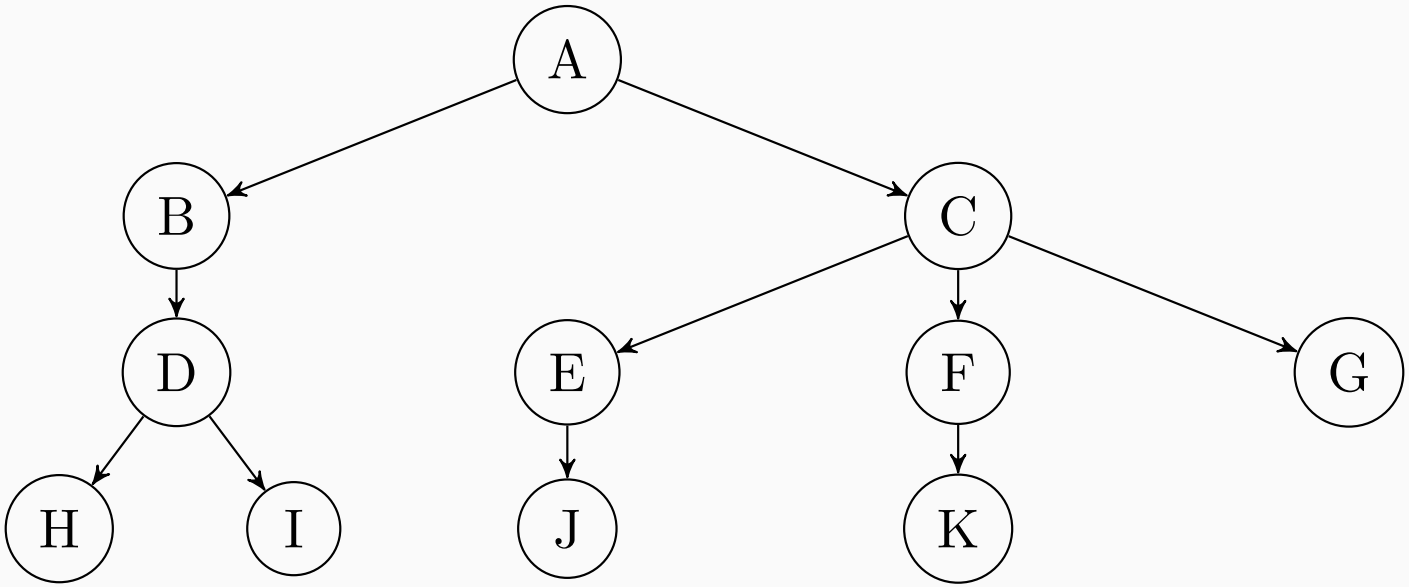
V jakém pořadí budou zpracovávány uzly následujícího stromu? Očíslujte vrcholy podle otevíracích a zavíracích časů.

Řešení 3



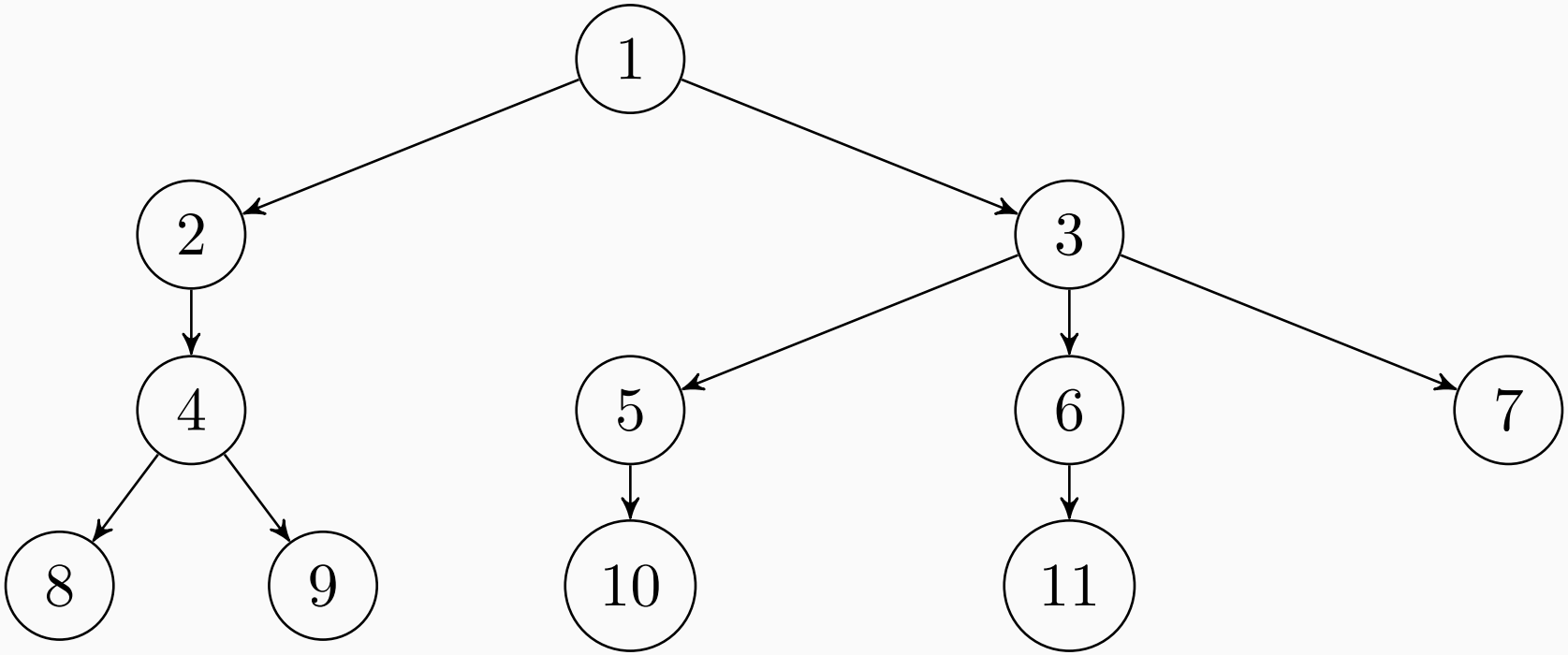
Příklad 4

Navrhněte algoritmus, který projde strom do šířky.



V jakém pořadí budou zpracovávány uzly následujícího stromu? Očíslujte vrcholy podle otevíracích časů.

Řešení 4



Příklad 5

Uvažte následující algoritmus pro průchod stromem do šířky:

1. Vlož kořen do prázdné fronty.
2. Dokud fronta není prázdná, opakuj:
   2.1 Vyjmi první prvek z fronty a zpracuj ho.
   2.2 Vlož do fronty všechny potomky právě vyjmutého listu.

Rekonstruujte strom, víte-li že průběžný obsah fronty při průchodu stromem do šířky je následující:

A

BC

C

DE

EFG

FG

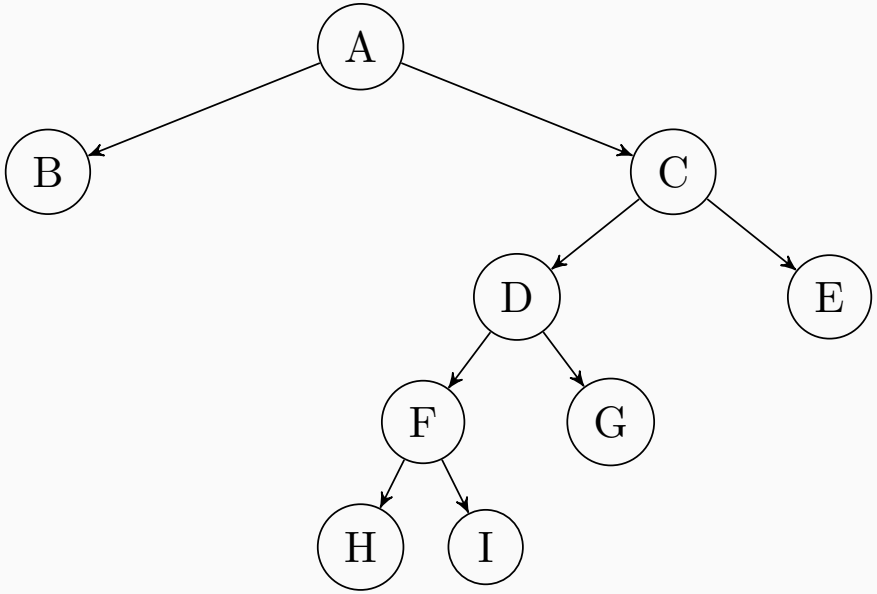
GHI

HI

I

Formulujte obecný algoritmus, který řeší tuto úlohu.

Řešení 5



V každém okamžiku je první vrchol fronty aktuálně zpracováváný klíč. Pokud se podíváme do obsahu fronty v dalším okamžiku, na konci se objeví jeho potomci. Obecný algoritmus lze formulovat takto:

1. Dokud není obsah fronty prázdný, opakuj:
   - 1.1 Vezmi vrchol v s klíčem podle první hodnoty ve frontě.
   - 1.2 Posuň se na frontu v dalším čase.
   - 1.3 Vrcholu v přidej jako potomky nové vrcholy očíslované klíči, které se objevily na konci fronty.

Příklad 6

Uvažujme, že na vstup algoritmus pro prohledávání stromu do hloubky/šířky dostaneme místo stromu obecný graf.

Navštíví algoritmus všechny vrcholy? Bude algoritmus nadále konečný? Pokud ne, jak tento problém vyřešíme?

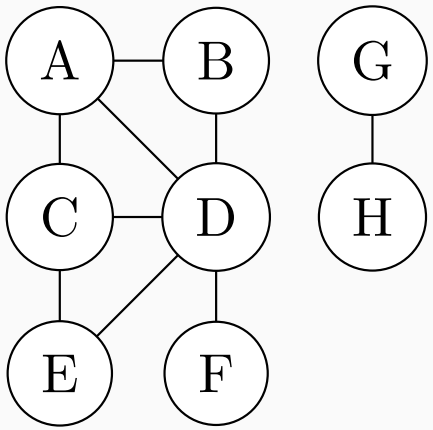
Řešení 6

V případě průchodu do hloubky algoritmus nenavštíví všechny vrcholy, pokud narazí na cyklus. Naopak prohledávání do šířky vrcholy navštíví všechny.

Ani jeden z algoritmů nebude konečný, pokud graf obsahuje cyklus. Řešením je pamatovat si, které vrcholy jsme již navštívili a které ne.

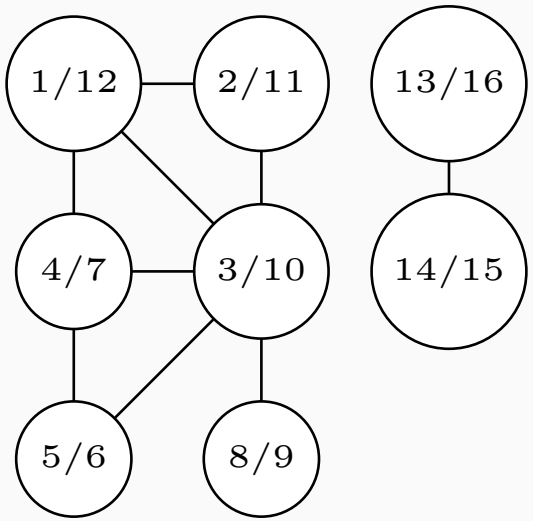
Příklad 7

V jakém pořadí budou zpracovávány uzly následujícího grafu při průchodu do hloubky? V každém okamžiku zpracovávejte následníky v abecedním pořadím.



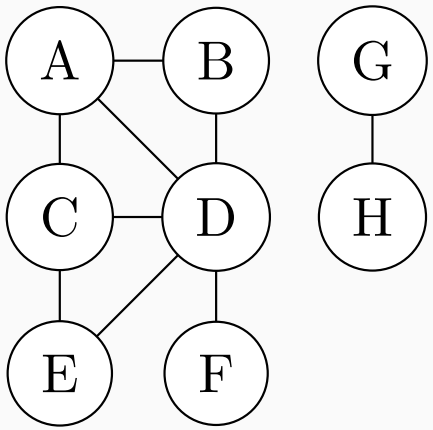
Očíslujte vrcholy podle otevíracích a zavíracích časů.

Řešení 7



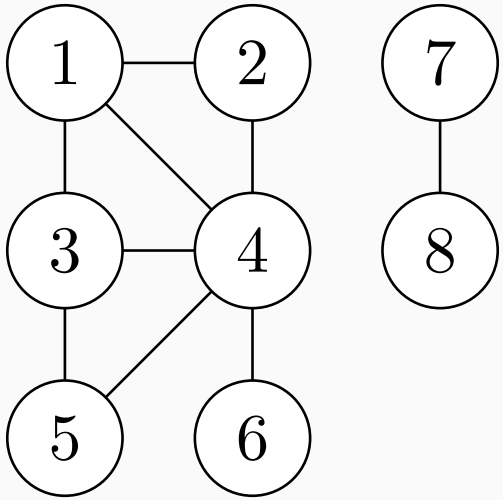
Příklad 8

V jakém pořadí budou zpracovávány uzly následujícího grafu při průchodu do šířky? V každém okamžiku zpracovávejte následníky v abecedním pořadím.



Očíslujte vrcholy podle otevíracích časů.

Řešení 8

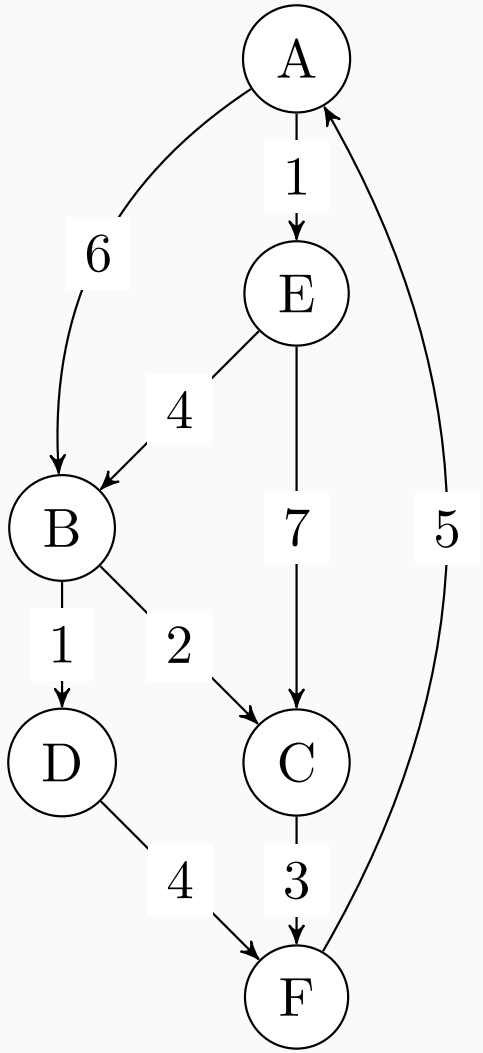


Příklad 9

Najděte cestu z vrcholu E do vrcholu A v následujícím grafu. Předpokládejte, že následníky v každém okamžiku zpracováváte v abecedním pořadí.

Zformulujte obecný algoritmus, který najde cestu z vrcholu do vrcholu. Jak naleznete cestu obsahující nejméně hran?

Bonusová otázka: Jak byste našli nejkratší cestu, pokud by hrany měly přiřazené délky?



Řešení 9

Pro nalezení cesty, která obsahuje nejméně hran lze použít prohledávání do šířky. Pokud nám pouze stačí najít nějakou cestu, lze použít prohledávání do hloubky. Cesta, která obsahuje nejméně hran je `E,C,F,A`. Nejlevnější cesta je `E,B,C,F,A` nebo `E,B,D,F,A`. Pro nalezení nejkratší cesty je třeba uzly řadit ve frontě podle toho, jaká je cena za nejkratší cesty do daného vrcholu. Více na `https://en.wikipedia.org/wiki/Dijkstra's_algorithm`.

Příklad 10

Převozník chce převézt z jednoho břehu na druhý hlávku zelí, kozu a vlka. Do loďky s sebou může vzít buď zelí, nebo kozu, nebo vlka, ale víc se tam nevejde. Nechá-li na břehu hlávku zelí a kozu, koza zelí sežere. Nechá-li na břehu kozu a vlka, pak vlk sežere kozu.

Jakým způsobem musí převozník postupovat, aby nedošlo k žádné škodě? Zkuste problém vyřešit pomocí programu.

Řešení 10

```
class StateWrapper {
    // xor of state with those is equal to application of given action to the state
    private final int MOVE_BOAT = 0b0001;           private final int MOVE_BOAT_AND_GOAT = 0b0011;
    private final int MOVE_BOAT_AND_WOLF = 0b0101; private final int MOVE_BOAT_AND_CABBAGE = 0b1001;
    // the state space, false for the original bank, true for the desired one
    // instead of boolean array we will use integer for storing the state
    protected final int state;
    protected final List<String> actions;

    public StateWrapper(int state, List<String> actions, String action) {
        this.state = state;
        this.actions = actions == null ? new ArrayList<>() : new ArrayList<>(actions);
        if (action != null) this.actions.add(action);
    }
    public boolean isFinal() {
        // 0b0000 is initial state, final is if everything is on the other side
        return state == 0b1111;
    }
    public boolean isAddmissible() {
        // admissible if there is boat on the same side as goat or goat is not together with wolf nor cabb
        final boolean goat = ((state & Main.GOAT) != 0);
        final boolean boat = ((state & Main.BOAT) != 0);
        return goat == boat || (goat != ((state & Main.CABBAGE) != 0) && goat != ((state & Main.WOLF) !=
    }
    public List<StateWrapper> listChildren() {
        List<StateWrapper> list = new ArrayList<>(4);
        list.add(new StateWrapper(state ^ MOVE_BOAT, actions, "Move boat"));
        if (((state & Main.BOAT) != 0) == ((state & Main.GOAT) != 0))
            list.add(new StateWrapper(state ^ MOVE_BOAT_AND_GOAT, actions, "Sail with goat"));
        if (((state & Main.BOAT) != 0) == ((state & Main.WOLF) != 0))
            list.add(new StateWrapper(state ^ MOVE_BOAT_AND_WOLF, actions, "Sail with wolf"));
        if (((state & Main.BOAT) != 0) == ((state & Main.CABBAGE) != 0))
            list.add(new StateWrapper(state ^ MOVE_BOAT_AND_CABBAGE, actions, "Sail with cabbage"));
        return list;
}}
```

Řešení 10

```
public class Main {
    public static final int BOAT = 0b0001; public static final int GOAT = 0b0010;
    public static final int WOLF = 0b0100; public static final int CABBAGE = 0b1000;

    public static void main(String[] args) {
        Deque<StateWrapper> queue = new LinkedList<>();
        queue.add(new StateWrapper(0x0, null, null));
        Set<Integer> visited = new HashSet<>(16);
        visited.add(0x0);

        while (!queue.isEmpty()) {
            StateWrapper state = queue.poll();
            if (state.isFinal()) {
                System.out.println(state.actions);
                return;
            }

            for (StateWrapper reachable : state.listChildren()) {
                if (reachable.isAddmissible() && !visited.contains(reachable.state)) {
                    queue.add(reachable);
                    visited.add(reachable.state);
}}}}}
```

Programovací úlohy k procvičení

`https://open.kattis.com/problems/safepassage`

Reference

`http://kam.mff.cuni.cz/~kuba/ka/pruchod.pdf`