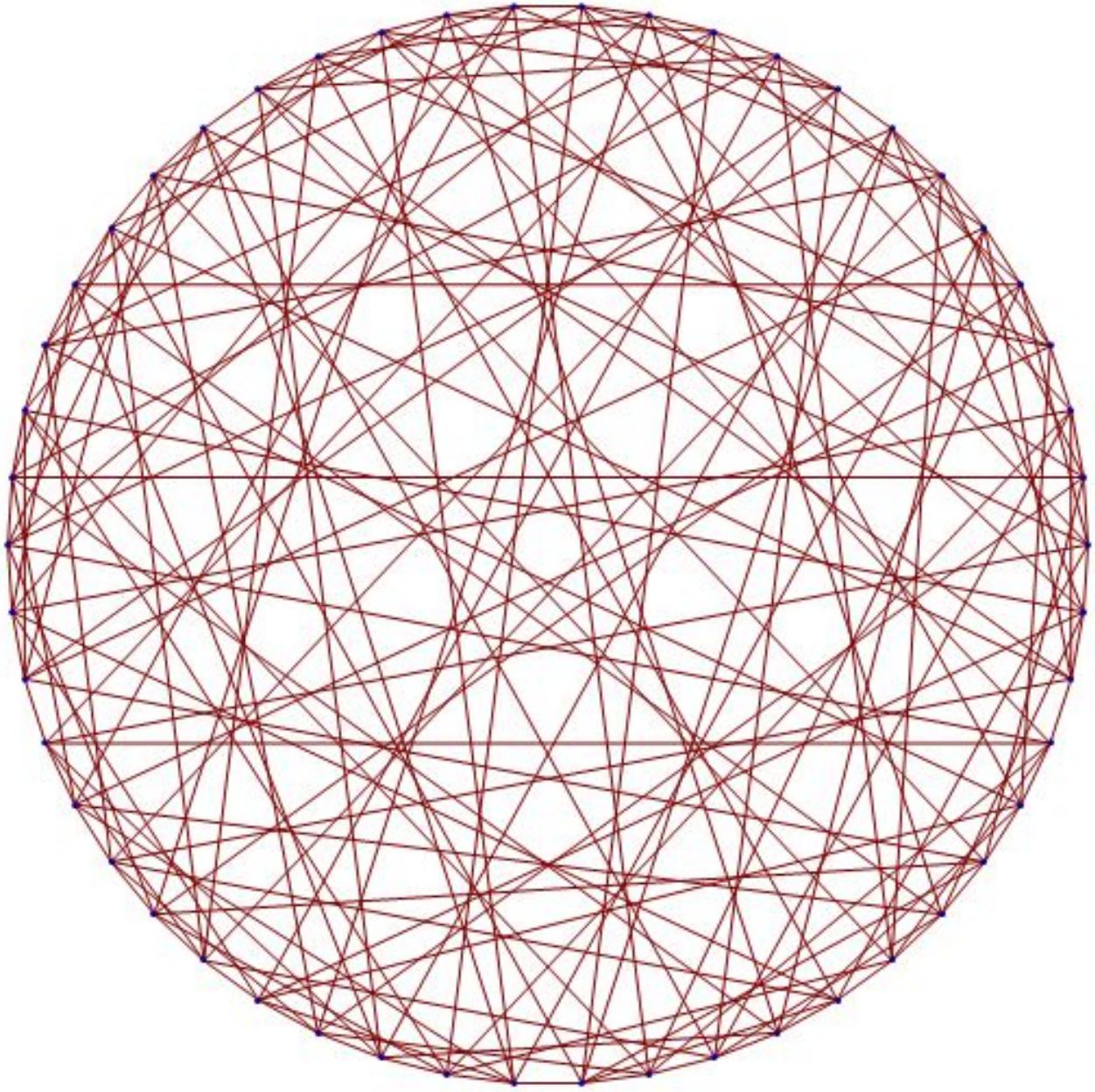
Prehľadávacie algoritmy eulerovské grafy, hamiltonovské kružnice

Algebra a diskrétna matematika

Prednáška č. 6

doc. RNDr. Jana Šiagiová, PhD.

Hoffman-Singletonov graf



Prehľadávanie grafov

Často v grafoch chceme zistiť určitú vlastnosť (súvislosť, priemer, obvod).

Zo zoznamu susedov alebo matice susednosti to nie je ľahké hneď zistiť.

Mnohé takéto úlohy si vyžadujú preskúmanie grafu, kedy postupne navštevujeme všetky vrcholy a hrany grafu.

Prehľadávanie by malo byť efektívne a systematické.

Hlavná myšlienka prehľadávania je postupné označovanie vrcholov

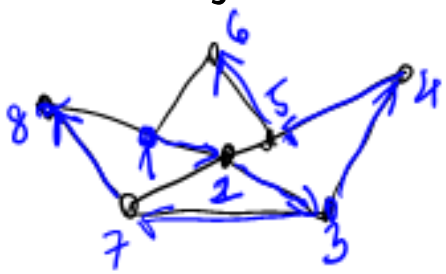
- **neobjavený** - všetky vrcholy na začiatku
- **objavený** - vrchol sme objavili, ale ešte nepreskúmali
- **preskúmaný** - preskúmali sme vrchol s celým jeho okolím

Podľa spôsobu prehľadávanie poznáme dve najviac používané metódy:

- Prehľadávanie grafu do **hĺbky** (depth-first search DFS)
- Prehľadávanie grafu do **šírky** (breadth-first search BFS)

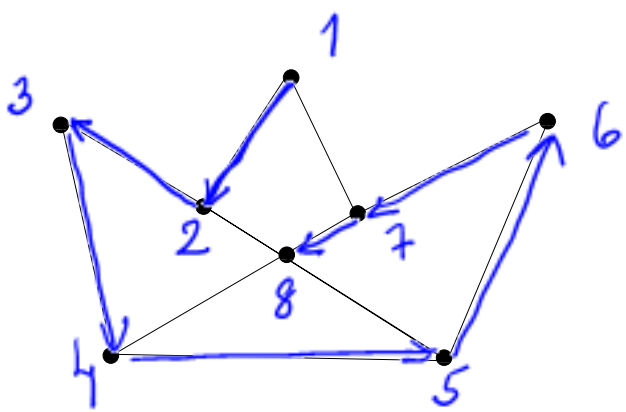
Prehľadávanie grafu do hĺbky (depth-first search)

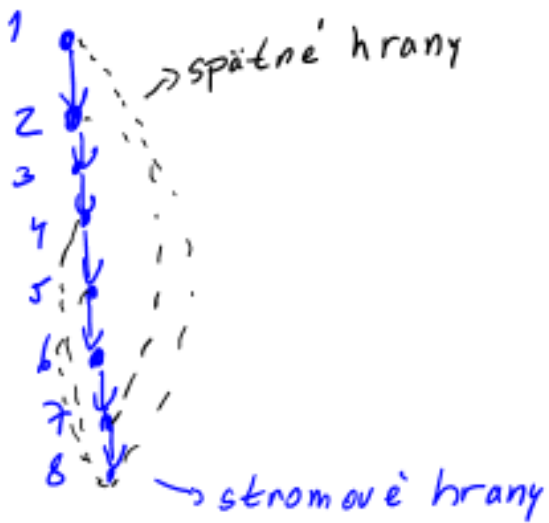
Hlavná idea: postup cez susedov vrcholov tak hlboko, ako je možné.



- začneme vo zvolenom vrchole
- objavíme jedného jeho suseda
- potom suseda tohto suseda, atď
- ak sa ďalej nedá pokračovať a existujú ešte neobjavené vrcholy, tak postupujúc naspäť nájdeme prvý vrchol s ešte neobjaveným susedom a opakujeme postup, až kým nepreskúmame všetky vrcholy

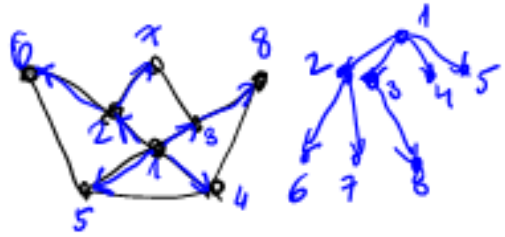
Vrcholom priraďujeme čísla v poradí, ako ich objavujeme.
Každá hrana dostane šípku označujúcu smer prvého prechodu hranou.
Hrany sú *stromové*, ak nás dovedú k novým vrcholom, inak *spätné*.





Prehľadávanie grafu do šírky (breadth-first search)

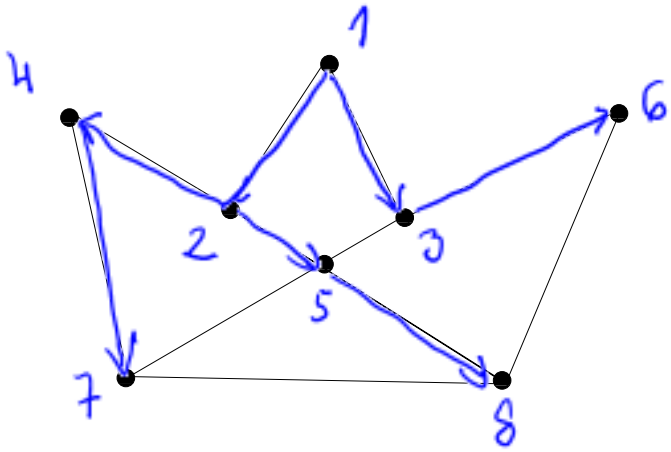
Hlavná idea: prehľadávame vždy všetkých susedov skúmaného vrchola.

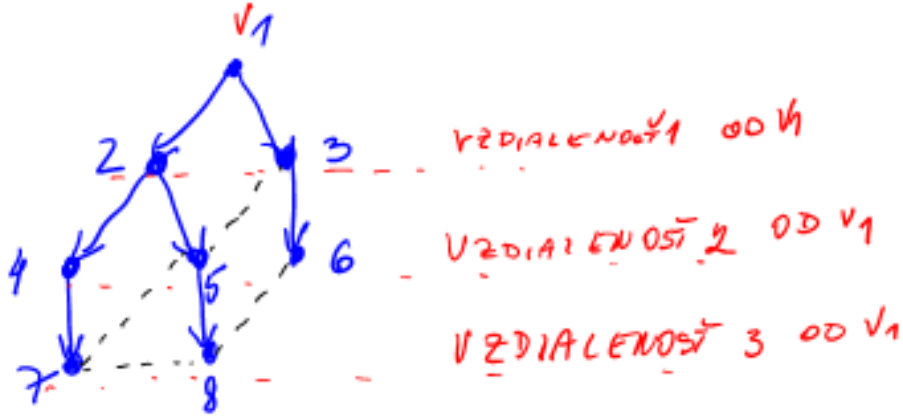


- najprv skúmame zvolený vrchol v
- potom jeho všetkých susedov
- potom všetkých susedov týchto susedov, atď
- ukončíme, keď preskúmame všetky vrcholy grafu (komponentu)

Vrcholy prechádzame podľa vzrastajúcej vzdialenosti od vrchola v. Teda algoritmus vytvorí rozklad vrcholovej množiny podľa vzdialeností od v.

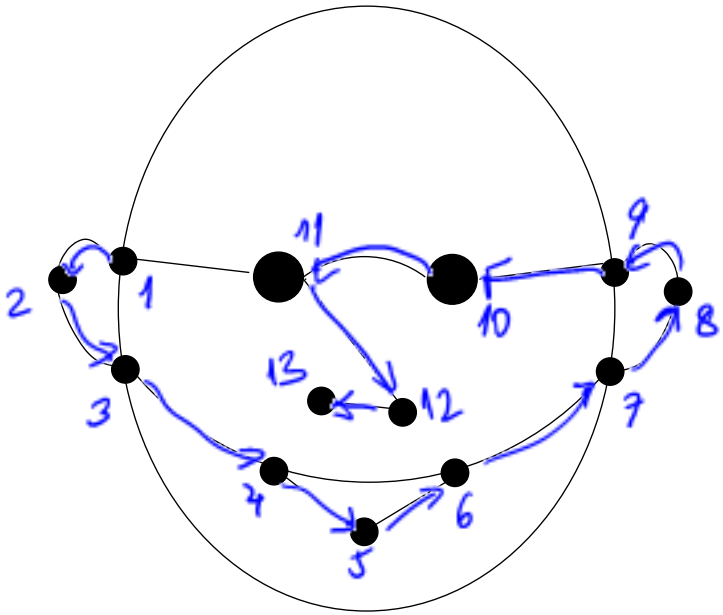
Postup do šírky však negeneruje žiaden sled.

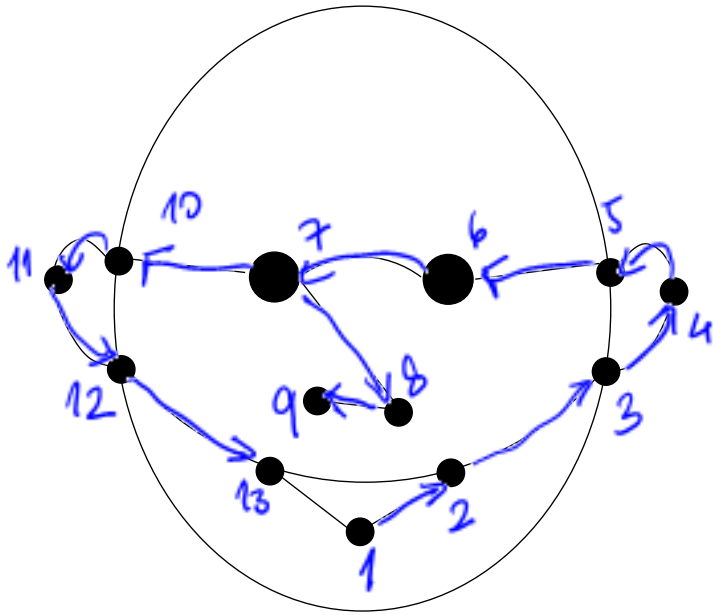


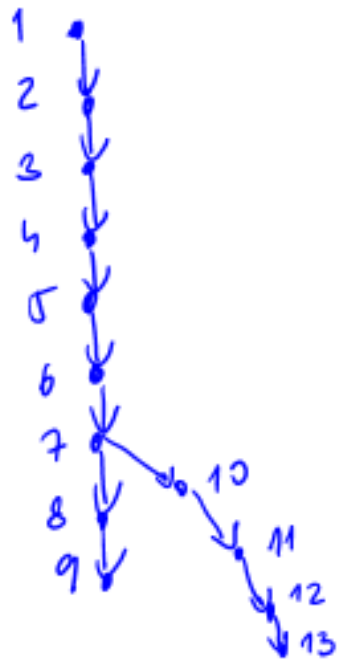


Pri obidvoch prehľadávaniach sa vytvára kostra v každom komponente súvislosti. Môže byť rozdielna.

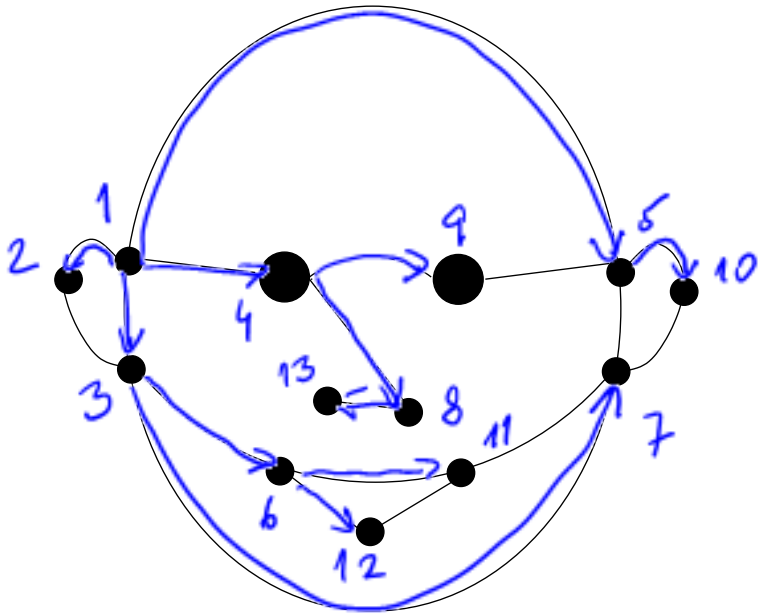
Nájdite neizomorfné kostry pomocou DFS

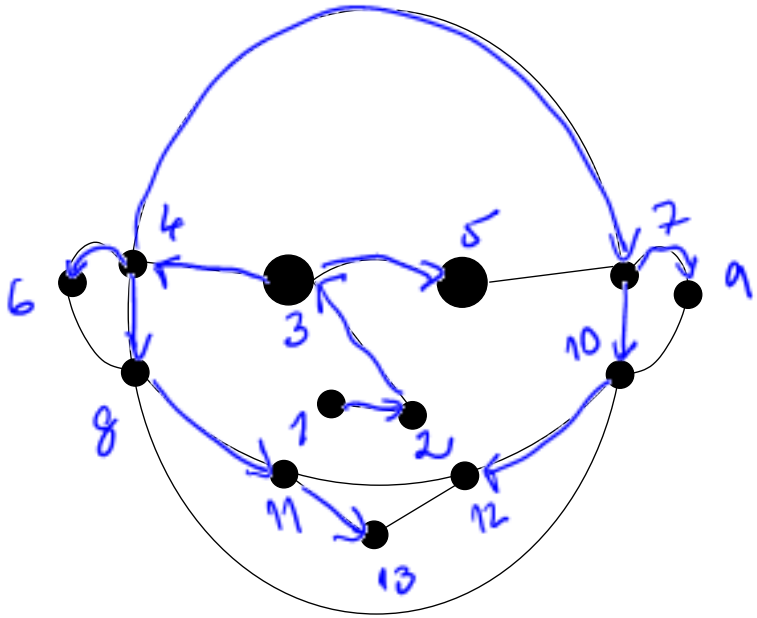


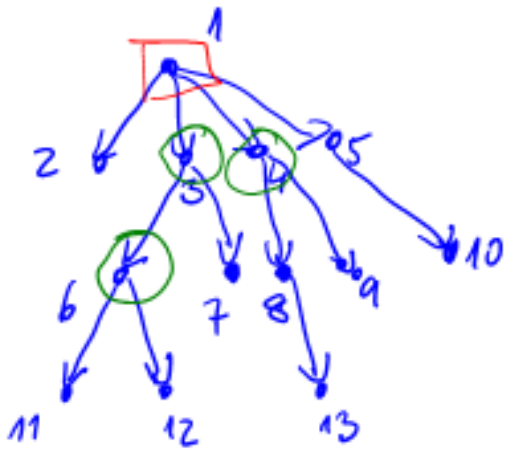


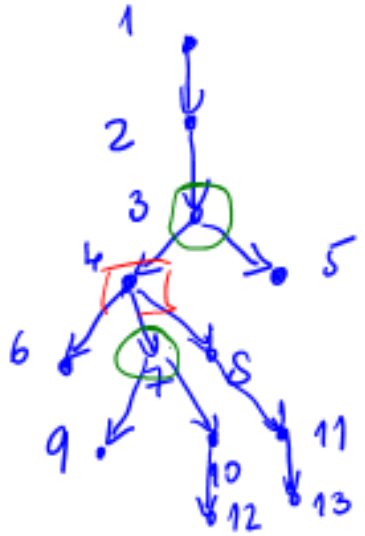


Nájdite neizomorfné kostry pomocou BFS









max stupeň 4

3 vrcholy stupňa 3

neizomorfné

max stupeň 4

2 vrcholy stupňa 3

Problém minimálnej kostry

S týmto problémom sa môžeme stretnúť napr.

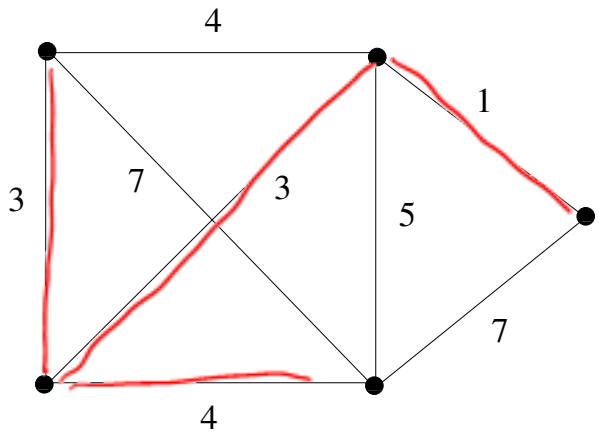
- pri navrhovaní najlacnejšej súvislej siete prenosných liniek
- pri tvorení leteckej siete, aby celková dĺžka prepojení bola čo najkratšia

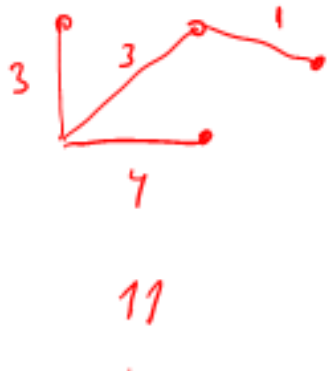
Problém minimálnej kostry

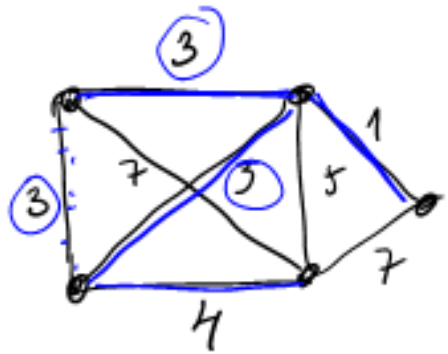
Pre súvislý graf $G = (V, E)$ s nezáporným ohodnotením hrán w nájdite kostru $T = (V, E')$ grafu G s najmenšou možnou hodnotou $w(T)$.

Prvý, ale komplikovaný algoritmus – Borůvka (1926)

Najznámejší jednoduchší algoritmus – Kruskal (1956)







Kruskalov algoritmus

Minimálnu kostru v grafe budeme konštruovať postupným pridávaním hrán s najmenšou váhou tak, aby sme nevytvorili žiaden cyklus.

Vstup: Súvislý hranovo ohodnotený graf $G = (V, E)$ s n vrcholmi

Výstup: Podmnožina hrán $A \subseteq E$ taká, že graf (V, A) je minimálna kostra grafu G.

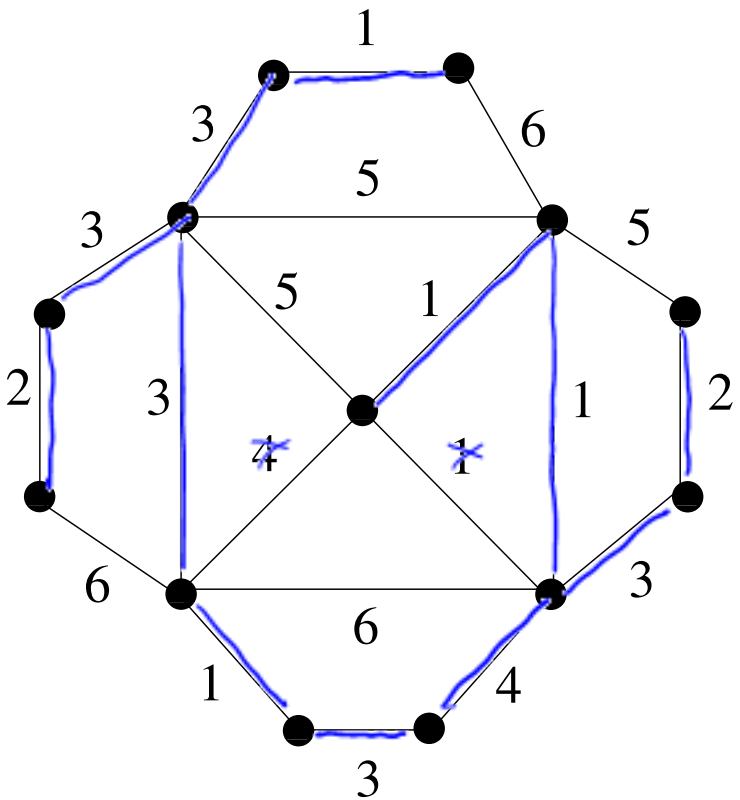
Algoritmus

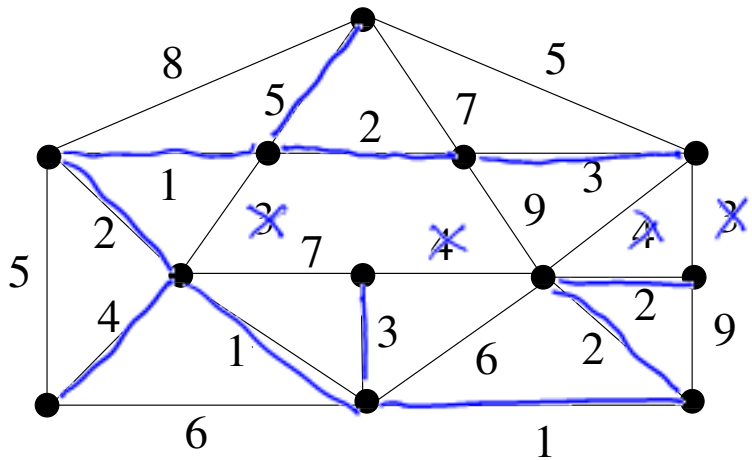
- **Krok 1:** Polož $A = \emptyset$.
- **Krok 2:** Polož $F = E$.
- **Krok 3:** Ak $F = \emptyset$ alebo je graf (V, A) strom, ukonč, inak prejdi na Krok 4.
- **Krok 4:** Vyber z množiny F hranu e s minimálnym ohodnotením (ak ich je viac, vyber ľubovoľnú z nich). Ak graf $(V, A \cup \{e\})$ neobsahuje kružnicu, pridaj hranu e do množiny A. Choď na Krok 3.

Správnosť algoritmu: Algoritmus končí grafom $T = (V, A)$ s $n - 1$ hranami, ktorý je kostrou grafu G.

Kruskalov algoritmus - Príklad

Pomocou Kruskalovho algoritmu nájdite najlacnejšie kostry.

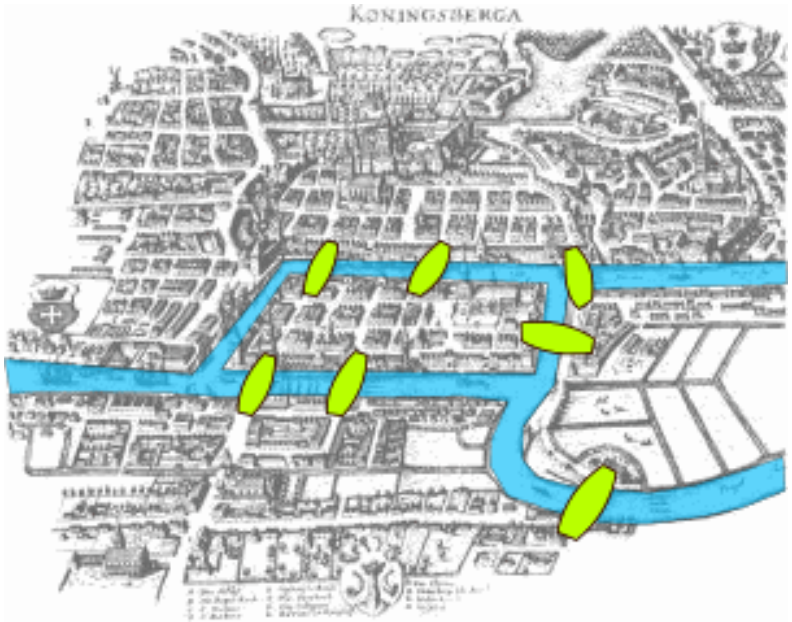


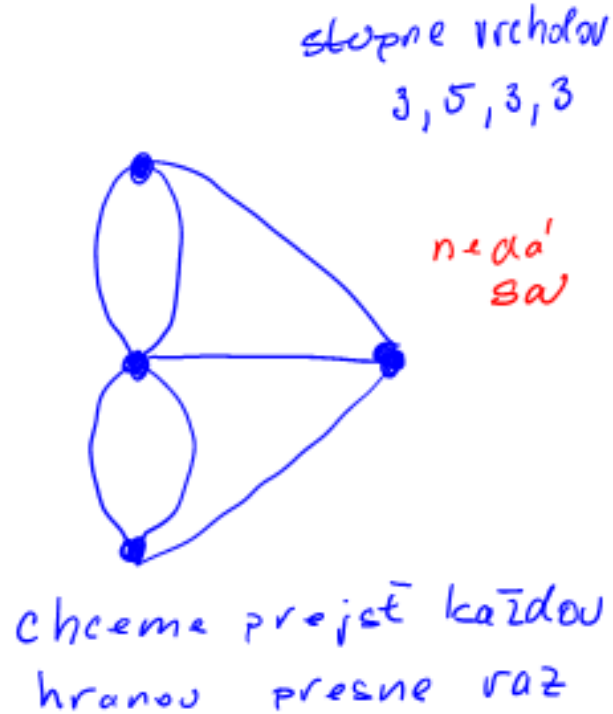


Problém 7 mostov v Královci (Königsberg, Kaliningrad)

Považuje sa za začiatok teórie grafov.
V roku 1736 ho riešil **Leonhard Euler** (1707, Švajčiarsko - 1783, Rusko).

Mesto leží na dvoch brehoch rieky, má dva ostrovy a 7 mostov.





Úloha: Zostavte cestu cez mesto, ktorá prejde každým mostom práve raz a vráti sa do východzieho miesta.

Eulerovské grafy

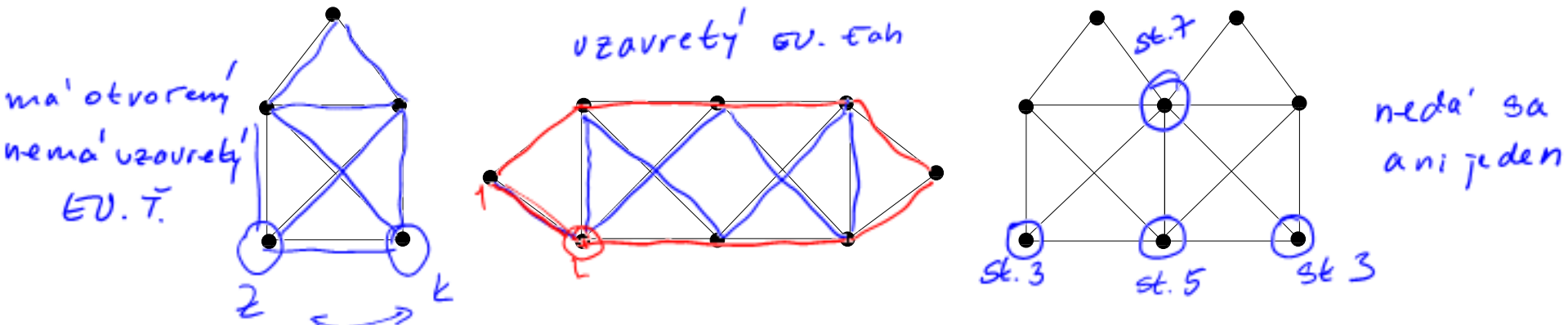
Úloha: Nakreslite daný graf jedným uzavretým ťahom bez zdvihnutia tužky z papiera, pričom žiadna hrana sa neobkreslí viackrát.

OTVORENÝ

OTVORENÝ

Uzavretý eulerovský ťah v grafe je uzavretý sled hrán a vrcholov, v ktorom sa každá hrana vyskytuje práve raz a každý vrchol aspoň raz.

Graf je **eulerovský** práve vtedy, keď má aspoň jeden uzavretý eulerovský ťah.

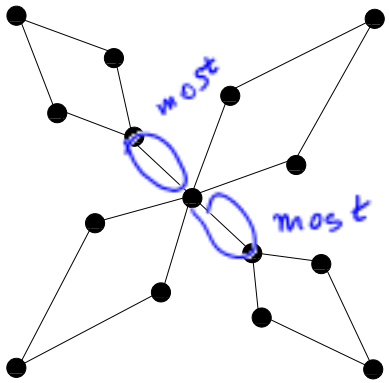


Charakterizácia eulerovských grafov: Graf je eulerovský práve vtedy, keď je súvislý a všetky jeho vrcholy sú párneho stupňa.

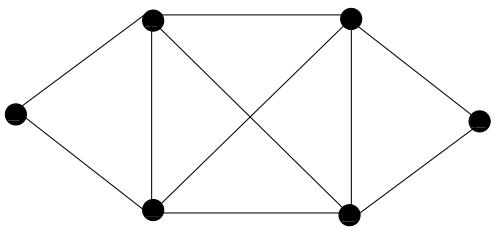
OTVORENÝ EUL. ŤAH ⇒ PRESNE 2 VRCHOLY NEPÁRNEHO STUPŇA

Most grafu

Mostom grafu $G = (V, E)$ je taká hrana $e \in E$, pre ktorú platí, že graf $G - e$ má väčší počet komponentov ako graf G.



Tvrdenie: Ak má graf všetky vrcholy párneho stupňa, tak neobsahuje most.



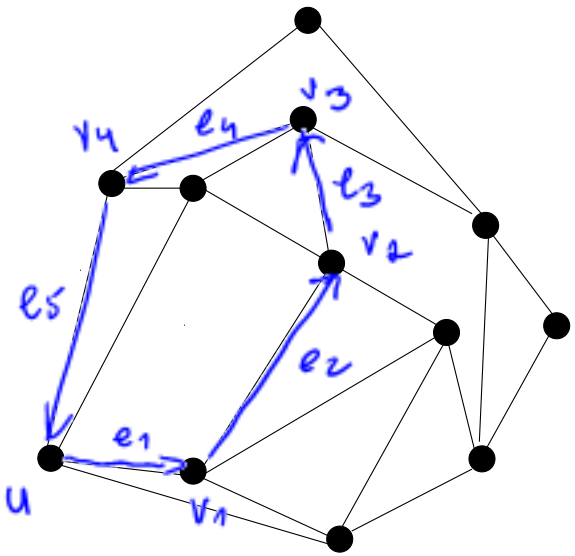
Procedúra generovania uzavretého ťahu v grafe

Vstup: eulerovský graf H, vrcholom $u \in H$

Výstup: uzavretý ťah T obsahujúci vrchol u

Procedúra - Proc(H, u):

- vyberieme ľubovoľnú hranu $e_1 = uv_1$ v H
- vytvoríme graf H_1 vynechaním hrany e_1 z grafu H ($H_1 = H - e_1$)
- vyberieme ľubovoľnú hranu $e_2 = v_1v_2$ a vytvoríme $H_2 = H_1 - e_2$
- iterujeme, kým sa nevrátime do vrchola u – v i-tom kroku z vrchola v_{i-1} vyberieme hranu $e_i = v_{i-1}u$
- zostrojili sme uzavretý ťah $ue_1v_1e_2v_2 \dots v_{i-1}e_iu = T$



Algoritmus vygenerovania eulerovského ťahu

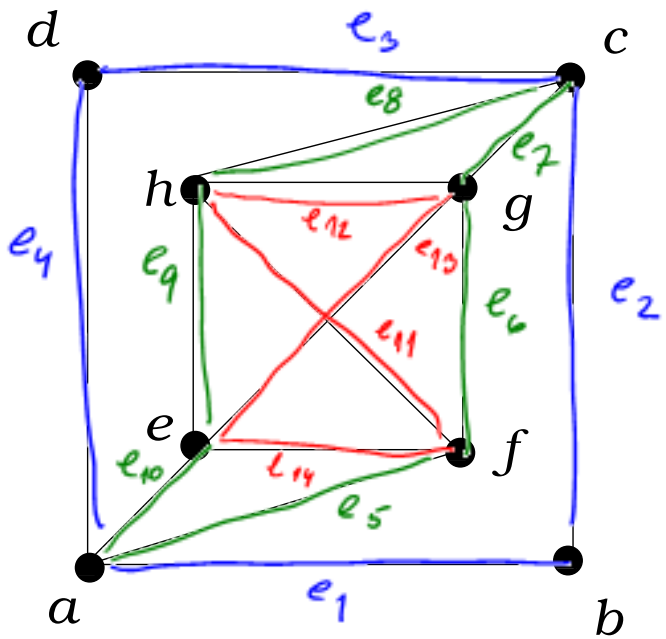
Vstup: eulerovský graf $G = (V, E)$

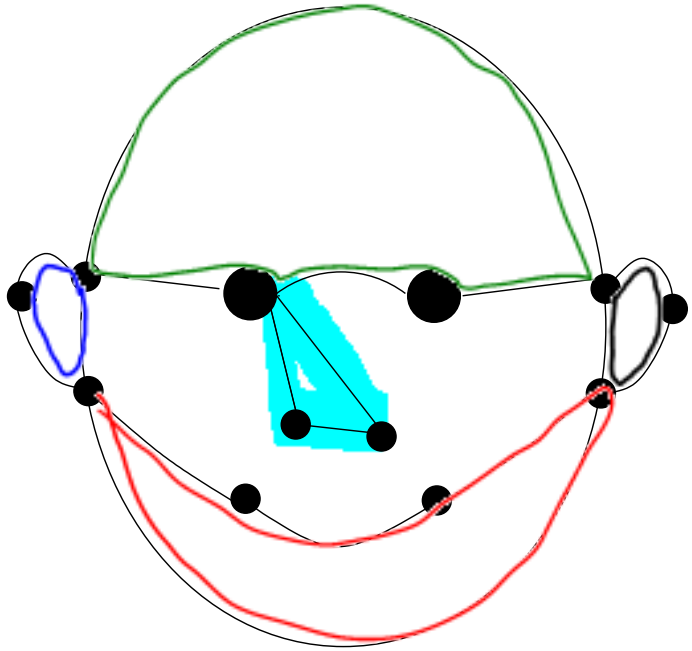
Výstup: eulerovský ťah T

Algoritmus:

- **Krok 1:** Polož $H = G$.
- **Krok 2:** Polož $T = \emptyset$.
- **Krok 3:** Spusť Proc($\tilde{H}, u$) v nejakom súvislom komponente $\tilde{H}$ grafu H v nejakom vrchole $u \in H \cap T$ (ak $T = \emptyset$, $u \in H$), výstup označ T'
- **Krok 4:** Za T polož $T \cup T'$; t.j. ak $T = v_0 e_1 v_1 \dots e_r u e_{r+1} \dots e_i v_0$ a $T' = u f_1 \dots f_s u$, tak $T = v_0 e_1 v_1 \dots e_r u f_1 \dots f_s u e_{r+1} \dots e_i v_0$.
- **Krok 5:** Za H polož $H - E(T')$.
- **Krok 6:** Ak $H = \emptyset$, ukonč, inak choď na Krok 3.

Nakreslite jedným ťahom





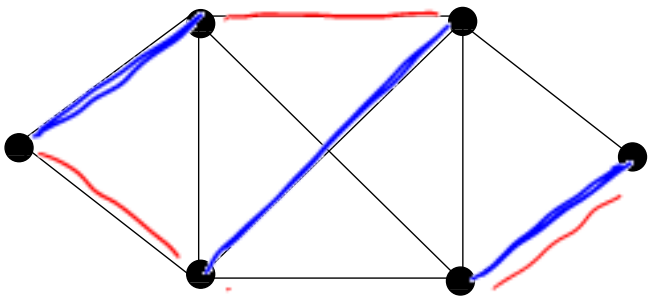
T = a e1 b e2 c e3 d e4 a

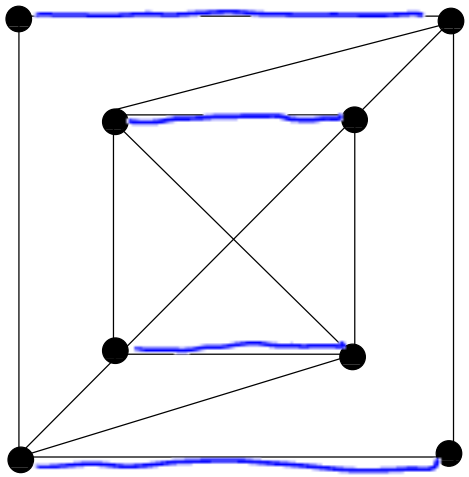
T = a e5 f e6 g e7 c e8 h e9 e e10 a e1 b e2 c e3 d e4

T = a e5 f e11 h e12 g e13 e e14 f e6 g e7 c e8 h e9 e e10 a e1 b e2 c e3 d e4

Perfektné párovanie (perfect matching)

Hovoríme, že graf G s párnym počtom vrcholov $2n$ má **perfektné párovanie**, ak G obsahuje n nezávislých hrán (žiadne 2 nemajúce spoločný vrchol).





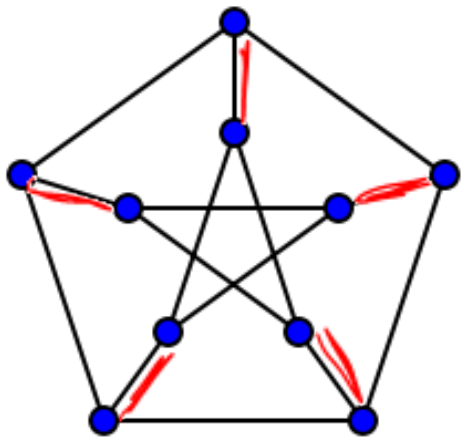
Aplikácie: Priraďovanie činností pracovníkom, sociálne vedy,

Ide o veľmi študovanú oblasť; my si uvedieme len 2 výsledky.

Perfektné párovania - známe výsledky

Veta (König, 1916): Každý pravidelný bipartitný graf s aspoň jednou hranou má perfektné párovanie.

Veta (Petersen, 1891) Každý pravidelný graf stupňa 3 bez mostov má perfektné párovanie. (Tu graf nemusí byť bipartitný!)



Problém obchodného cestujúceho

Problém obchodného cestujúceho spočíva v určení trasy začínajúcej a končiacej v tom istom meste, pričom cestujúci každé iné mesto navštívi práve raz a súčet prejdených vzdialeností (resp. nákladov, resp. časov) bude minimálny možný.

Graf G: vrcholy reprezentujú mestá a hrany cesty medzi mestami. Zároveň nech každá hrana nesie *ohodnotenie* vyjadrujúce napr. dĺžku spojenia, alebo náklady na prejdenie spojenia, alebo čas, atď.

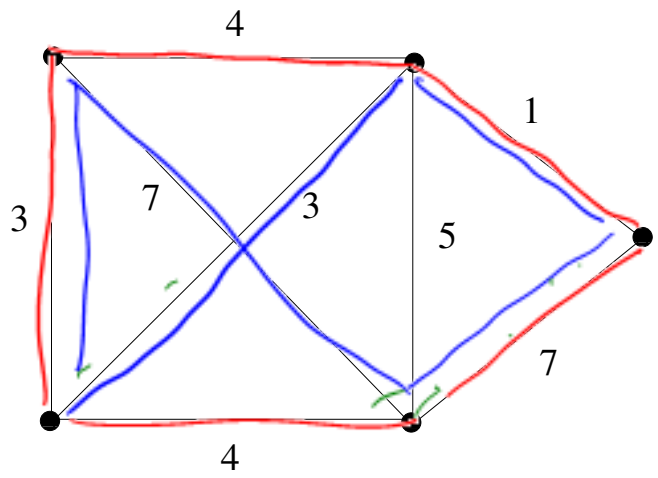
Kružnica v G je **hamiltonovská**, ak obsahuje všetky vrcholy grafu G.

Ekvivalentná formulácia problému obchodného cestujúceho:
Nájsť v G hamiltonovskú kružnicu s najmenším celkovým ohodnotením.

William Rowan Hamilton (anglický matematik 19. stor.).

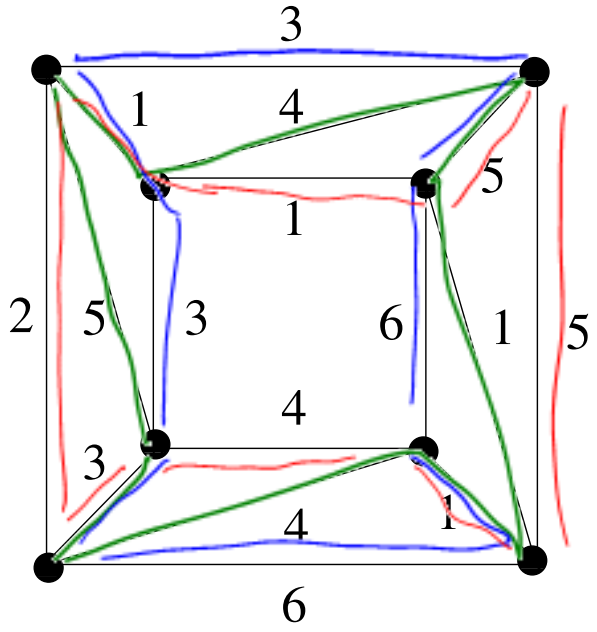
Problém obchodného cestujúceho - príklad

Nájdite optimálnu hamiltonovskú kružnicu.



$3+4+4+7+1=19$ min

$3+3+1+7+7=21$



$3+5+1+4+5+1+1+4 = 24$

$3+5+6+1+6+3+3+1 = 28$

$1+2+3+4+1+5+5+1 = 22$

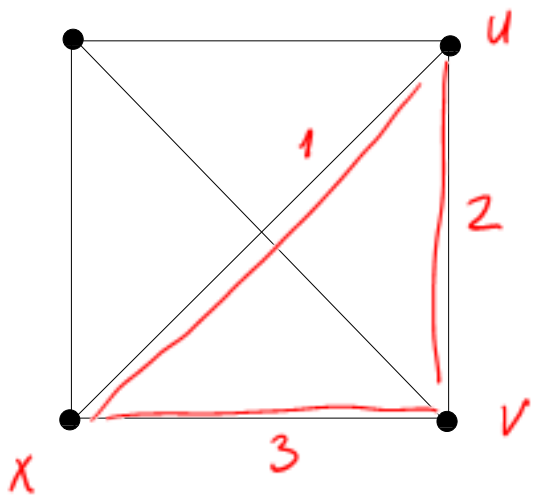
???

Problém obchodného cestujúceho - heuristika

Algoritmy na hľadanie optimálnej hamiltonovskej kružnice – komplikácie!

Vysvetlenie neskôr. Nateraz sa uspokojíme s **heuristikou**, t.j. procedúrou, ktorá nájde hamiltonovskú kružnicu s "rozumne malým" ohodnotením.

Budeme uvažovať o ohodnotenom *úplnom* grafe G (aby sme sa vyhli problému existencie hamiltonovskej kružnice), pričom pre (kladné) ohodnotenie w hrán predpokladáme **trojuholníkovú nerovnosť:** $w(uv) \leq w(ux) + w(xv)$, pre každý vrchol x.



Obchodný cestujúci - heuristika zdvojenej kostry

Heuristika zdvojenej kostry.

- Pomocou Kruskalovho algoritmu nájdeme v G najlacnejšiu kostru T.
- Na T vytvoríme uzavretý sled S, ktorý každú hranu T prejde 2-krát.
- Pomocou S budujeme kružnicu C z ľubovoľného vrchola u rekurzívne:

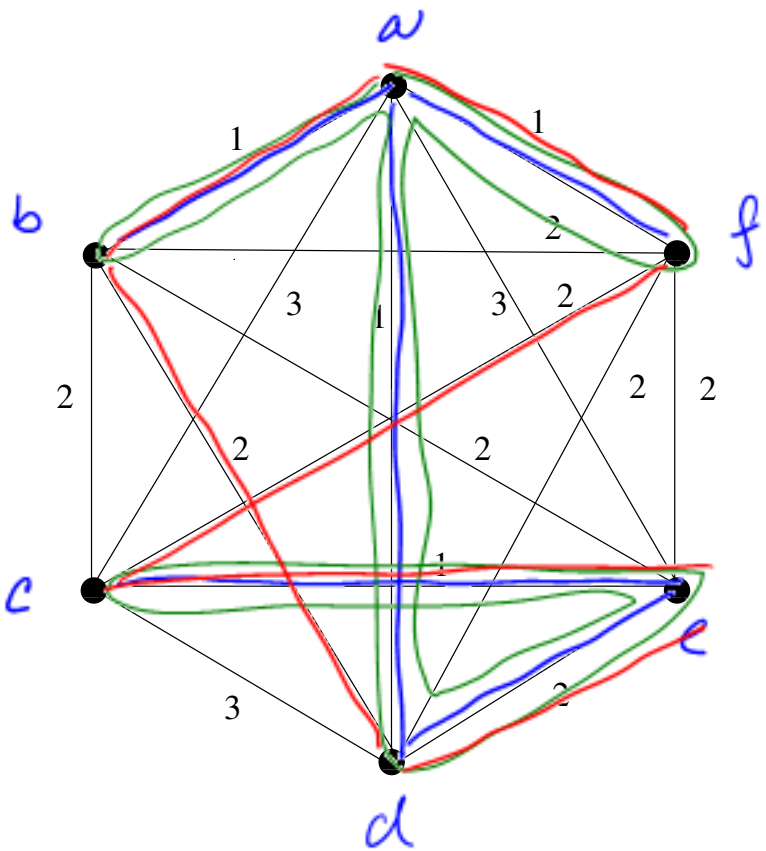
1. Vydáme sa z u v smere sledu S, až kým nie sme nútení prejsť nejakou hranou v opačnom smere, t.j. keď $S = u...vwv...$. Vezmeme dočasne $C = u...vw$.

2. Pokračujeme v traverzovaní sledu S, tentoraz z vrchola w, až po prvý výskyt nejakého vrchola x (čiže teraz $S = u...vwv...x...$), ktorý nie je v našej dočasnej C; do C pridáme vrchol x a hranu wx. Ak taký x nie je, tak bola prejdená celá trasa S; do C pridáme hranu wu a skončíme.

3. Ak $S = u...vwv...xy...$ a $y \notin C$, budujeme ďalej C ako v 1 (x preberie rolu vrchola u v bode 1); ak $y \in C$ (a teda xy je prejdená druhýkrát), pokračujeme ako v 2 (x preberie rolu vrchola w v bode 2).

Z trojuholníkovej nerovnosti: $\text{ohod}(C) \leq 2 \cdot \text{ohod}(T)$ – "rozumne malé".

Príklad na heuristiku zdvojenej kostry



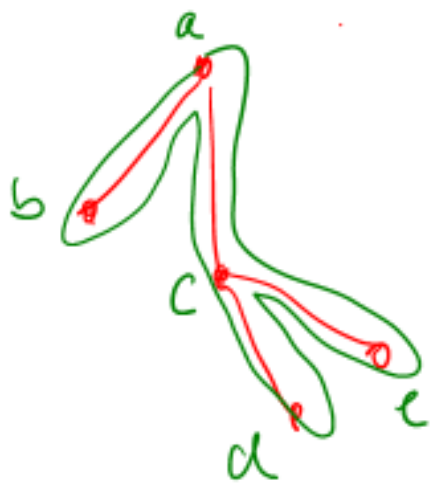
$S = a\ b\ a\ d\ e\ c\ e\ d\ a\ f\ a$

$C = a\ b\ d\ e\ c\ f\ a$ → VÝSTUP

HAMILT. KRUŽNICA

hodnota:

$1 + 2 + 2 + 1 + 2 + 1 = 9$



$S = a\ b\ a\ c\ d\ e\ e\ c\ a$

Algoritmická zložitosť – odľahčený úvod

Algoritmické riešenia problémov - Príklady:

(1) zostrojiť kostru daného grafu

(2) nájsť Prüferov kód grafu

(3) rozhodnúť, či daný graf je súvislý

(4) rozhodnúť, či je graf hamiltonovský

Výstupy algoritmov:

- *objekt*
- odpoveď ÁNO alebo NIE – *rozhodovacie* problémy

Jednoduché?
Graf, v ktorom máme niečo nájsť alebo o ňom urobiť nejaké rozhodnutie, môže mať tisíce vrcholov. Na vstupe algoritmu je v podobe zoznamu vrcholov a ich susedov, prípadne v tvare matice. Počítač ten graf nevidí!

Opíšeme teraz základný pojem **zložitosti** rozhodovacieho problému; presnú definíciu sa dozviete neskôr v špeciálnych predmetoch.

Algoritmická zložitosť

Pod **zložitosťou** algoritmického problému budeme rozumieť počet "krokov" $f(n)$, ktoré algoritmus v najhoršom prípade vykoná, aby na výstupe dal odpoveď ÁNO alebo NIE, ak "dĺžka vstupu je n".

Či už počet krokov meriame pomocou opisu algoritmu alebo pomocou taktov procesora, dostávame síce rôzne čísla – n a $100n$ – ale stále je to konštanta krát n, kde konštanta závisí od mašiny a implementácie.

V informatike túto situáciu zapíšeme symbolom $O(n)$.

Ak teda $f(n)$ z opisu zložitosti je počet "krokov" nutných na prejdenie všetkých n vrcholov, tak $f(n) = O(n)$, čo formálne znamená, že $f(n) \leq cn$ pre nejakú konštantu c a pre všetky n.

Podobne budeme hľadieť na pojem "dĺžka vstupu".

Či už je to zoznam vrcholov, ako ho máme na papieri, alebo prekódovaný na postupnosť núl a jednotiek, vždy v prípade napr. cesty na n vrcholoch môžeme rovnako dobre povedať, že vstup má dĺžku $O(n)$.

Obvykle sa symbol O používa len na zložitosť, a nie na dĺžku vstupu

Algoritmická zložitosť

Príklad: Rozhodovací problém má zistiť, či graf s n vrcholmi a m hranami, zadaný na vstupe, je súvislý.

Môžeme teda predpokladať, že dĺžka vstupu je $n + 2m$.
Počet krokov prehľadávacieho algoritmu je $O(n + 2m)$.
Uvedomme si, že každú hranu navštívime najviac dva razy.

Niekedy sa zložitosť udáva *len* v terminológii počtu vrcholov n grafu. Keďže $m \leq n(n-1)/2$, môžeme jednoducho povedať, že zložitosť prehľadávacieho algoritmu je nanajvýš $O(n^2)$. Podstatné je, že n^2 je *polynóm* v premennej n. Je to rastúca funkcia, ale nie "divoko" rastúca.

Algoritmy, ktoré na rozhodovacie problémy o grafoch s n vrcholmi dajú odpoveď ÁNO alebo NIE a spotrebujú pri tom najviac $O(n^k)$ krokov pre *konštantné* k, sa nazývajú **polynomiálne**, alebo **patriace do triedy** P.

Príklady: Určenie, či priemer grafu je $\leq d$ pre dané d; rozhodnutie, či graf má perfektné párovanie; alebo či daný graf je rovinný, atď.

Algoritmická zložitosť

Je však celý rad rozhodovacích problémov, na ktoré zatiaľ nepoznáme žiaden polynomiálny algoritmus.

Príklady: Rozhodnúť, či daný graf s n vrcholmi na vstupe je hamiltonovský, alebo či jeho chromatické číslo je $\leq \ell$ pre ľubovoľné *konštantné* $\ell \geq 3$.

Aj tie najlepšie známe algoritmy na tieto úlohy dokážu spracovať vstupný graf s n vrcholmi v najlepšom prípade až po $O(a^n)$ krokoch, kde $a > 1$.

V informatike sa skúma celá trieda takýchto problémov, ktoré možno "rýchlo" – v polynomiálnom čase – pretransformovať jeden na druhý, a v konečnom dôsledku na rozhodnutie, či daný n-vrcholový graf na vstupe je hamiltonovský.

Tejto triede problémov sa hovorí **NP-úplné problémy**.
(NP znamená *nedeterministické polynomiálne*).

Algoritmická zložitosť

Rozdiel medzi zložitosťou $O(n^k)$ polynomiálneho algoritmu a zložitosťou $O(a^n)$ NP-úplného problému.

Príklad: Určte rozdiel medzi zložitosťou $O(n^3)$ a $O(2^n)$ pre rôzne hodnoty vstupu n.

- Ak $n = 10$, tak $O(10^3) = c \cdot 1000$ a $O(2^{10}) = C \cdot 1024$
- Ak $n = 100$, tak $O(100^3) = c \cdot 1000000$ a $O(2^{100}) = C \cdot 1267650600228229401496703205376$
- Ak $n = 400$, tak (až na multiplikatívnu konštantu) porovnávame $400^3 = 64000000$ s 2^{400}; to druhé je viac, ako fyzikmi odhadovaný počet elementárnych častíc vo vesmíre!

Algoritmická zložitosť

P-NP problém: Jeden z najslávnejších problémov v súčasnosti zo zoznamu 7 miléniových problémov Clayovho matematického inštitútu, Massachusett, USA. Vyriešenie každého z nich je dotovaný miliónom USD! (Jeden z nich už je vyriešený.)

P-NP problém je otázka, či P=NP; v ekvivalentnej formulácii, či existuje algoritmus polynomiálnej zložitosti na rozhodnutie, či ľubovoľný daný n-vrcholový graf je hamiltonovský. (Vzhľadom na ekvivalentnosť v triede NP-úplných problémov by to znamenalo existenciu polynomiálnych algoritmov pre *všetky* NP-úplné problémy.)

Verí sa, že P≠NP, ale nikto to nevie dokázať! Tu je významný problém *izomorfizmu* grafov: Rozhodnúť, či dva n-vrcholové grafy na vstupe sú izomorfné. Zatiaľ nepoznáme žiaden polynomiálny algoritmus na tento problém, ale na druhej strane ani nikto nevie dokázať, že je NP-úplný!

Predpokladá sa, že ide o problém, ktorý je striktne medzi P a NP! Ak by to niekto dokázal, tak by zároveň vyriešil aj P-NP problém.