DFS Opakování

Standardní průchod bludištěm

1. Vydej se do bludiště. Kdykoli půjdeš nějakou chodbu, označ ji značkou, Když půjdeš chodbou podruhé (to poznáš podle toho, že už má značku), označ ji druhou značkou. Chodba s jednou značkou bude tvá chodba otevřená, chodba se dvěma značkami bude tvá chodba zavřená, Chodba jíž jsi ještě nešel, bude chodba čerstvá. Žádnou chodbou nepůjdeš více než dvakrát.
2. Když přijdeš na rozcestí, spočti, kolik z něj nebo do něj vede chodeb otevřených, včetně té, jíž jsi právě přišel.
3. Když jsou na rozcestí tři chodby otevřené, vrať se ihned otevřenou chodbou, jíž jsi právě přišel, čímž ji zavřeš.
4. Když do/z rozcestí vede jediná chodba otevřená, podívej se, jestli z rozcestí vede nějaká čerstvá chodba. Pokud ano, pokračuj touto chodbou, jinak pokračuj jedinou chodbou otevřenou, čímž ji zavřeš,
5. U chodby, která nikam nevede a končí zdí, si představujeme, že na jejím konci je také rozcestí a provede se v něm druhá půle pravidla 4.

Druhy rozcestí

Když do/z rozcestí vedou pouze chodby čerstvé, je to rozcestí čerstvé (FRESH).
Když do/z rozcestí vedou pouze chodby uzavřené,je to rozcestí uzavřené (CLOSED).
Všechna ostatní rozcestí jsou otevřená (OPEN).

Životní cyklus rozcestí

Než vstoupíš do bludiště, je každé rozcestí druhu FRESH. Postupně jak procházíš bludištěm, mění se druh každého rozcestí nejprve na OPEN a pak na CLOSED.

Opakovaný start

Po návratu z bludiště se může stat, že některá rozcestí zůstala neobjevena (FRESH), protože do nich z původního vchodu do bludiště nevedou žádné chodby. Je tedy třeba postupně vyzkoušet všechny dosud nenavštívené vchody do bludiště. Každý vchod do bludiště považujeme take za rozcestí.

DFS strom

Všechny chodby, které někdy během DFS spojují OPEN rozcestí s rozcestím FRESH, to jest takové chodby, pomocí níchž objevujeme dosud nenavštívená rozcestí, nazveme stromovými chodbami. Všechny stromové chodby dohromady tvoří les a neobsahují žádnou kružnici.

Časové značky v DFS

Máme nespojitý přidaný čas, jiný než ten fyzický Newtonovský, Einsteinovský atd., který přibývá po jednotkách. Každému rozcestí postupně připisujeme otevírací a zavírací čas. Kdykoli narazíme rozcestí jež bylo dosud FRESH, čas přibyde o 1 a na rozcestí napíšeme jeho hodnotu, to bude otevírací čas rozcestí. Kdykoli se chystáme opustit rozcestí, jež vzápětí bude CLOSED, čas také přibyde o 1 a vedle otevíracího času připíšema na rozcestí ještě aktuální hodnotu času a to bude zavírací čas rozcestí. Po projití celého bludiště bude mít každé rozcestí určeno jednoznačně otvírací a zavírací čas.

Acyklické grafy

Veškeré uvedené postupy mají složitost úměrnou počtu hran v grafu nebo, když má graf méně hran než uzlů, počtu uzlů v grafu. Postupy mají přehledový charakter, někdy je lze dále drobně vylepšovat, což také závisí na aktuální úloze a implementačních omezeních, volbě datových strukrur atd.

Detekce acykličnosti grafu
Spustíme DFS z libovolného uzlu, v uzlech registrujeme čas otevření a uzavření.
Pokud při pohybu dopředu v DFS najdeme uzel, který je otevřený a není uzavřený, pak jsme právě v dopředném pohybu uzavřeli cyklus, a tudíž graf acyklický není. Pokud tato situace nenanstane během celého DFS, graf je acyklický.
Čas otvírání/zavírání počítejme od 1, časy otevření/zavření všech uzlů inicializujme 0, otevřený uzel poznáme tak, že má nenulový čas otevření a (ještě stále) nulový čas zavření.

Topologické uspořádání
Provedeme DFS pro celý graf (může být nesouvislý) a potom uzly uspořádáme v pořadí klesajících zavíracích časů. Lze to udělat i efektivněji tak, že si definujeme dodatečný zásobník Z a během DFS vždy, když zavíráme uzel, uložíme na Z referenci na tento uzel. Po skončení DFS jsou na zásobníku shora dolů reference na uzly ve správném topologickém pořadí, stačí je postupně odebírat operací pop.
Topologických uspořádání může být více, pro většinu účelů se hodí kterékoli. Jediné topologické uspořádání existuje právě tehdy, když v grafu s n uzly existuje cesta délky $n-1$.

Nejdelší cesta v acyklickém grafu
1. Graf topologicky uspořádáme.
2. Každý uzel x si bude pamatovat dva údaje: Délku nejdelší cesty, která do něho vede, a svého bezprostředního předchůdce na této cestě.
3. Do všech uzlů zapíšeme nulu jako délku nejdelší cesty a null jako předchůdce.
4. Uzly grafu projdeme v pořadí topologického uspořádání a v každém uzlu x provedeme s každým jeho sousedem y (ten je díky top. usp. vpravo od x) operaci:
   pokud délka nejdelší cesty do y je menší než jedna plus délka nejdelší cesty do x,
   potom přepiš délku nejdelší cesty do y na jedna plus délka nejdelší cesty do x a jako předchůdce y zaznamenej x.
5. Nakonec probereme všechny uzly a určíme ten, do nějž vede nejdelší cesta, případně to můžeme registrovat průběžně již v bodě 4.

Nejdelší cesta ve váženém acyklickém grafu
V předchozím algoritmu nahradíme výraz "jedna" výrazem "váha hrany (x, y)":
1. Graf topologicky uspořádáme.
2. Každý uzel x si bude pamatovat dva údaje: Délku nejdelší cesty, která do něho vede, a svého bezprostředního předchůdce na této cestě.
3. Do všech uzlů zapíšeme mínus nekonečno jako délku nejdelší cesty a null jako předchůdce, do kořenů zapíšeme 0 jako délku nejdelší cesty.
4. Uzly grafu projdeme v pořadí topologického uspořádání a v každém uzlu x provedeme s každým jeho sousedem y (ten je díky top. usp. vpravo od x) operaci:
   pokud délky nejdelší cesty do y je menší než váha hrany (x, y) plus délka nejdelší cesty do x,
   potom přepiš délku nejdelší cesty do y na váhu hrany (x, y) plus délka nejdelší cesty do x a jako předchůdce y zaznamenej x.
5. Nakonec probereme všechny uzly a určíme ten, do něhož vede nejdelší cesta, případně to můžeme registrovat průběžně již v bodě 4.

Počet všech cest z uzlu *a* do uzlu *b* v acyklickém grafu

1. Graf topologicky uspořádáme.
2. Každý uzel x si navíc bude pamatovat jediný údaj: Počet všech cest, které do něho vedou z uzlu *a*.
3. 3. Do každého uzlu zapíšeme nulu jako počet všech cest, které do něho vedou z uzlu *a*, jenom do *a* zapíšeme 1.
4. Postupně projdeme uzly počínaje *a* v topologickém uspořádání a v a každém uzlu *x* provedeme s každým jeho sousedem *y* (ten je díky top. usp. vpravo od *x*) operaci:
   K počtu cest vedoucích do uzlu *y* z uzlu *a* připočti počet cest vedoucích do uzlu *x* z uzlu *a*.
5. Skončíme v okamžiku, kdy v bodě 4. bude platit $x = b$. U *b* pak bude připsán počet všech cest z uzlu *a* do uzlu *b*.

Počet vůbec všech cest v acyklickém grafu

1. Graf topologicky uspořádáme.
2. Každý uzel x si navíc bude pamatovat jediný údaj: Počet všech cest, které do něho vedou (odkudkoli).
3. 3. Do všech uzlů kromě kořenů zapíšeme nulu jako počet všech cest, které do něho vedou, do kořenů zapíšeme 1.
4. Postupně projdeme uzly v topologickém uspořádání a v a každém uzlu *x* provedeme s každým jeho sousedem *y* (ten je díky top. usp. vpravo od *x*) operaci:
   K počtu cest vedoucích do uzlu *y* připočti počet cest vedoucích do uzlu *x*.
5. Nakonec sečteme počty cest vedoucích do všech uzlů, které nejsou kořenem.

Nejkratší cesta mezi uzly *a* a *b* v acyklickém grafu

Použijeme BFS se začátkem v *a*.

Nejkratší cesta mezi uzly *a* a *b* ve váženém acyklickém grafu

Můžeme použít Dijkstrův algoritmus se začátkem v *a*, ale má to háčky: Váhy hran nesmí být záporné, k selhání Dijkstrova algoritmu stačí i jediná záporně ohodnocená hrana a ani acykličnost nepomůže, leda by chom měli jistotu, že graf je stromem. Pak ale jistě není zapotřebí Dijkstrova algoritmu, stačí jednoduchý BFS/DFS. Navíc většinou asymptotická složitoast Dijkstrova algoritmu je o něco větší než pouze úměrná počtu hran (např. vynásobená logaritmem počtu uzlů, apod).

Výhodnější a asymptoticky přinejmenším stejně rychlá alternativa, které nezáleží na záporně ohodnocených hranách je podobná algoritmu pro hledání nejdelší cesty, jen nerovnosti jsou otočené:

1. Graf topologicky uspořádáme.
2. Každý uzel x si navíc bude pamatovat dva údaje: délku nejkratší cesty z *a*, která do něho vede a svého bezprostředního předchůdce na této cestě.
3. Do všech uzlů zapíšeme nekonečno jako délku nejkratší cesty z *a* a null jako předchůdce, do uzlu *a* zapíšeme nulu jako délku nejkratší cesty.
4. Uzly grafu projdeme v pořadí topologického uspořádání počínaje uzlem *a* a v každém uzlu *x* provedeme s každým jeho sousedem *y* (ten je díky top. usp. vpravo od *x*) operaci:
   pokud délky nejkratší cesty z *a* do *y* je větší než váha hrany (x, y) plus délka nejkratší cesty z *a* do *x*,
   potom přepiš délku nejkratší cesty z *a* do *y* na váhu hrany (x, y) plus délka nejkratší cesty z *a* do *x*.
   a jako předchůdce *y* zaznamenej *x*.
5. Skončíme v okamžiku, kdy v bodě 4. bude platit $x = b$. U *b* pak bude připsána délka nejkratší cesty z *a* do *b*.

Silné komponenty

Silná komponenta je taková množina uzlů grafů, že mezi každou dvojicí uzlů této množiny vede nějaká orientovaná cesta (ve směru šipek - orientace hran) tam i zpět. Přitom tato množina je maximální možná ve smyslu inkluze. Uzly, které neleží na žádné kružnici, představují také silné komponenty.

První možnost nalezení silných komponent

1. Provedeme DFS
2. Otočíme všechny hrany grafu do opačného směru
3. Provedeme DFS v upraveném grafu přičemž uzly otvíráme v klesajícím pořadí zavíracích časů prvního DFS.
4. Každý DFS strom po skončení bodu 3 představuje kostru právě jedné silné komponenty.

Druhá možnost nalezení silných komponent - Tarjanův algoritmus.
U každého uzlu budeme registrovat dodatečný údaj tzv. lowlink a navíc použijeme ještě jeden zásobník S.
Lowlink uzlu *x* se snaží registrovat nejnižší otevírací dobu všech otevřených uzlů, do kterých se podařilo nahlédnout mezi otevřením a uzavřením uzlu *x*.
Při otevření uzlu se jeho lowlink nastaví na jeho otevírací čas.
Když hledáme, kam dále pokračovat v DFS z uzlu *x* a probíráme přitom sousedy uzlu *x*, pak pro každého otevřeného souseda *z* uzlu *x* provedeme:
lowlink *x* := minimum dosavadního lowlinku *x* a otevíracího času *z* (do *z* jen nahlédneme).
Při uzavírání uzlu *x* a návratu do jeho předchůdce *y* ve stromu prohledávání se lowlink *y* nastaví na minimum z lowlinku *x* a lowlinku *y*.

Detekce silné komponenty
Pokud při uzavírání uzlu *x* je jeho lowlink *x* roven otevíracímu času *x*, znamená to, že jsme našli novou silnou komponentu, kterou určíme pomocí zásobníku S.

Na zásobník S ukládáme reference na uzly. Při otevřeni uzlu *x* uložíme na S referenci na *x*. Uzly z S odstraňujeme pouze tehdy, když dojde k detekci silné komponenty v předchozím odstavci. Potom odstraňujeme reference z vrcholu S tak dlouho, dokud neodstraníme i referenci na uzel *x*, v němž k detekci došlo. Všechny tyto odstraněné reference představují jednu silnou komponentu grafu.