Zadání projektů z předmětu **Algoritmy I**
letní semestr 2019/2020
prezenční studium

Verze zadání

1. dubna 2020 První verze

Obecné pokyny

1. Celkem je k dispozici šest zadání projektů.
2. Každý student má přiděleno jedno zadání. Na webu předmětu Algoritmy I je zveřejněn seznam studentů a jim přiřazených zadání.
3. S případnými dotazy kontaktujte svého cvičícího emailem.
4. **Termín odevzdání je 10. května 2020 ve 23:59**. Tento termín je konečný a nebude dále posunován.
5. Projekt odevzdáváte jako zip archiv s řešením (solution) pro vývojové prostředí Visual Studio 2019. Jméno zip archivu bude odpovídat Vašemu loginu. Například student James Bond má login `bon007`, archiv se bude jmenovat `bon007.zip`.
6. Než projekt odešlete, vyčistěte projekt smazáním souborů vzniklých při kompilaci. Vyčištění provedete přímo v prostředí Visual Studio – menu `Build`, pak `Clean Solution`. Takto vyčištěný projekt zabalte do zip archivu.
7. Každý cvičící Vám sdělí, jakým způsobem budete projekt odevzdávat – emailem, uložením na sdílené úložiště, pomocí ftp a tak podobně.
8. Součástí zdrojových kódů Vašeho programu bude programátorská dokumentace ve formě dokumentačních komentářů, zpracovatelných programem Doxygen, viz `www.doxygen.org`. Vygenerovanou dokumentaci není nutné odevzdávat, postačuje pokud Vámi odevzdaný archiv bude obsahovat konfigurační soubor `doxyfile`, případné adresáře pro vygenerovanou dokumentaci, vkládané obrázky atd.
9. Termíny obhajoby budou vypsány v systému Edison.
10. Řešení projektu bude hodnoceno podle následujících kritérií:
    (a) správnost řešení,

(b) způsob implementace (rozložení implementace do funkcí, hlavičkových a zdrojových souborů a tak dále),

(c) způsob zápisu, čitelnost, zdrojových kódů (odsazování, pojmenování proměnných, funkcí, tříd a tak dále) a

(d) efektivita implementovaného algoritmu[1].

Správnost řešení je podmínka nezbytná, aplikace, která nebude poskytovat správné výsledky bude hodnocena automaticky 0 body.

11. **Použité zdroje je nutné správně citovat.** K řešení můžete používat odbornou literaturu, učebnice, příklady zdrojových kódů z ostatních předmětů, internetové zdroje. V tom případě je nutné uvést, že „Tento algoritmus jsem převzal z...“, „Tuto část kódu jsem převzal z...“. Tyto případné citace umístíte do dokumentačních komentářů.

[1] Smyslem tohoto kritéria není nutit Vás k implementaci nejlepšího známého algoritmu. Tímto kritériem si ponecháváme prostor pro případné snížení bodového hodnocení za použití algoritmu zcela nevhodného, nesmyslného, zmateného.

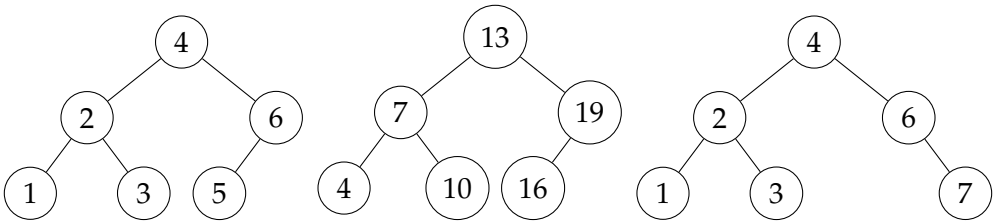
1 Struktura binárních stromů

1.1 Problém

Máte dány dva binární vyhledávací stromy T_x a T_y. Implementujte funkci či metodu třídy, která bude vracet `true` pokud mají dané dva stromy shodnou strukturu, jinak bude vracet `false`. Struktura stromu je nezávislá na datech ve stromu uložených, jde pouze o existenci či neexistenci uzlů a hran mezi nimi.

1.2 Ukázka

Uvažujme stromy na obrázku 1. Shodnou strukturu se stromem T_1 má strom T_2, přestože obsahují různá data. Stromy T_1 a T_3 nemají shodnou strukturu – uzel obsahující číslo 6 má ve stromu T_1 pouze levého potomka, ve stromu T_3 má naopak pouze pravého potomka. Pochopitelně, každý strom má shodnou strukturu sám se sebou.



(a) Strom T_1 (b) Strom T_2 (c) Strom T_3

Obrázek 1: Ukázky stromů

1.3 Implementace

- Stromy budou obsahovat pouze celá čísla typu `int`.
- Stromy budou zadány ve formě textového souboru jako posloupnost čísel, na každém řádku bude pouze jedno číslo. Každý strom je uložen v samostatném souboru.
- Stromy ze souborů získáte tak, že čísla načítaná postupně z textového souboru budete vkládat, pomocí obvyklého algoritmu, do binárního vyhledávacího stromu (základní nevyvážená varianta).

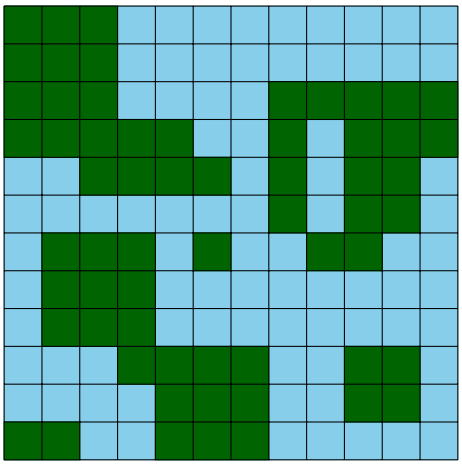
2 Mapa světa

2.1 Problém

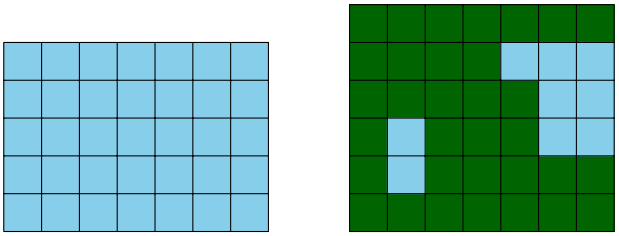
Máte zadánu mapu světa, která je tvořena čtvercovou sítí, kde zelené čtverce reprezentují pevninu a modré, jak jinak, moře. Ukázková mapa světa je zobrazena na obrázku 2. Souvislá oblast pevninských čtverců, které spolu sousedí aspoň jednou hranu tvoří *kontinent*. Implementujte funkci, či metodu třídy, která na dané mapě světa spočítá všechny kontinenty.

2.2 Ukázka

Mapa světa na obrázku 2 obsahuje celkem 6 kontinentů. Ale může existovat i mapa světa bez jediného kontinentu, obrázek 3a, nebo mapa světa tvořená jediným kontinentem, obrázek 3b.



Obrázek 2: Mapa světa



(a) Vodní svět (b) Jeden kontinent

Obrázek 3: Další ukázkové mapy

2.3 Implementace

- Mapa světa je uložena v textovém souboru v následujícím formátu:
  - na prvním řádku je uveden počet řádků mapy světa r,
  - na druhém řádku je uveden počet sloupců mapy světa c a
  - na dalších r řádcích je uveden řetězec tvořený c jedničkami nebo nulami, kde jednička odpovídá pevnině a nula moři.

  Mapa světa z obrázku 2 bude uložena takto:

```
12
12
111000000000
111000000000
111000011111
111110010111
001111010110
000000010110
011101001100
011100000000
011100000000
000111100110
000011100110
110011100000
```

- Vámi implementovanou funkci otestuje na všech třech zde uvedených ukázkových mapách.

3 Tranzitivní uzávěr

3.1 Teorie

Nejprve trochu teorie. Mějme dvě množiny A a B. Kartézským součinem $A \times B$ množin A a B rozumíme množinu všech uspořádaných dvojic (a, b) takových, že $a \in A$ a zároveň $b \in B$. Například kartézským součinem osmi prvkové množiny

$$A = \{sedma, osma, devitka, desitka, spodek, svrsek, kral, eso\}$$

se čtyř prvkovou množinou

$$B = \{srdce, listy, kule, zaludy\}$$

je třiceti dvou prvková množina

$$A \times B = \{(sedma, srdce), \dots, (sedma, zaludy), (osma, srdce), \dots, (eso, kule), (eso, zaludy)\}.$$

Pokud je množina A shodná s množinou B, pak kartézský součin $A \times A$ značíme A^2 a mluvíme o kartézské mocnině.

Dále si zavedeme pojem binární relace. Opět mějme množiny A a B. Binární relací R pak nazýváme podmnožinu kartézského součinu $A \times B$, formálně $R \subseteq A \times B$. Binární relace vyjadřuje vztah (relaci) prvků jedné množiny k prvkům v množině druhé. Fakt, že prvek $a \in A$ je v relaci R s prvkem $b \in B$ zapisujeme obvykle jako aRb.

Binární relace mají různé vlastnosti. Jednou z těchto vlastností je tranzitivita relace. Relaci $R \subseteq A^2$ nazýváme tranzitivní, právě když platí

$$aRb \wedge bRc \Rightarrow aRc$$

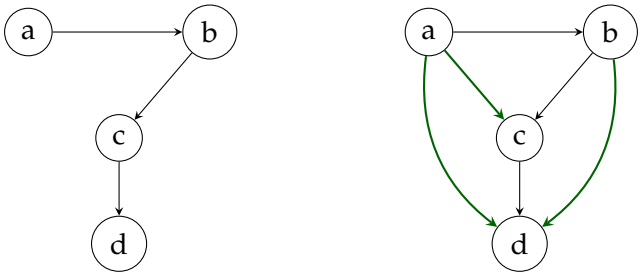
pro všechny prvky $a, b, c \in A$. Například relace „být větší“, značená běžně jako $<$, na kartézské mocnině přirozených čísel $\mathbb{N}^2$, je tranzitivní, protože zřejmě platí $a < b \wedge b < c \Rightarrow a < c$.

Na obrázku 4a je graficky znázorněna relace $K \subseteq M^2$ nad množinou $M = \{a, b, c, d\}$. Písmena můžou například označovat počáteční písmena jmen Adam, Božena, Cyril a David. Binární relaci K, znázorněnou orientovanými hranami, můžeme slovně zapsat jak vztah „být kamarád“. Z obrázku jasně vidíme například, že Adam je kamarád Boženy, Božena je kamarádkou Cyrila. Z téhož obrázku je dále patrné, že Adam nemusí být nutně kamarádem Cyrila[2]. Neplatí tedy obecně, že $aKb \wedge bKc \Rightarrow aKc$.

Binární relace K tedy není tranzitivní. Zajímavou úlohou může být snaha doplnit relaci K tak, aby se stala tranzitivní a zároveň obsahovala nejmenší možný počet vztahů[3]. Výsledná relace $T(K)$ je zobrazena na obrázku 4b, nově přidané vztahy jsou znázorněny zelenými šipkami. Takto vytvořená relace $T(K)$ se nazývá *tranzitivní uzávěr* (angl. transitive closure) relace K.

[2]Jak je v reálném světě obvyklé, Adam nemusí znát všechny Boženčiny kamarády.

[3]Je jasné, že pokud bychom doplnili relaci K na celý kartézský součin, tak relace K bude tranzitivní. Což pro naši ukázkovou relaci znamená, že „všichni se kamarádí se všemi“. Takové řešení však není asi zajímavé



(a) binární relace K

(b) tranzitivní uzávěr $T(K)$

Obrázek 4: Příklad binární relace a jejího tranzitivního uzávěru

3.2 Problém

Ve Vašem projektu předpokládejme binární relaci $R \subseteq A^2$, kde A je množina přirozených čísel $A = \{0, 1, \ldots, n-1\}$. Vaším úkolem je:

1. navrhnout a implementovat vhodné datové struktury pro reprezentaci a manipulaci s binární relací R a
2. navrhnout a implementovat algoritmus výpočtu tranzitivního uzávěru pro zadanou binární relaci R.

3.3 Implementace

Relace R je uložena v textovém souboru následujícím způsobem:

- na prvním řádku je uložena hodnota n a
- na dalších řádcích je vždy uložena dvojice čísel a a b, které náleží do relace R, tedy platí pro ně aRb.

Binární relace z obrázku 4a bude uložena takto[4]:

```
4
0 1
1 2
2 3
```

Výsledný tranzitivní uzávěr $T(R)$ vypište do textového souboru ve stejném formátu.

[4] Pokud označíme a jako 0, b jako 1...

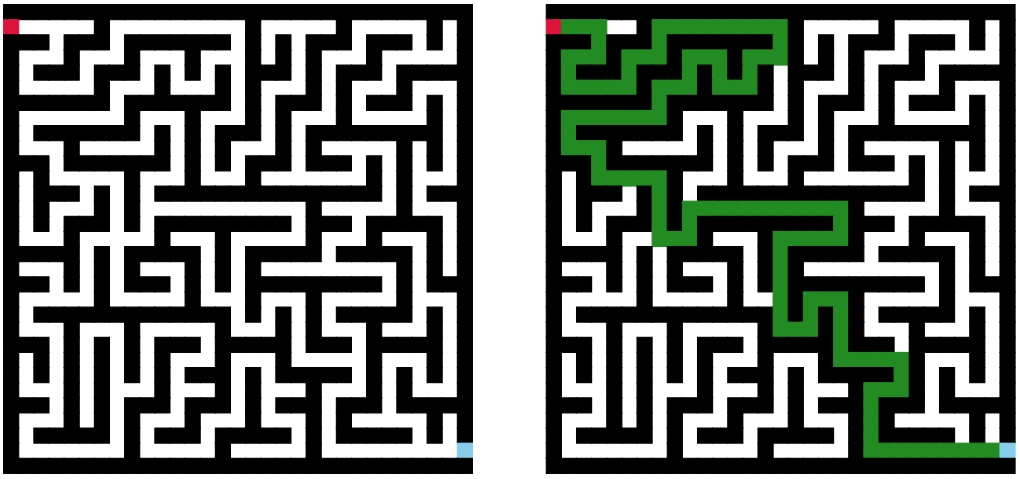
4 Nalezení cesty bludištěm

4.1 Problém

Předpokládejme, že máte zadáno bludiště. Do bludiště jsou z vnějšku dva vstupy, jeden budeme považovat za vchod a druhý za východ. Implementujte funkci, metodu třídy, která nalezne cestu bludištěm od vchodu k východu.

4.2 Ukázka

Ukázkové bludiště je uvedeno na obrázku 5a. Bludiště je sestaveno ve čtvercové síti, černé čtverečky představují zdi bludiště, bílé jsou cestičky, červený čtvereček představuje vchod do bludiště a modrý čtvereček označuje východ z bludiště. Nalezená cesta, vyznačena zeleně, v ukázkovém bludišti je zobrazena na obrázku 5b.



(a) ukázkové bludiště (b) cesta v bludišti

Obrázek 5: Hledání cesty v bludišti

4.3 Implementace

Bludiště je, jak již bylo řečeno, vytvořeno ve čtvercové síti a je uloženo v textovém souboru v následujícím formátu:

- na prvním řádku je uveden počet řádků čtvercové sítě bludiště r,
- na druhém řádku je uveden počet sloupců čtvercové sítě bludiště c a
- na dalších r řádcích je uveden řetězec tvořený c číslicemi 0, 1, 2 nebo 3, kde

0 odpovídá cestičce,
1 odpovídá zdi,
2 odpovídá vchodu do bludiště a
3 odpovídá východu z bludiště.

Bludiště z obrázku 5a bude uloženo takto:

```
31
31
11111111111111111111111111111111
20000010000000001000000100000101
11101110111111101010111011110101
10001000100010001010100010000001
10111011101010101110101110111111
10000010001000101000101000100001
11111110111111101011101011110101
10000000100010001010001000000101
10111111101010111010111111111101
10001000001010100010100000010101
11101111111010101110111010101101
10100000100010001000000001010001
10111010101111111111111111010111
10100010100000000000010000010001
10101110101111111111011101111101
10001000001000100000100010001101
11111010111111010111110111110101
10001010100010101000000000010101
11101010111010101111111111010111
10000010000010001000100010000010001
10111111111111110101010111110101
10001000100000100010100010001000101
11101010101110111111011101111101
10101010101000100010000001000101
10101010101011101011111010101010
10001010001000101000100010101000 1
11101010111011101011101110111111
10001010100010001000100101000000101
10111110101110111111010111110101
10000000100000100000100000000003
11111111111111111111111111111111
```

Řešení problému, to jest bludiště s nalezenou cestu, vypište ve stejném formátu jako vstup. Nalezenou cestu označte číslicí 4.

5 Symetrie binárního stromu

5.1 Problém

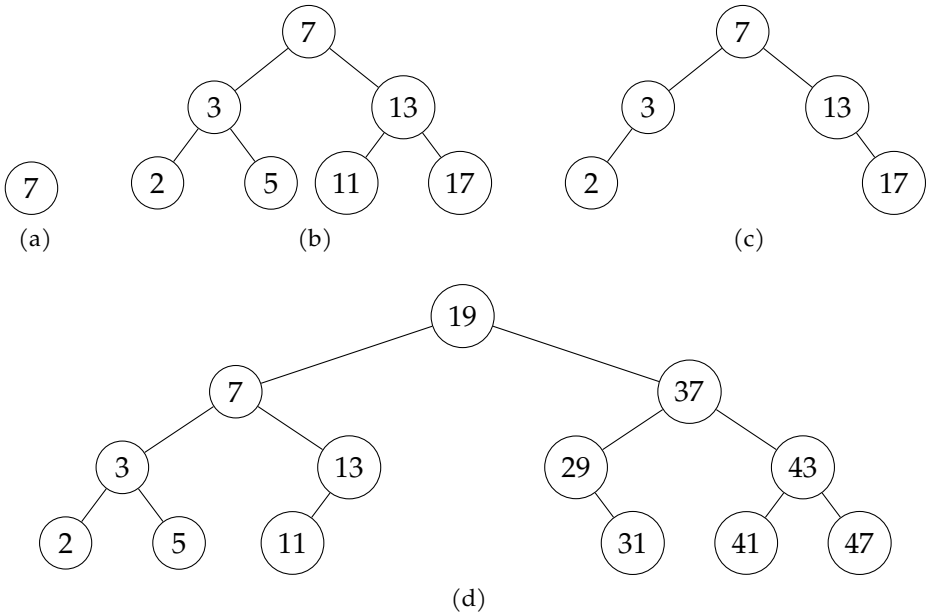
Máte dán binární vyhledávací strom T. Implementujte funkci, či metodu třídy, která bude vracet `true`, pokud je binární strom T symetrický, jinak bude vracet `false`. Symetrie binárního stromu je nezávislá na datech ve stromu uložených, jde pouze o existenci či neexistenci uzlů a hran mezi nimi. Binární strom je symetrický, pokud je sám sobě zrcadlovým obrazem.

5.2 Ukázka

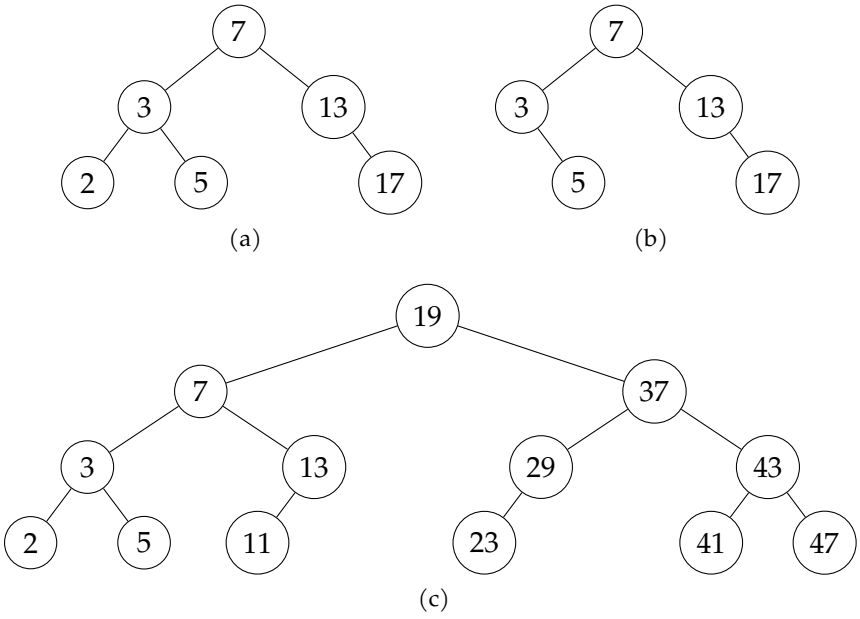
Na obrázku 6 je zobrazeno několik ukázkových symetrických binárních stromů, včetně binárního stromu s jedním uzlem, viz obrázek 6a. Na tomto místě je vhodné upozornit na fakt, že prázdný binární strom je také považován za symetrický. Na obrázku 7 pak můžeme vidět několik nesymetrických binárních stromů.

5.3 Implementace

- Stromy budou obsahovat pouze celá čísla typu `int`.
- Stromy budou zadány ve formě textového souboru jako posloupnost čísel, na každém řádku bude pouze jedno číslo. Každý strom je uložen v samostatném souboru.
- Stromy ze souborů získáte tak, že čísla načítaná postupně z textového souboru budete vkládat, pomocí obvyklého algoritmu, do binárního vyhledávacího stromu (základní nevyvážená varianta).



Obrázek 6: Symetrické binární stromy



Obrázek 7: Nesymetrické binární stromy

6 Iterátor nad grafem

6.1 Problém

V tomto zadání máte dán neorientovaný graf, vrcholy obsahují hodnoty typu `int`. Pokud potřebujeme provést s každým vrcholem grafu nějakou akci, využijeme typicky některý z algoritmů průchodu grafem. Průchod grafem lze implementovat jako jednu funkci, či metodu, která projde graf a s každých vrcholem provede zadanou akci. Jakmile je taková funkce jednou zavolána a zahájí svou činnost, bývá velmi obtížně ji nějakým způsobem pozastavit nebo pro specifický vrchol provést nějakou odlišnou akci.

Druhou možností, jak takový průchod grafem provést je použít *iterátor*. Iterátor nad grafem si můžeme představit jako objekt, který implementuje metody pro postupný průchod všemi vrcholy grafu. Na rozdíl od průchodu grafem implementované v jediné funkci, je činnost iterátoru rozdělena do několika funkcí a je plně řízena aplikačním programátorem. Typické využití iterátoru je následující:

```
Graph G("MyGraph.txt"); // vytvoreni grafu
GraphIterator Iter(G); // vytvoreni iteratoru pro graf G
for(
     Iter.Reset(); // inicializace, reset, iteratoru
     !Iter.IsEnd(); // dotaz na konec pruchodu
     Iter.Next()  // posun na dalsi vrchol
  )
{
  cout << Iter.CurrentKey(); // zpracovani vrcholu
}
```

Jak je asi z ukázky patrné, iterátor umožňuje transparentním způsobem projít všechny vrcholy grafu pomocí jednoduchého cyklu a odstiňuje aplikačního programátora od implementace grafu.

Vaším úkolem je implementovat dva iterátory procházející všechny vrcholy v tomto grafu. První iterátor bude procházet graf do šířky, druhý do hloubky.

6.1.1 Iterátor průchodem do hloubky

Iterátor `DFSIterator` bude pro svou práci využívat zásobník, kdy na vrcholu zásobníku je uložen aktuální vrchol grafu. Pod ním jsou v zásobníku všechny dosud objevené vrcholy čekající na zpracování. Dále je nutné aby si iterátor nějakým způsobem, v nějaké datové struktuře, evidoval, který vrchol již byl či nebyl zpracován.

Iterátor bude implementovat tyto metody:

- konstruktor `DFSIterator` – zde se iterátoru předá graf se kterým bude iterátor pracovat.
- `Reset` – metoda uloží do zásobníku vrchol grafu s nejnižším číslem. Iterace začíná tímto vrcholem.

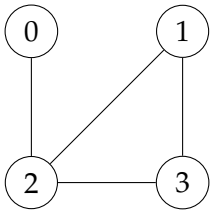
- `Next` – funkce posune aktuální vrchol iterátoru na následující vrchol. Metoda otestuje zásobník, jestli je prázdný. Dále postupuje takto:
  1. Pokud není prázdný, vyjme aktuální vrchol ze zásobníku a do zásobníku uloží všechny neobjevené sousedy právě vyjmutého vrcholu. Pokud takový soused neexistuje, neukládá se do zásobníku nic.
  2. Pokud je zásobník prázdný, pokusí se najít pomocí evidence vrcholů další nezpracovaný vrchol a uloží jej na zásobník. Iterátor přešel do další souvislé komponenty grafu. Pokud takový vrchol neexistuje, neukládá do zásobníku nic.
- `IsEnd` – vrací `true`, pokud je iterace ukončena, jinak vrací `false`. Iterace je ukončena, pokud je zásobník prázdný a všechny vrcholy byly označeny jako zpracované.
- `CurrentKey` – vrací data z vrcholu grafu, který je na vrcholu zásobníku. Pozor, na hodnotu v aktuálním vrcholu se můžu dotazovat opakovaně, nelze proto vrchol ze zásobníku v této funkci odstranit.

6.1.2 Iterátor průchodem do šířky

Iterátor `BFSIterator` bude pro svou práci využívat frontu, kdy na začátku fronty je uložen aktuální vrchol. Za ním jsou ve frontě všechny dosud objevené vrcholy čekající na zpracování. Dále je nutné aby si iterátor nějakým způsobem, v nějaké datové struktuře, evidoval, který vrchol již byl či nebyl zpracován.

Iterátor bude implementovat tyto metody:

- konstruktor `BFSIterator` – zde se iterátoru předá graf se kterým bude iterátor pracovat.
- `Reset` – metoda uloží do zásobníku vrchol grafu s nejnižším číslem. Iterace začíná tímto vrcholem.
- `Next` – funkce posune aktuální vrchol iterátoru na následující vrchol. Metoda otestuje frontu, jestli je prázdná. Dále postupuje takto:
  1. Pokud není prázdná, vyjme aktuální vrchol z fronty a do fronty uloží všechny neobjevené sousedy právě vyjmutého vrcholu. Pokud takový soused neexistuje, neukládá se do fronty nic.
  2. Pokud je fronta prázdná, pokusí se najít pomocí evidence vrcholů další nezpracovaný vrchol a uloží jej do fronty. Iterátor přešel do další souvislé komponenty grafu. Pokud takový vrchol neexistuje, neukládá do fronty nic.
- `IsEnd` – bude vracet `true` pokud je iterace ukončena, jinak vrací `false`. Iterace je ukončena, pokud je fronta prázdná a všechny vrcholy byly označeny jako zpracované.



Obrázek 8: Neorientovaný graf

- `CurrentKey` – vrací hodnotu v aktuálním vrcholu, čili ve vrcholu, který je v hlavě fronty. Pozor, na hodnotu v aktuálním vrcholu se můžu dotazovat opakovaně, nelze proto vrchol z fronty v této funkci odstranit.

6.2 Implementace

- Jak bylo řečeno v úvodu zadání, každý vrchol grafu obsahuje, je označen, celým číslem i, kde $0 \leq i < n$. Graf bude uložen v textovém souboru v následujícím formátu. Na prvním řádku bude číslo n. Na každém řádku bude uložena dvojice čísel celých čísel i a j oddělených mezerou. Čísla i a j představují vrcholy grafu mezi kterými existuje hrana. Ukázkový graf z obrázku 8 bude uložen takto:

```
4
0 2
3 1
2 1
2 3
```

- Vámi implementované iterátory musí pracovat správně i s **nesouvislými** grafy!
- V tomto projektu je nutné implementovat neorientovaný graf. Implementace může být velice jednoduchá a postačuje jen implementace metod nezbytných pro činnost iterátorů – nic víc není nutné implementovat.