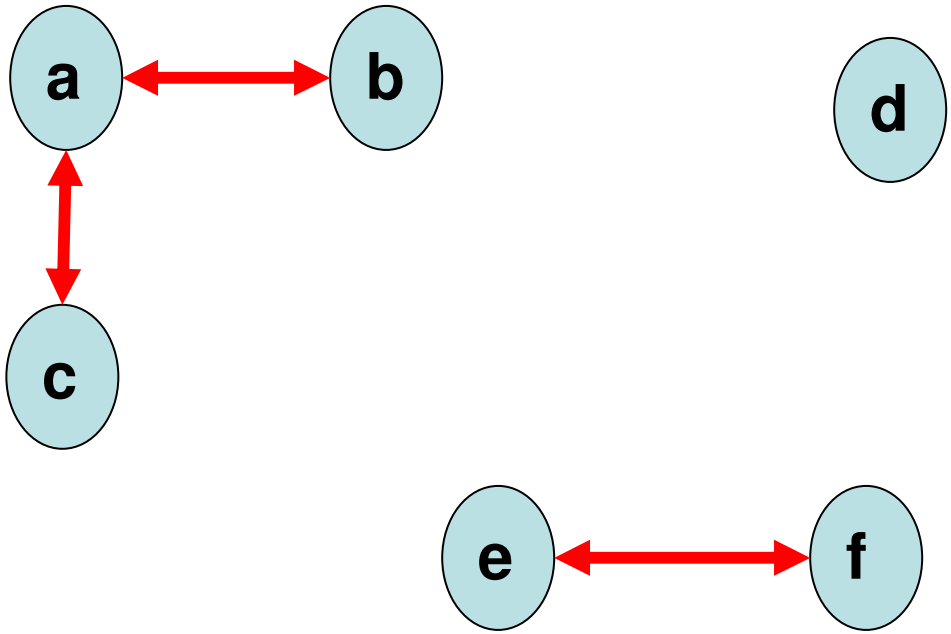



```
fk.blizkiPriatelia(a, b);
fk.blizkiPriatelia(a, c);
fk.blizkiPriatelia(e, f);
```

```
System.out.println(fk);                          // nejaká reprezentacia grafu
System.out.println(fk.vsetci());                    // nejaká permutácia "a","b","c","e","f"
System.out.println(fk.pocetPriatelov("c"));       // 1
System.out.println(fk.pocetPriatelov("a"));       // 2
System.out.println(fk.spolocniPriatelia("b", "c"));    // 1
System.out.println(fk.vzdialenyPriatel("b", "c"));     // true
System.out.println(fk.vzdialenyPriatel("a", "e"));     // false
```

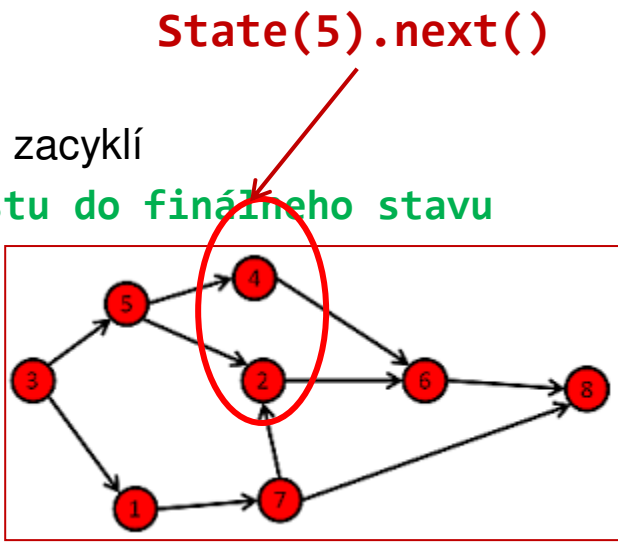
DFS/BFS/Bactracking

Ide o prehľadávanie stavového priestoru, abstrakcia pre stav môže byť:

```
interface State {
   public State();                          // počiatočný stav hľadania
   abstract boolean isFinalState();         // test na koncový stav hľadania
   abstract State[] next();                 // nasledujúce/susedné stavy
   abstract Set<State> next();              // nasledujúce/susedné stavy
   abstract boolean isCorrect();            // test na korektnosť stavu
}
```

Naivné prehľadávanie pre acyklický graf, ktoré sa na cyklickom zacyklí

```
public class Search<S extends State> // hľadáme cestu do finálneho stavu
public void searchWhichLoops(S s) {
   if (s.isFinalState())
        add(s); // pridaj do zoznamu riešení
   else
        for (State ns : s.next())
           search(ns);   // rekurzia do susedov
}
```



Aby sa to nezacyklilo

(objavila to už Ariadna pri hľadaní Thezea v labyrinte s Minotaurom)

```
public void search(S s, ArrayList<S> visited) {
    if (s.isFinalState())
        add(s); // pridaj do zoznamu riešení
    else
        for (State ns : s.next()) {
            if (!visited.contains(ns)) {// nebol si ?
                visited.add(ns); // označ
                search(ns, visited);
                visited.remove(ns); // odznač
            }
        }
}
```

BTW, je to depth-first alebo breadth-first ?

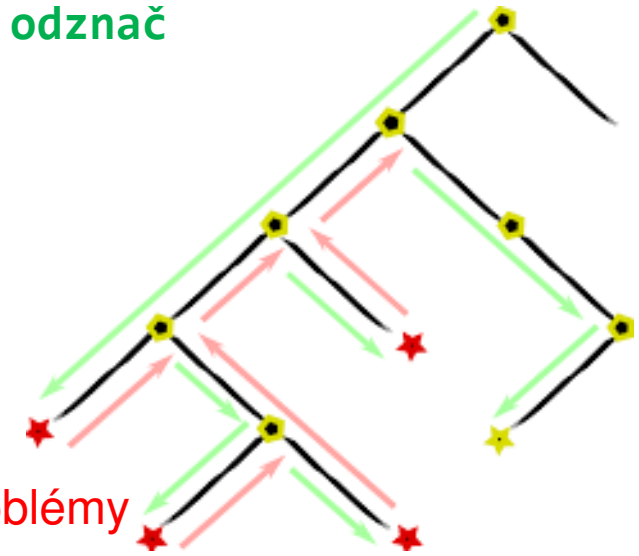
Backtracking

(orezáva podstromy určite neobsahujúce riešenie)

```
public void search(S s, ArrayList<S> visited) {
   if (s.isFinalState())
        add(s);
   else
        for (State ns : s.next()) { // môže to viesť k riešeniu ?
             if (!visited.contains(ns) && ns.isCorrect()) {
                  visited.add(ns); // označ
                  search((S) ns, visited);
                  visited.remove(ns); // odznač
             }
        }
}
```

Šikovný `isCorrect` výrazne zredukuje zväčša exponenciálny priestor stavov, ale ten aj tak zostane exponenciálny

Preto: backtrack nepoužívame na neexponenciálne problémy



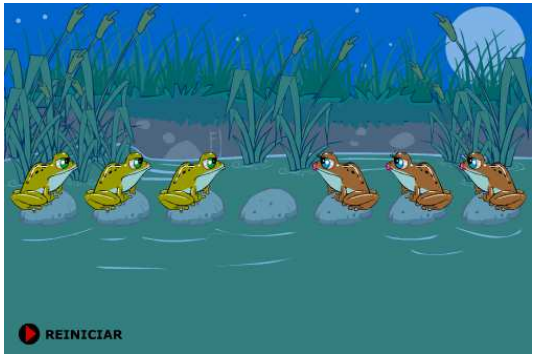
Ako by vyzeral BFS

```
private void search(ArrayList<S> queue, ArrayList<S> visited, boolean DFS) {
   while (queue.size() > 0) {
        S s = queue.remove(0);          // vyber prvý z fronty
        if (s.isFinalState())           // ak si už v cieli
              add(s);                   // pridaj do zoznamu riešení
        else
           for (State ns : s.next()) {
              if (!visited.contains(ns) && ns.isCorrect()) {
                visited.add(ns);
                if (DFS)                // ak depth-first search
                   queue.add(0, ns); // pridaj na začiatok fronty
                else                    // ak breadth-first search
                   queue.add(queue.size(), ns);// pridaj na koniec
              }
           }
   }
}
```

[>>><_<<, >>_<><<<, >_><><<<, ><>_><<, ><><>_<, ><><><_, ><><_<>, ><_<><>, _<><><>, <_><><>, <<>_><>, <<><>_>, <<><_>>, <<_<>>>, <<<_>>>]
[>>_><<<, >><>_<<, >><><_<, >><_<><, >_<><><, _><><><, <>_><><, <><>_><, <><><>_, <><><_>, <><_<>>, <_<><>>, <<_><>>, <<<>_>>, <<<_>>>]

Žabky

```
class ZabkyState implements State {
String z6;                          // šesť žiab, 3 pravé >>>, 3 ľavé <<<
public ZabkyState() {               // počiatočný stav
   z6 = ">>>_<<<";
}
public boolean isFinalState() { // koncový stav
   return z6.equals("<<<_>>>");
}
public ZabkyState[] next()  {
   ArrayList<ZabkyState> nxt = new ArrayList<ZabkyState>();
   nxt.add(new ZabkyState(z6.replace("_<", "<_"))); // ľavá cez medzeru
   nxt.add(new ZabkyState(z6.replace(">_", "_>"))); // pravá cez medzeru
   nxt.add(new ZabkyState(z6.replace("_><", "<>_"))); // ľavá cez pravú
   nxt.add(new ZabkyState(z6.replace("><_", "_<>"))); // pravá cez ľavú
   nxt.remove(this); nxt.remove(this);
   return nxt.toArray(new ZabkyState[]{});
}
```



Backtracking/Zabky.java