Doprovodný soubor k zápočtovému programu Graphs

Uživatelská dokumentace

Interaktivní program slouží k tvorbě a následnému vykreslení orientovaného nebo neorientovaného grafu včetně možnosti na něm pustit vybrané grafové algoritmy.
Byl vyvinut pomocí platformy *Microsoft Visual Studio Community 2019* v jazyce C#.

Návod k použití

Po spuštění programu se před Vámi objeví okno rozdělené na dvě části.

Tmavá část, vlevo, pracuje pouze jako místo k vykreslování a tvorbě grafu. Pro vytvoření vrcholu klikněte levým tlačítkem myši. Po jeho vytvoření se z něho začne natahovat hrana, kterou je možno zakotvit v jiném vrcholu, nebo s ní vytvořit nový, znovu, kliknutím levého tlačítka myši. Pokud chcete táhnutí zastavit – klikněte pravým tlačítkem u myši.
Pro tvorbu hrany je nutné mít k dispozici alespoň jeden vrchol, na který je kliknuto levým tlačítkem.
Vprostřed vytvořené hrany je zobrazena její váha – tu lze změnit kliknutím levého tlačítka dostatečně do zmíněného středu hrany - bohužel, ne na váhu samotnou. Po kliknutí máte prostor pro psaní - můžete využít pouze čísel, tlačítka pro znaménko minus a tlačítka backspace pro mazání dosud napsaného čísla. Pro ukončení psaní kamkoliv klikněte, nebo stiskněte jiné než zmíněné povolené klávesy.
Orientovanost hran je možné zařídit kliknutím na jejich konce levým tlačítkem myši. To samé lze říci i pro odstranění orientovanosti. Je možnost tvořit i dvojitě orientované hrany – stačí kliknout na druhý konec, proti směru orientované hrany. Zobrazí se druhá váha – přepisovaní se teď změnilo – přepsat specifickou váhu lze kliknutím na hranu ve směru orientovanosti dotyčné hrany, jejíž váhu chceme přepsat.
Aby byl přehled o očíslování jednotlivých vrcholů – je proto možné jej přejetím myši po vrcholu zjistit.
K dispozici je i mazání dosud vytvořených objektů – lze ho docílit stisknutím pravého tlačítka

u myši přímo na odstraňovaný objekt. S mazaným vrcholem se odstraní i hrany, které z něj vedou.

Světlá část, vpravo, funguje, zkráceně, jako okno pro práci s algoritmy a maticí sousednosti, dole, která zachycuje vykreslený graf. Vytvořený graf, tedy jeho matici sousednosti, lze tlačítkem *Copy* zkopírovat. Okno v dolní části panelu lze využít pro nahrání své vlastní matice sousednosti, která se po kliknutí na tlačítko *Insert* sama nahraje a v tmavé části vykreslí jako graf. Nahrávat lze pouze ve formátu s hranatými závorkami, kde jsou jednotlivé elementy odděleny čárkou - [[0, 1], [1, 0]].
V horní části je možné si tlačítky „◀ ▶“ vybrat algoritmus, který na grafu pustit – tlačítkem *Start*. Po spuštění se napravo od textu *Results:* objeví výsledky v různých formátech. Pro některé algoritmy je možné si vedle textu „Pick a source“ vybrat počáteční vrchol algoritmu – pokud není vybrán, je automaticky 1.
Každý algoritmus má graf pro příklad – nahrát ho lze kliknutím na tlačítko v pravém dolním rohu pod textem Import example.

Programátorská dokumentace

Program je rozdělen do pěti veřejných tříd – Form1, Vertex, Edge, Heap, Algorithms. Tvořený graf se ukládá do matice sousednosti, jednotlivé vrcholy do List<Vertex>, hrany do List<Edge>. Po kliknutí na tlačítko Start se vytvoří nový objekt třídy Algorithms, kde se v jeho konstruktoru zpracuje předaná matice sousednosti, a spustí se vybraný grafový algoritmus. Jeho výsledek se přenese jako string do jednoho z Labelů u textu *Results:*.

public partial class Form1 : Form

private void **Clicked** – Hlavní řídicí metoda. Spustí se po kliknutí na Panel1, rozřadí typ kliknutí vyhodnotí situaci a pošle signál dál. Komunikuje s dragTimerem, který ovládá tažení hrany, s LeftClickedVertex a LeftClickedEdge. Dále používá pomocné funkce při tvorbě a vykreslování vrcholů a hran.

public void **AdjustMatrix** – Slouží pouze pro upravení matice sousednosti.

public void **CheckMatrixSize** – Po každém přidání vrcholu se spustí a zkontroluje, jestli nepřeteče matice sousednosti. Pokud ano, dvakrát se nafoukne a všechny hrany se přepíší. Tzn. časová složitost je úměrná počtu hran, nikoliv počtu prvků v matici.

public string **PrintMatrix** – Pouze přepíše čtvercové matice do *stringu* s respektem k nekonečnům.

public void **InsertMatrix** – Tato nepřehledná funkce slouží pro čištění, ke kontrole a pro přečtení a zapsání uživatelského vstupu z pasteBoxu. Funkce je spuštěná tlačítkem Insert.

public string **MatrixToString** – Stejně jako funkce PrintMatrix slouží k převodu matice na *string*. Rozdíl je v tom, že slouží pouze pro zkopírování na pozdější použití. Kopíruje se s hranatými závorkami. Spouští se tlačítkem Copy.

public void **InitEdge** – Pomáhá inicializovat hranu. Pokud tvořím orientovanou, pomůžu si funkcí **LeftClickedEdge**.

public void **LeftClickedVertex** – Nastanou dvě možnosti, pokud táhnu hranu (drag == *true*) a kliknu na vrchol – vytvoří se hrana a upraví se matice sousednosti a drag se nezastaví. Pokud netáhnu, tak zachytím předchozí stav obrazovky a začnu.

public void **LeftClickedEdge** – Zachycuje dva scénáře – klikám na konce hran, abych změnil orientaci, nebo na prostředek, abych přepsal váhu hrany. Spočítám vzdálenosti myši od konců hran. Pokud je menší z nich méně než 40, nastal první scénář – najdu vektor mezi vrcholy, přeškáluji si ho. Chci po něm ujít o kousek blíže směrem ke středu hrany, kde můžu zakotvit znovu přeškálovaný, teď už normálový, vektor tak, že pokud ho jednou přičtu a jednou odečtu, získám body, ze kterých spustím čáry do vrcholu tak, aby tvořily čepičku šipky. Nejdříve si ale zjistím, o jakou hranu se jedná a podle toho se zachovám. Pokud je hrana jednoduše orientovaná – mohu kliknout buď na konec, kam směřuje – pak udělám z orientované hrany neorientovanou, nebo na opačný konec – z jednoduše orientované vytvořím oboustranně orientovanou. Pokud je hrana oboustranně orientovaná, když už na ni klikám, nastane pouze jedna možnost a to taková, že se z ní stane hrana jednoduše orientovaná. A pokud chci přepisovat váhu hrany – obě vzdálenosti myši od vrcholů jsou větší než 40 – zjistím si jakou váhu přepisuji, začnu načítat znaky a spustím writeTimer.

public void **DeleteVertex** – Nastane, pokud se netáhne hrana a uživatel kliknul jiným než levým tlačítkem myši na vrchol. Odstraní se, jak vrchol, tak hrany, které z něj, nebo do něj vedou. Upraví se matice sousednosti, přečíslují se vrcholy, s vyšším číslem, než má ten, který je odstraňován a přemaluje se graf.

public void **DeleteEdge** – Upraví se matice sousednosti a smaže se dotyčná hrana.

public void **PaintOver** – Obsahuje dvě vnořené funkce – na namalování hran a na namalování vrcholů. Nejdříve se ale celý panel vybarví do barvy pozadí – tzn. přemalují se nakreslené objekty.

public void **DrawEdge** – Nastaví se kvalita, zaoblí se pero a namaluje se čára mezi specifikovanými vrcholy.

public void **DrawVertex** – Viz DrawEdge, ale s Ellipse.

public Vertex **GetHoveredVertex** – Pro určitou pozici myši se zjistí, jestli náhodou nespadá do kružnice nějakého z vrcholů. Pokud ano, vrátí se dotyčný vrchol.

public Edge **GetHoveredEdge** – Pro každou hranu spočítá, jak daleko je od přímky, která má směrový vektor hrany, vzdálená pozice myši. Pokud je vzdálená na méně než 7, zkontroluji, jestli se kurzor nachází na hraně, a ne na přímce.

public Vertex **HoveredV** – Zjištěný vrchol od funkce GetHoveredVertex, na kterém se nachází kurzor myši, kosmeticky zvětší, nebo zmenší.

public Edge **HoveredE** – Zjištěnou hranu od funkce GetHoveredEdge zvětší nebo zmenší podle potřeby.

private void **dragUpdate_Tick** – Pokud táhnu hranu, tento timer ji každých 20 milisekund překreslí pozadím zachyceným v předchozím kompletním stavu. Pokud ji táhnu přes vrchol, přitáhnu ji jako magnet doprostřed vrcholu. Hranu nepřekresluji, pokud to není nutné.

private void **fixedUpdate_Tick** – Každých 20 milisekund pouští funkce, které kontrolují, jestli se kurzor nevyskytuje na jednom z nakreslených objektů. Následně informaci zpracuje.

private void **fixedUpdate_Start** – Spustí se, pokud se myš dostane na Panel1.

private void **fixedUpdate_End** – Spustí se, pokud se myš dostane z Panel1.

private void **copyBtn_Click** – Zkopíruje matici sousednosti.

private void **insertBtn_Click** – spustí InsertMatrix.

private void **Form1_Resize** – Pokud je okno přeškálováno, přemaluje nakreslený graf.

private void **Form1_LocationChanged** – Pokud je pozice okna změněna, přemaluje graf.

public void **StopWriting** – Zastaví načítání znaků společně s writeTimer, vyčistí napsaný text, převede ho na číslo, uloží ho do správné váhy hrany a přepíše matici sousednosti.

private void **writeTimer_Tick** – Dodá vizuální efekt kurzoru při psaní.

private void **Form1_KeyPress** – Zapisuje všechny znaky, které jdou přepsat na číslo. Pokud je aktivována jiná klávesa, spustí StopWriting.

public bool **validDistances** – Pokud je každý vrchol vzdálený od myši alespoň o specifikovanou kritickou vzdálenost, vrátí *true*, jinak *false*.

public void **CapturePreviousState** – Do Bitmap uloží záznam obrazovky Panelu1, vytvoří z něj TextureBrush a z něj Pen. Obě hodnoty uloží.

public float **EvalDistance** – Spočítá euklidovskou vzdálenost mezi dvěma body.

public void **ScaleVector** – Přeškáluje vektor tak, aby měl specifickou délku.

public Point **GetTheMiddle** – Vrátí prostřední bod mezi dvěma body.

public void **switchLabels** – Podle hodnoty *alg* změní text a algoritmus v menu.

private void **rightBtn_Click** – Zvýší hodnotu *alg* a zavolá switchLabels.

private void **leftBtn_Click** – Sníží hodnotu *alg* a zavolá switchLabels.

private void **startBtn_Click** – Zjistí, jestli je některá z hran orientovaná, jaký je source vrchol a podle *alg* spustí specifický algoritmus, vypíše výsledek.

private void **exampleBtn_Click** – Podle hodnoty *alg* předá předpřipravený string funkci InsertMatrix.

public class Vertex

- Třída vrcholu. Udržuje si číslo, poloměr vrcholu (kvůli hover funkcím) a svoji pozici.

public class Edge

- Třída hrany. Udržuje si vrcholy, které spojuje, orientovanost, seznam dvou sad tří Pointů, které udávají: na nultém indexu – pozici vrcholu kam (při orientovanosti) hrana vede a na zbylých dvou – pozice odkud vedou části šipky. Dále obsahuje svoji váhu jako integer a weighLabel, který váhu / váhy zobrazuje.

public class Heap

- Třída binární haldy s prvky tvaru immuntable tuple integerů. Nepoužívá nultý index a pro každý prvek v haldě je zaznamenána jeho pozice v *pos* poli integerů, které dostane z konstruktoru svoji délku (ta je určena maximální hodnotou klíče).

public void **Insert** – Přidá na poslední pozici v haldě, zaznamená si ji a nechá vybublat nahoru pomocí BubbleUp.

private void **BubbleUp** – Vybublá nahoru s tím, že pokaždé upraví pozici prvku. Pracuje v čase O(log n).

public int **ExtractMin** – Uloží si key prvku z prvního indexu, poslední prvek haldy vymění s prvním a zabublá dolů. Při této akci vždy upravuje pozice jednotlivých prvků v externím poli *pos*. Jakmile dobublá v čase O(log n), vrátí uložený klíč.

public void **DecreaseKey** – Dostane klíč prvku a hodnotu, na kterou ji má u klíče snížit. Podívá se do pole *pos*. Pokud se v haldě nevyskytuje vrátí se, pokud se zde vyskytuje, zavolá na něj funkci BubbleUp.

public bool **Empty** – vrací false, pokud je halda neprázdná (má více jak jeden prvek). Jinak true.

public class Algorithms

- Třída pro grafové algoritmy. V konstruktoru dostane matici sousednosti a kolik prvků z ní má použít – tu předělá s respektem k počtu vrcholů a také z ní vytvoří seznam následníků. Obojí se, společně s počtem vrcholů (*n*), uloží.

int[,] **GetMatrix** – Zapíše si specifický čtverec matice do matice sousednosti objektu Algorithms.

int[][] **seznamNasledniku** – S pomocným Listem přepíše matici sousednosti do seznamu následníků.

bool **Relax** – Relaxace hrany. Pokud funkce relaxovala vrátí *true*.

Tuple<string, int[], int[], List<int>, List<(int, int)>, List<(int, int)>, List<(int, int)>, List<(int, int)>> **DFS** – Podrobný algoritmus opakovaného prohledávání do hloubky. Vrátí všechny komponenty ve formě stringu, časy otevření, uzavření, vrcholy – tak jak se uzavíraly, mosty, zpětné, dopředné a příčné hrany. O(m + n)

public string **soloDFS** – Jednoduché prohledávání do hloubky, které se spustí od uživatelem vybraného vrcholu.

public string **BFS** – Prohledávání do hloubky se specifikovaným source vrcholem.

public Tuple<int[], int[]> **Dijkstra** – Implementace Dijkstra algoritmu s binární haldou. O(m + n*log n)

public string **Components** – Zavolá podrobný DFS algoritmus a vrátí pouze string s komponenty.

public int[] **SSK** – Vezme vrcholy tak, jak se uzavíraly v podrobném DFS a na transpozici grafu pustí prohledání do hloubky. Každý DFS strom je silně souvislou komponentou. Vrátí seznam, kde jsou na místě vrcholů čísla jejich SSK. O(n*m + n+ m)

public int[,] **FW** – Implementace Floyd-Warshall algoritmu se zlepšenou prostorovou složitostí na O(n^2). Časová složitost O(n^3).

public int[,] **Jarnik** – Implementace Jarníkova algoritmu s haldou. Minimální kostra je vrácena jako matice sousednosti. O(m*log n)

public string **TopologicalSort** – Díky podrobnému DFS zkontroluje, jestli neobsahuje cyklus a vrátí obráceně přepsaný string s vrcholy tak, jak se uzavíraly při průchodu do hloubky. O(m + n)