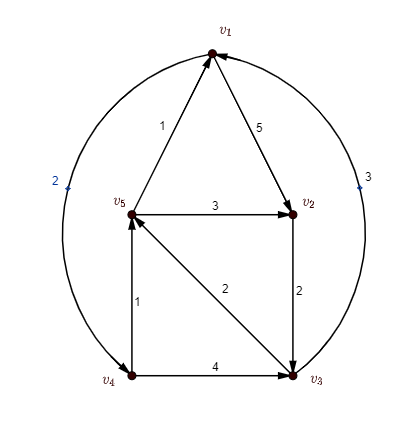
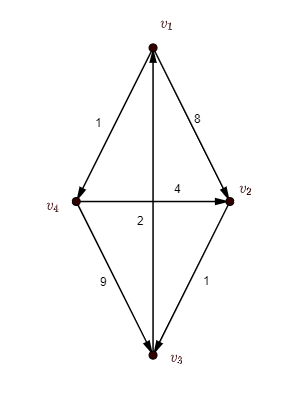
# Grafové algoritmy

## Floyd-Warshalow algoritmus

najkratšia cesta v ohodnotenom grafe)



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Floyd-warshallov alg. | | | | | | | |
| najkratšia cesta v ohodnotenom grafe | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 | 0 |  |  |  |  |  | nekonečno - dáme všade, kde nie je ohodnotená hrana |
| 2 |  | 0 |  |  |  |  | diagonály budú mať 0 |
| 3 |  |  | 0 |  |  |  | ak sa dá dastať z bodu A do bodu B zapíšeme hodnotu ohodnotenej hrany |
| 4 |  |  |  | 0 |  |  |  |
| 5 |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 | 0 |  |  |  |  |  |  |
| 2 |  | 0 |  |  |  |  |  |
| 3 |  |  | 0 |  |  |  | sčitujem žlté bunky napr. B14+ C12, pýtam sa B14+C12 < nekonečno/hodnota v C14, ak áno zapíšem novú hodnotu |
| 4 |  |  |  | 0 |  |  |  |
| 5 |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 | 0 |  |  |  |  |  |  |
| 2 |  | 0 |  |  |  |  |  |
| 3 |  |  | 0 |  |  |  |  |
| 4 |  |  |  | 0 |  |  |  |
| 5 |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 | 0 |  |  |  |  |  |  |
| 2 |  | 0 |  |  |  |  |  |
| 3 |  |  | 0 |  |  |  |  |
| 4 |  |  |  | 0 |  |  |  |
| 5 |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 | 0 |  |  |  |  |  |  |
| 2 |  | 0 |  |  |  |  |  |
| 3 |  |  | 0 |  |  |  |  |
| 4 |  |  |  | 0 |  |  |  |
| 5 |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 | 0 |  |  |  |  |  |  |
| 2 |  | 0 |  |  |  |  |  |
| 3 |  |  | 0 |  |  |  |  |
| 4 |  |  |  | 0 |  |  |  |
| 5 |  |  |  |  | 0 |  |  |



for i = 1 to N

for j = 1 to N

if there is an edge from i to j

dist[0][i][j] = the length of the edge from i to j

else

dist[0][i][j] = INFINITY

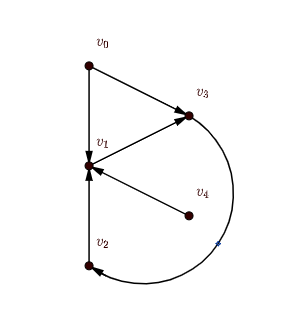
for k = 1 to N

for i = 1 to N

for j = 1 to N

dist[k][i][j] = min(dist[k-1][i][j], dist[k-1][i][k] + dist[k-1][k][j])

## Roy-Warshallow algoritmus



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Roy-warshallov alg. | | | | | | | |
| najkratšia cesta v ohodnotenom grafe | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 |  | |  | | --- | |  | |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  | ak sa dá dastať z bodu A do bodu B zapíšeme X |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  | podobne ako Flyod-Warshallov alg., len: |
| 1 |  |  |  |  |  |  | X \* nekonecno = nekonecno |
| 2 |  |  |  |  |  |  | X \* X = X |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Na nájdenie uzáveru grafu.

*void warshall(int g[n][n]) {*

*for (int x = 0; x<n; x++) {*

*for (int y = 0; y < n; y++) {*

*if (g[y][z] == 0) {*

*g[y][z]=g[y][x] \* g[x][z];*

*}*

*}*

*}*

## Prehľadávanie grafu

Na prechádzanie vrcholmi grafu a zisťovanie jeho rôznych vlastností sa používajú dva základné algoritmy:

·      prehľadávanie do hĺbky (depth-first search, DFS)

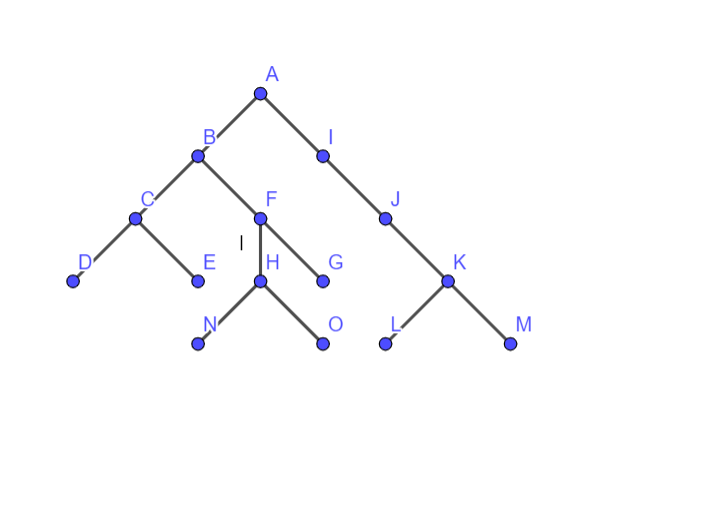
·      prehľadávanie do šírky (breadth-first search, BFS )

### Prehľadávanie do hĺbky (DFS)

Pri prehľadávaní grafu začíname v zadanom vrchole. Prehľadávanie do hĺbky navštívi prvého suseda (ak ho ešte nenavštívilo) a potom rekurzívne navštívi svojho prvého nenavštíveného suseda. Keď príde k vrcholu, ktorý už navštívil, rekurzia sa vynorí a navštevuje ďalších nenavštívených susedov. Preto hovoríme o prehľadávaní do hĺbky, lebo sa prioritne vnára do hlbšej rekurzie. Aby sa program nezacyklil, musí si pamätať už navštívené vrcholy (alebo si ich nejako označiť - napríklad zafarbiť).

Treba si uvedomiť, že takto zabezpečíme, že prehľadávanie do hĺbky navštívi vrchol práve raz. Keď program skončí, v zozname navštívených vrcholov máme len tie, ktoré tvoria jeden komponent. Týmto algoritmom môžeme zistiť počet vrcholov v jednom komponente grafu. Tiež ho môžeme využiť na zistenie počtu komponentov v grafe.

**Testovanie dosiahnutia vrcholov, acyklickosť, bludisko.**



### Prehľadávanie do šírky (BFS)

Prehľadávanie do šírky bude využívať rad. Rad poznáme z bežného života, keď napríklad čakáme v rade na obed alebo v rade na pokladňu. V podstate rad na spracovanie prvkov bude zoznam, kde najnovší prvok, čakajúci na spracovanie, pridáme na koniec zoznamu. Ako prvý obslúžime vždy prvok, ktorý je na začiatku radu, čiže na začiatku zoznamu. Odtiaľ ho vyberieme metódou pop(0).

Prehľadávanie do šírky bude fungovať tak, že do radu zaradíme prvý vrchol (z ktorého začíname). Potom pomocou while cyklu budeme dovtedy, kým nebude rad prázdny, robiť nasledovné:

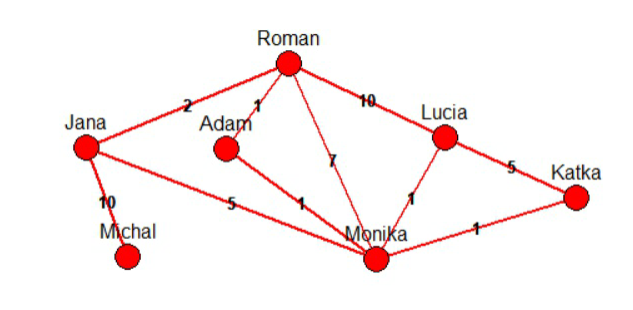
- vyberieme prvého čakajúceho,

- ak nie je ešte medzi navštívenými vrcholmi, zaznačíme si ho, že sme ho práve navštívili, a zafarbíme ho,

- zistíme všetkých jeho susedov, a ak nie sú medzi navštívenými vrcholmi, zaradíme ich na koniec radu, aby počkali na spracovanie.

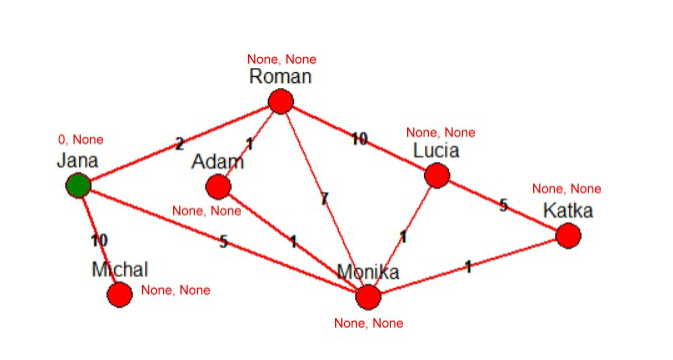
BFS – hľadanie mostov grafov, artikulácia, testovanie planarity grafov.

## Hľadanie najkratšej cesty

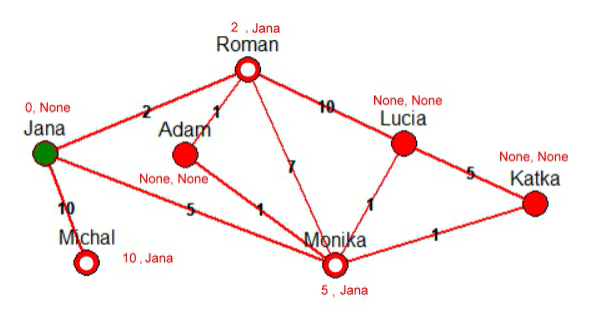


Najprv si na obrázku ukážeme Dijkstrov algoritmus. Chceme nájsť najkratšiu cestu (ohodnotenú) z vrcholu Jana do vrcholu Katka. Napríklad Jana s Katkou si chcú vymeniť prostredníctvom svojich kamarátov nejakú informáciu za čo najkratší čas. Pre **každý vrchol si nastavíme vzdialenosť na** **None** (ešte ju nemáme definovanú) a **jeho predchodcu na None** (zatiaľ žiadny). **Všetky vrcholy** si zaradíme do **zoznamu nespracovane. Začiatočnému vrcholu nastavíme vzdialenosť (cesty) na nulu**.

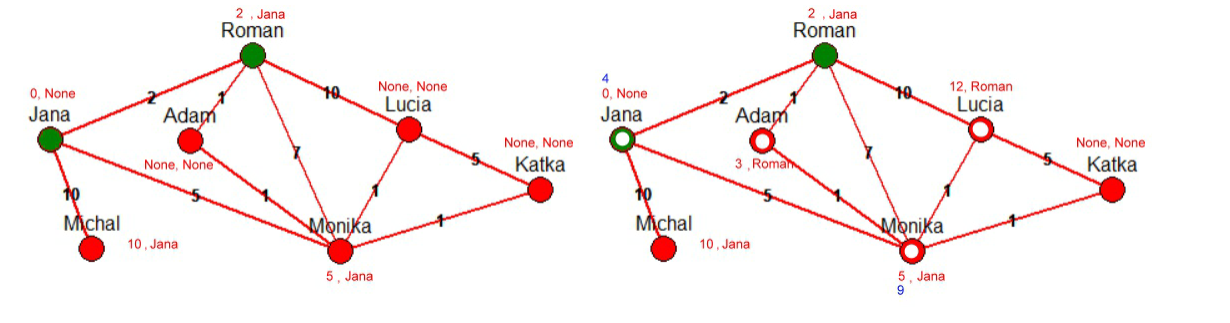
Z nespracovaných vrcholov vyberieme najlepší (taký, ktorý ma zatiaľ vypočítanú **najmenšiu vzdialenosť**). Na začiatku je to Jana so vzdialenosťou 0 (samej od seba, lebo je začiatok). Najlepší vrchol odstránime zo zoznamu nespracovane. **Nespracované sú červené a spracované sú zelené**.



Teraz prejdeme všetkých susedov Jany a pre každého suseda si vypočítame novú vzdialenosť. Ak je menšia ako vzdialenosť doposiaľ, zapamätáme si aj predchodcu (vrchol, z ktorého sme prišli a našli najmenšiu vzdialenosť).Susedov sme si označili bielym bodom vo vrchole. Pri názve vrcholu vidíme aj terajšiu vzdialenosť a predchodcu.

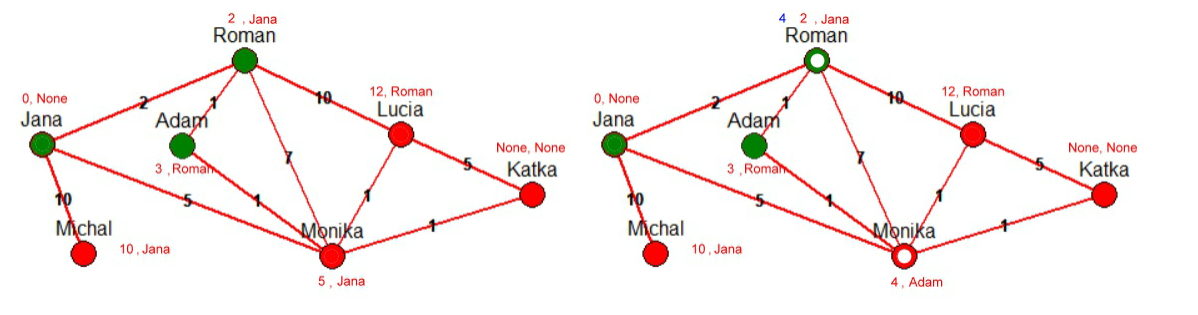


Z nespracovaných vrcholov (červených) opäť vyberieme vrchol s najmenšou vzdialenosťou. V tomto prípade je to Roman (vzdialenosť = 2). Romana odstránime z nespracovaných vrcholov (označíme si ho zelenou farbou) a ideme vypočítať vzdialenosti pre všetkých jeho susedov.

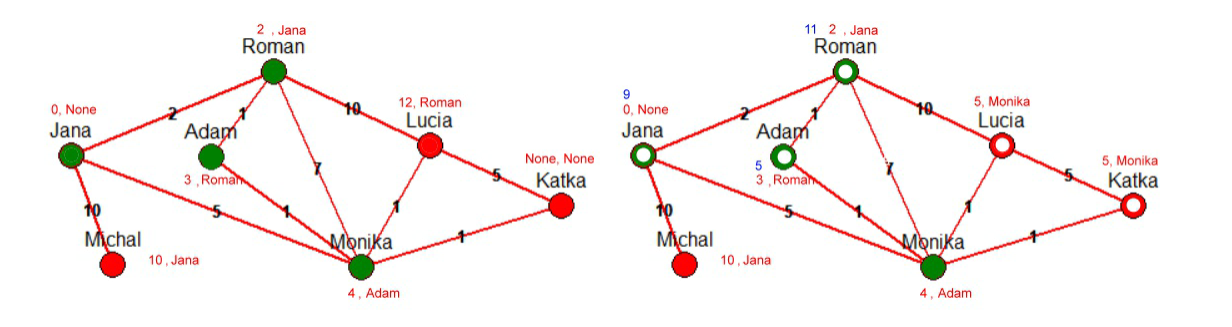


Susedom Romana (označený bielym bodom vo vrchole) sme vypočítali nové vzdialenosti. Napríklad nová vzdialenosť pre Moniku by bola 9, lebo k Romanovi sme sa dostali so vzdialenosťou 2 a váha hrany Roman Monika je 7 (7 + 2 = 9). Lenže k Monike sme sa dostali už kratšou cestou so vzdialenosťou 5 od Moniky, takže pôvodné informácie nezmeníme a stále si u Moniky budeme pamätať doterajšie (najlepšie). Zlepšili sa u Adama a Lucie, takže u nich si ich aj zapamätáme (aj s predchodcom (odkiaľ sme sa k nim dostali)).

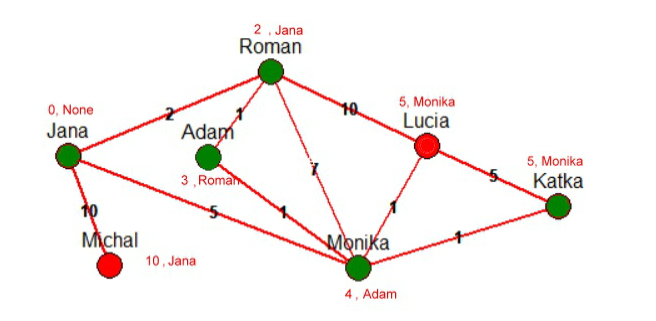
Opäť z nezaradených vyberieme najmenšie, čiže Adam – 3. Označíme ho za spracovaného a ideme vypočítať nové vzdialenosti pre všetkých jeho susedov:



Lepšiu vzdialenosť sme získali pre Moniku (4), takže si ju zapamätáme aj s informáciou predchodcu (Adam). U Romana sa zhoršila, tak si ju nezmeníme. Opäť vyberieme najlepšieho z nezaradených, čo je Monika, a označíme ju za spracovanú. Následne vypočítame vzdialenosti pre všetkých susedov Moniky.



Vidíme, že po vypočítaní nových vzdialeností susedov Moniky sa zlepšila vzdialenosť u Katky a Lucie, tak si ich zapamätáme aj s informáciou o predchodcovi. Tým sme sa dostali ku Katke a môžeme skončiť.



Algoritmus skončí, keď sú všetky vrcholy spracované. Môžeme skončiť aj skôr, stačí, že sme spracovali vrchol,ktorý je cieľom cesty. Teraz môžeme spätným chodom zistiť postupne všetkých predchodcov Katky a máme najkratšiu cestu v ohodnotenom grafe. Ideme po predchodcoch dovtedy, kým neprídeme k vrcholu, ktorého predchodcom je None. Keď túto postupnosť otočíme, máme celú najkratšiu cestu v správnom poradí.