Uniwersytet Jagielloński w Krakowie
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Łukasz Kostrzewa
Nr albumu: 1080514

Wizualizacja, edycja i przetwarzanie grafów on-line

Praca magisterska
na kierunku Informatyka stosowana

Praca wykonana pod kierunkiem
dr hab. Barbary Strug
Zakład Projektowania i Grafiki Komputerowej

Kraków 2017

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Kraków, dnia Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Kraków, dnia Podpis kierującego pracą

Spis treści

Wstęp

„This question is so banal, but seemed to me worthy of attention *in that geometry, nor algebra, nor even the art of counting was sufficient to solve it*[1] *"*. Tak w 1736 roku pisał Leonhard Euler w liście do Giovanniego Marinoniego, włoskiego matematyka i inżyniera, o jednym z pierwszych problemów w teorii grafów – problemie mostów królewskich. Banalny, ale warty uwagi.

W dzisiejszych czasach teoria grafów rozwiązuje wiele nietrywialnych problemów, a część z nich nadal pozostaje otwarta. Grafy znalazły praktyczne zastosowanie w wielu różnorodnych dziedzinach nauki, takich jak informatyka, ekonomia, socjologia, jak również chemia, lingwistyka, geografia czy nawet architektura. Bez wątpienia teoria grafów jest dziedziną matematyki i informatyki, która zasługuje na uwagę, co postaram się w niniejszej pracy przedstawić.

Głównym celem mojej pracy jest stworzenie aplikacji służącej do wizualizacji i edycji grafów w przeglądarce. W przeciągu kilku ostatnich lat mogliśmy zaobserwować gwałtowny wzrost znaczenia aplikacji internetowych. Co dziwne, na dzień dzisiejszy w sieci praktycznie nie ma rozwiązania, które pozwalałoby wczytać graf, wyświetlić, w łatwy sposób przetworzyć, a następnie wyeksportować do znanego formatu. Praca ta jest odpowiedzią na ów deficyt.

W pracy dokonam również przeglądu i analizy bibliotek JavaScript oraz technologii służących do wizualizacji grafów w przeglądarce.

[1] Cytat zaczerpnięty z [6], wyróżnienie własne.

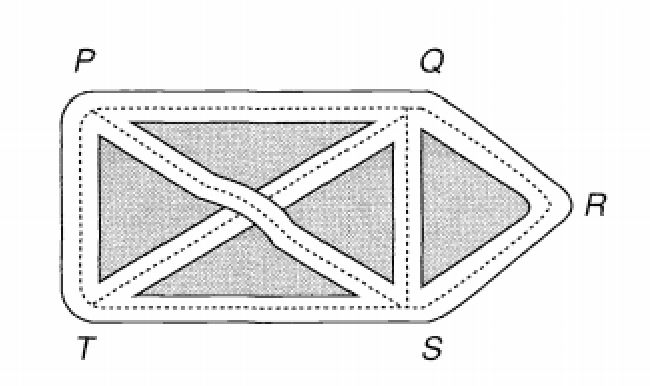
Rozdział 1

Podstawy teoretyczne

W tym rozdziale omówię czym są grafy – na początku przedstawię intuicyjne wyjaśnienie, po czym podam formalną definicję. Pojawią się także definicje pojęć związanych z grafami, które będą występować w kolejnych rozdziałach pracy. Następnie przytoczę przykłady znanych grafów, takich jak graf pełny czy graf cykliczny. W ostatniej sekcji przedstawię zastosowania grafów.

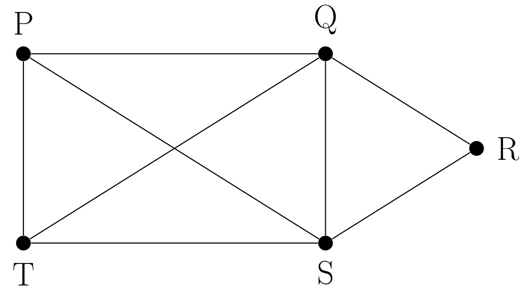
1.1 Czym są grafy?

Poniższy rysunek 1.1 przedstawia fragment mapy drogowej.



Rysunek 1.1: Fragment mapy drogowej [9, s. 11]

Możemy ją w uproszczeniu przedstawić za pomocą punktów i odcinków, tak jak na rysunku 1.2.

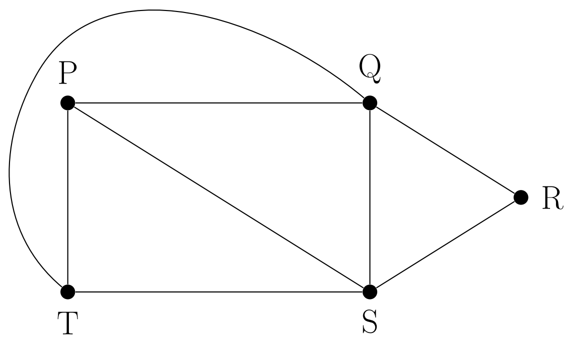


Rysunek 1.2: Uproszczone przedstawienie fragmentu mapy drogowej

Punkty P, Q, R, S, T nazywamy **wierzchołkami**, odcinki nazywamy **krawędziami**, a cały wykres – **grafem**. Punkt przecięcia odcinków PS z QT nie jest wierzchołkiem (nie odpowiada on skrzyżowaniu ulic).

Ten sam graf może modelować również inną sytuację. Przykładowo wierzchołkami mogą być osoby, a krawędź może oznaczać relację znajomości. Tak więc osoba P zna osobę T, ale nie zna osoby R (choć mają wspólnych znajomych Q i S).

Tę samą sytuację obrazuje także graf przedstawiony na rysunku 1.3, w którym pozbyliśmy się przecięcia odcinków PS i QT. Jednak nadal graf ten dostarcza informacji o tym, czy dane osoby znają się lub czy pomiędzy dwoma skrzyżowaniami istnieje bezpośrednia droga. Informacje, które tracimy dotyczą własności „metrycznych” (takich jak długość[1] czy kształt drogi).



Rysunek 1.3: Fragment mapy drogowej lub relacja znajomości narysowana bez „przecięć”

Tak więc graf przedstawia pewien zbiór punktów i dostarcza informacji, które z nich są ze sobą połączone. Oznacza to, że dwa grafy, które modelują

[1]Choć tę informację możemy zachować, przypisując do każdej krawędzi **wagę**.

tę samą sytuację, tak jak na rysunkach 1.2 oraz 1.3, są uznawane za identyczne [9, s. 12] (niezależnie od sposobu w jaki narysujemy krawędzie oraz jak rozmieścimy wierzchołki).

W tym miejscu warto również wspomnieć, że podobnie jak pomiędzy dwoma skrzyżowaniami lub miastami może istnieć wiele dróg, tak i w grafach dwa wierzchołki może łączyć więcej niż jedna krawędź (tzw. **krawędzie wielokrotne**). Inną analogią są drogi jednokierunkowe – ich grafowym odpowiednikiem są **krawędzie skierowane**.

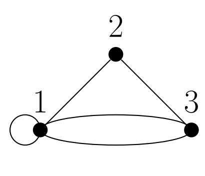
Po tej wstępnej sekcji, która miała na celu zarysować czym są grafy, nastąpi sekcja zawierająca formalne definicje oraz pojawi się więcej pojęć związanych z teorią grafów.

1.2 Definicje

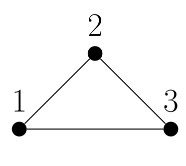
Graf ogólny, graf prosty

Graf (**graf ogólny**, **multigraf**) G jest parą $(V(G), E(G))$, gdzie $V(G)$ jest skończonym, niepustym zbiorem elementów zwanych **wierzchołkami**, a $E(G)$ jest skończoną rodziną nieuporządkowanych par elementów zbioru $V(G)$ zwanych **krawędziami** [9, s. 20] (tj. $E(G) \subseteq \{\{u, v\} : u, v \in V(G)\}$). Zbiór $V(G)$ nazywamy zbiorem wierzchołków, a rodzinę $E(G)$ – **rodziną krawędzi** grafu G; gdy nie ma możliwości pomyłki często są skracane do odpowiednio V oraz E. (Niektóre definicje nie wymagają, aby zbiory V oraz E były skończone [11, s. 143], ale ponieważ w naszych zastosowaniach będziemy mieli do czynienia ze zbiorami skończonymi, przyjmiemy, że zbiory te są skończone). Wierzchołki $u, v \in V$ są **połączone** krawędzią $\{u, v\}$ (lub krócej uv), gdy $\{u, v\} \in E$.

Zauważmy, że taka definicja dopuszcza sytuację, w której dwa wierzchołki są połączone więcej niż jedną krawędzią (tzw. **krawędź wielokrotna**) oraz gdy wierzchołek jest połączony z samym sobą (tzw. **pętla**). Graf, który nie posiada krawędzi wielokrotnych oraz pętli nazywamy **grafem prostym** [9, s. 19].



Rysunek 1.4: Przykład grafu ogólnego



Rysunek 1.5: Przykład grafu prostego

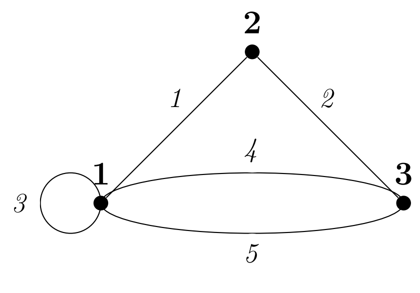
Sąsiedztwo

Wierzchołki $u, v \in V$ są **sąsiednie** jeśli istnieje krawędź uv (wówczas wierzchołki u i v są **incydentne** z tą krawędzią). Dwie krawędzie są **sąsiednie**, jeśli są incydentne z tym samym wierzchołkiem.

Stopień wierzchołka $v \in V$ (oznaczany jako $deg(v)$) jest liczbą krawędzi incydentnych z v. **Wierzchołek izolowany** to wierzchołek stopnia 0, a **wierzchołek końcowy** – stopnia 1.

Istnieją dwie standardowe reprezentacje grafów w pamięci komputera: jako **listy sąsiedztwa** lub jako **macierze sąsiedztwa** [4, s. 29, 13, s. 600]. Pierwsza z nich polega na zapamiętaniu dla każdego wierzchołka listy wierzchołków z nim sąsiadujących. Druga zakłada, że wierzchołki są ponumerowane liczbami ze zbioru $\{1, 2, \ldots, n\}$ (gdzie n oznacza moc zbioru V) i opiera się na stworzeniu macierzy wymiaru $n \times n$, której wyraz o indeksach i, j jest równy liczbie krawędzi łączących wierzchołek o numerze i z wierzchołkiem o numerze j.

Innym sposobem reprezentacji grafu za pomocą macierzy jest **macierz incydencji**. Jeśli krawędzie oznakujemy liczbami ze zbioru $\{1, 2, \ldots, m\}$ (gdzie m moc zbioru E), to jest to macierz o rozmiarze $n \times m$, której wyraz o indeksach i, j jest równy 1, jeśli wierzchołek z numerem i jest incydentny z krawędzią j, i jest równy 0 w przeciwnym przypadku [11, s. 27].



Rysunek 1.6

Macierz sąsiedztwa A i macierz incydencji M dla grafu z rysunku 1.6:

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}, \qquad M = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Podgraf

Podgraf $G' = (V', E')$ grafu $G = (V, E)$ to graf, którego wszystkie wierzchołki należą do V, a krawędzie należą do E (tj. $V' \subseteq V$ oraz $E' \subseteq E$).Jeśli

$V' = V$, to podgraf G' nazywany jest **podgrafem rozpinającym** [4, s. 229].

Podgraf jest **indukowany** przez V', jeśli zawiera wszystkie krawędzie z grafu G o końcach w wierzchołkach z V' (tj. $E' = \{(u, v) \in E : u, v \in V'\}$) [13, s. 1195].

Trasa, ścieżka, droga, cykl

Trasa w grafie G to skończony ciąg krawędzi $v_0v_1, v_1v_2 \ldots v_{m-1}v_m$ (zapisywany również w postaci $v_0 \to v_1 \to \ldots \to v_m$), w którym każde dwie kolejne krawędzie są albo sąsiednie, albo identyczne [9, s. 41]. Trasa wyznacza ciąg wierzchołków $v_0, v_1, \ldots, v_m$ – pierwszy z nich nazywamy **wierzchołkiem początkowym**, a ostatni **wierzchołkiem końcowym**. Liczba krawędzi na trasie to **długość trasy**.

Ścieżka to trasa, w której wszystkie krawędzie są różne. Jeśli również wszystkie wierzchołki $v_0, v_1, \ldots, v_m$ są różne (dopuszczając jedynie możliwość, aby wierzchołek początkowy był równy wierzchołkowi końcowemu), to ścieżka nazywana jest **drogą**. Droga (lub ścieżka) jest **zamknięta**, jeśli $v_0 = v_m$; ścieżka zamknięta, która posiada co najmniej jedną krawędź to **cykl**; droga zamknięta, która posiada co najmniej jedną krawędź to **cykl prosty**.

Jeśli istnieje ścieżka z u do v, to mówimy, że v jest **osiągalny** z u.

Cykl Eulera

Cykl Eulera to taki cykl w grafie, który przechodzi przez każdą jego krawędź dokładnie jeden raz. Graf spójny, który posiada cykl Eulera nazywany jest **grafem eulerowskim**.

TWIERDZENIE 1 (Euler, 1736). *Graf spójny G jest grafem eulerowskim wtedy i tylko wtedy, gdy stopień każdego wierzchołka grafu G jest liczbą parzystą.*

TWIERDZENIE 2. *Skierowany graf spójny G jest eulerowski wtedy i tylko wtedy, gdy każdy wierzchołek grafu G ma tyle samo krawędzi wchodzących i wychodzących.*

Cykl Hamiltona

Cykl Hamiltona to taki cykl w grafie, który przechodzi przez każdy jego wierzchołek dokładnie jeden raz. Graf spójny posiadający cykl Hamiltona nazywany jest **grafem hamiltonowskim**.

W przeciwieństwie do problemu stwierdzenia czy graf jest eulerowski, nie jest znany warunek konieczny i wystarczający na to, aby graf był hamiltonowksi. Problem stwierdzania czy graf jest hamiltonowski należy do jednych z najważniejszych nierozwiązanych problemów teorii grafów [9, s. 54].

Istnieją twierdzenia (np. Twierdzenie 3, 4), które na podstawie cech grafu pozwalają stwierdzić, czy graf jest hamiltonowski. Mają one postać: „jeśli graf ma wystarczająco dużo krawędzi, to ma cykl Hamiltona” [9, s. 54]. Są to jednak implikacje jednostronne – istnieją grafy hamiltonowskie, które nie spełniają poprzedników tych implikacji.

TWIERDZENIE 3 (Dirac, 1952). *Jeśli w grafie prostym G, który ma n wierzchołków (gdzie $n \geqslant 3$)*

$$deg(v) \geqslant \frac{n}{2}$$

dla każdego wierzchołka v, to graf G jest hamiltonowski.

TWIERDZENIE 4 (Orego, 1960). *Jeśli graf prosty G ma n wierzchołków (gdzie $n \geqslant 3$), oraz*

$$deg(v) + deg(w) \geqslant n$$

dla każdej pary wierzchołków niesąsiednich v i w, to graf G jest hamiltonowski.

Graf skierowany

Graf skierowany, **digraf** (ang. *directed graph*) G to para $(V(G),\ E(G))$, gdzie $V(G)$ niepusty, skończony zbiór elementów zwanych **wierzchołkami**, a $E(G)$ skończony zbiór par *uporządkowanych* elementów ze zbioru $V(G)$ zwanych **krawędziami** (lub **łukami** [9, s. 135]). Digraf G jest **digrafem prostym**, jeśli wszystkie krawędzie są różne oraz jeśli nie posiada pętli.

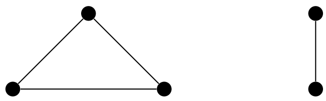
Jeśli $e = (v, w) \in E(G)$, to v nazywamy **początkiem krawędzi** e, a w – **końcem krawędzi** e.

Definicje z poprzedniej podsekcji w naturalny sposób uogólniają się na przypadek digrafów [9, s. 136].

Spójność

Załóżmy, że mamy dwa grafy $G_1 = (V_1, E_1)$ oraz $G_2 = (V_2, E_2)$, gdzie $V_1 \cap V_2 = \emptyset$. Wówczas **sumą** tych grafów $G_1 \cup G_2$ jest graf $G = (V_1 \cup V_2, E_1 \cup E_2)$. Graf nazywamy **spójnym**, jeśli nie można przedstawić go w postaci sumy dwóch grafów, w przeciwnym razie graf jest **niespójny** [9, s. 22]. Każdy graf niespójny G możemy przedstawić jako sumę grafów spójnych, nazywanych

spójnymi składowymi grafu G (rysunek 1.7 przedstawia graf posiadający dwie spójne składowe).



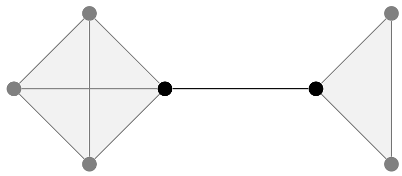
Rysunek 1.7: Przykład grafu mającego dwie spójne składowe

Niektórzy autorzy [11, s. 342] podają alternatywną, równoważną [9, s. 42] definicję grafu spójnego – jest to graf, w którym każda para różnych wierzchołków jest połączona drogą.

Graf skierowany G jest **silnie spójny**, jeśli dla dowolnych dwóch wierzchołków v i w istnieje droga z v do w. Każdy digraf silnie spójny jest spójny, ale nie wszystkie digrafy spójne są silnie spójne.

Dwuspójną składową grafu G nazywamy maksymalny podzbiór krawędzi, taki że każde dwie krawędzie z tego zbioru leżą na wspólnym cyklu prostym [13, s. 634]. W dwuspójnej składowej pomiędzy każdą parą wierzchołków istnieją dwie rozłączne krawędziowo drogi. Wierzchołki należące do co najmniej dwóch różnych dwuspójnych składowych nazywamy **wierzchołkami rozdzielającymi** (lub **punktami artykulacji** [13, s. 633]). Usunięcie wierzchołka rozdzielającego „rozspójnia” graf. Krawędzie, które nie należą do żadnego cyklu prostego nazywamy **mostami**. Ich usunięcie również „rozspójnia” graf.

Graf, który posiada tylko jedną dwuspójną składową nazywamy **grafem dwuspójnym** [4, s. 232].

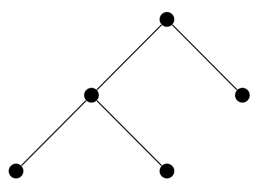


Rysunek 1.8: Przykład grafu zawierającego trzy dwuspójne składowe – punkty artykulacji i mosty zostały zaznaczone na czarno.

Drzewo, las, drzewo rozpinające

Drzewo to graf spójny nie posiadający cykli. **Las** to graf, którego spójnymi składowymi są drzewa.

Drzewem rozpinającym graf G nazywamy podgraf rozpinający G, który nie zawiera cykli [1, s. 10].

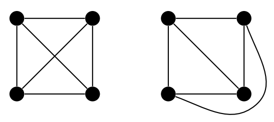


Rysunek 1.9: Przykład grafu będącego drzewem

Planarność

Graf planarny to graf, który można narysować na płaszczyźnie tak, aby żadne dwie krzywe obrazujące krawędzie nie przecinały się ze sobą. Odwzorowanie grafu na płaszczyźnie o tej własności nazywane jest **rysunkiem płaskim** (lub **grafem płaskim**) [9, s. 82].

Na rysunku 1.10 jest narysowany na dwa sposoby graf pełny K_4 (opisany w następnej sekcji 1.3) – drugi sposób jest przykładem rysunku płaskiego.



Rysunek 1.10: K_4 – przykład grafu planarnego

Kolorowanie

Kolorowanie wierzchołków

Jeśli graf G nie ma pętli oraz jeśli każdemu wierzchołkowi możemy przypisać jeden z k kolorów w taki sposób, aby sąsiednie wierzchołki miały różne kolory, to graf G jest grafem k**-kolorowalnym**. Jeśli graf G jest k-kolorowalny, ale nie jest $(k-1)$-kolorowalny, to mówimy, że jest k**-chromatyczny** lub że jego **liczba chromatyczna** wynosi k.

Kolorowanie krawędzi

Jeśli krawędzie grafu G możemy pokolorować k kolorami w taki sposób, aby żadne dwie sąsiednie krawędzie nie miały tego samego koloru, to graf G jest k**-kolorowalny krawędziowo**. Jeśli graf G jest k-kolorowalny krawędziowo, ale nie jest $(k-1)$-kolorowalny krawędziowo, to mówimy, że G ma **indeks chromatyczny** równy k.

Skojarzenia

Skojarzeniem w grafie $G = (V, E)$ nazywamy taki podzbiór krawędzi $M \subseteq E$, w którym żadne dwie krawędzie nie są ze sobą sąsiadujące. Wierzchołek $v \in V$ jest **skojarzony** w podzbiorze M, jeśli pewna krawędź z M jest incydentna z v. Skojarzenie jest **doskonałe** (lub **całkowite**), jeśli każdy wierzchołek jest skojarzony.

Poniższe twierdzenie podaje warunek konieczny i wystarczający, aby dany graf dwudzielny posiadał skojarzenie doskonałe.

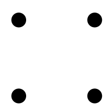
TWIERDZENIE 5 (Hall, 1935). *W grafie dwudzielnym $G = (V_1 \cup V_2, E)$, w którym $|V_1| = |V_2|$ istnieje skojarzenie doskonałe wtedy i tylko wtedy, gdy dla każdego podzbioru $K \subseteq V_1$ zachodzi*

$$|K| \leqslant |N(K)|, \quad \textit{gdzie } N(K) = \{w \in V_2 : \exists_{k \in K}\{k, w\} \in E\}$$

1.3 Przykłady grafów

Graf pusty

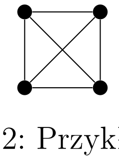
Graf pusty to graf, którego zbiór krawędzi jest zbiorem pustym. Każdy wierzchołek grafu pustego jest wierzchołkiem izolowanym. *„Grafy puste nie są zbyt interesujące”* [9, s. 30].



Rysunek 1.11: Przykład grafu pustego mającego cztery wierzchołki

Graf pełny

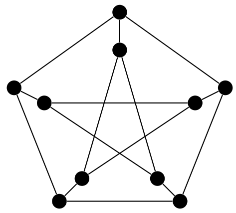
Graf pełny to graf prosty, którego każda para różnych wierzchołków jest połączona krawędzią. Graf pełny mający n wierzchołków (oraz $\frac{n(n-1)}{2}$ krawędzi) oznacza się symbolem K_n.



Rysunek 1.12: Przykład grafu K_4

Graf regularny

Graf regularny to graf, w którym każdy wierzchołek ma ten sam stopień. Jeśli każdy wierzchołek ma stopień r, to graf nazywa się **grafem regularnym stopnia** r (lub **grafem** r**-regularnym**) [9, s. 31]. Przykładem grafu regularnego stopnia 3 jest **graf Petersena** przedstawiony na rysunku 1.13. Każdy graf pusty jest grafem 0-regularnym, a graf pełny K_n jest grafem regularnym stopnia $n-1$.

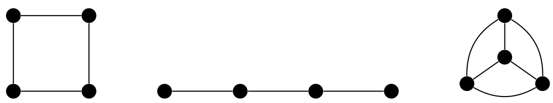


Rysunek 1.13: Graf Petersena

Graf cykliczny, graf liniowy, koło

Graf cykliczny to spójny graf regularny stopnia 2. Graf cykliczny mający n wierzchołków oznacza się symbolem C_n. **Graf liniowy** o n wierzchołkach (oznaczany symbolem P_n) to graf powstały przez usunięcie jednej krawędzi z C_n.

Graf powstający z grafu C_{n-1} poprzez dodanie dodatkowego wierzchołka i połączenie go ze wszystkimi pozostałymi nazywany jest **kołem** i oznaczany jest symbolem W_n.

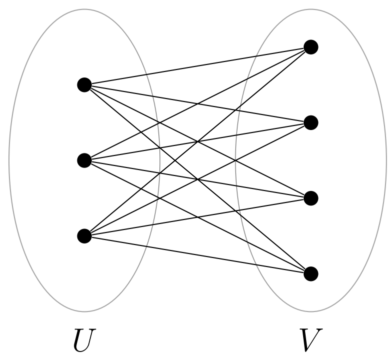


Rysunek 1.14: Przykład grafu C_4, P_4 i W_4

Grafy dwudzielne

Graf dwudzielny to graf, którego zbiór wierzchołków może być podzielony na dwa rozłączne zbiory U i V w taki sposób, że krawędzie nie łączą wierzchołków z tego samego zbioru.

Ponadto jeśli każdy wierzchołek ze zbioru U jest połączony dokładnie jedną krawędzią z każdym wierzchołkiem ze zbioru V, to taki graf jest nazywany **pełnym grafem dwudzielnym**. Jeśli moc zbioru U wynosi r, a moc zbioru V wynosi s, to taki graf jest oznaczany symbolem $K_{r,s}$ (ma on $r + s$ wierzchołków oraz rs krawędzi).



Rysunek 1.15: Przykład grafu $K_{3,4}$

1.4 Zastosowania grafów

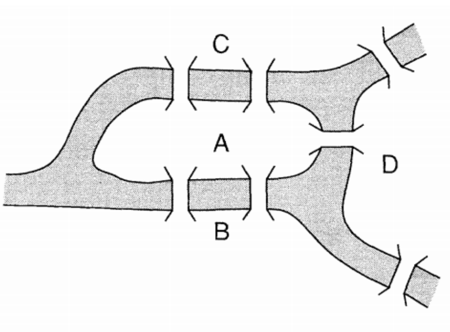
W dzisiejszych czasach grafy mają szerokie zastosowanie w wielu różnorodnych dziedzinach, takich jak informatyka, ekonomia, socjologia, chemia, lingwistyka, logistyka czy telekomunikacja. W tej sekcji przedstawię jedynie kilka przykładowych problemów oraz jak za pomocą teorii grafów mogą one zostać rozwiązane (nie wdając się w szczegóły algorytmów, które są ogólnodostępne [13, 2, 4]).

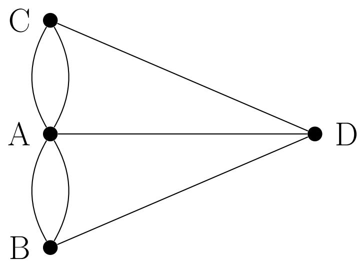
Mosty Królewieckie, cykl Eulera

Jednym z pierwszych zastosowań teorii grafów było rozwiązanie zagadnienia mostów królewieckich. Problem został rozwiązany przez Leonarda Eulera w XVIII wieku [9, s. 48].

Przez Królewiec przepływała rzeka Pregoła, w której rozwidleniach znajdowały się dwie wyspy. Ponad rzeką wybudowano siedem mostów, tak jak jest to pokazane na rysunku 1.16. Pytanie brzmiało: czy można przejść przez każdy most dokładnie jeden raz i powrócić do punktu wyjścia.

Jest to równoważne z pytaniem, czy graf pokazany na rysunku 1.17 posiada cykl Eulera (skąd pochodzi nazwa ów cyklu).





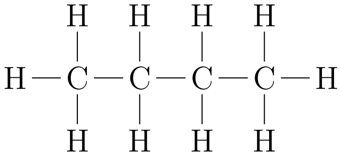
Rysunek 1.16: Schemat mostów w Królewcu z rzeką Pregołą [9]

Rysunek 1.17: Graf mostów w Królewcu

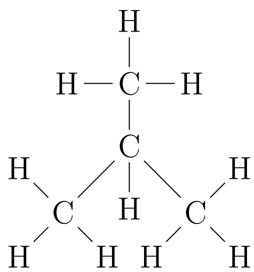
Zliczanie cząsteczek chemicznych

Zliczanie cząsteczek chemicznych należy do jednych z najwcześniejszych przykładów użycia drzew. [9, s. 76]. Wielki brytyjski matematyk Arthur Cayley był pierwszym, który dostrzegł związek pomiędzy wzorami strukturalnymi w chemii organicznej i grafami. [10, s. 59]. Jako pierwszy wymyślił również określenie „drzewo" [10, s. 60].

Cayley miał zamiar znaleźć sposób na obliczanie ile jest różnych izomerów *alkanów*, których ogólny wzór sumaryczny ma postać C_nH_{2n+2}. Metodą budowania drzewa od jego centrum (centrów) udało mu się prawidłowo obliczyć ilość alkanów posiadających do jedenastu atomów węgla [5, s. 180].



Rysunek 1.18: Wzór strukturalny n-butanu



Rysunek 1.19: Wzór strukturalny 2-metylopropanu

Poniższa tabela przedstawia liczbę różnych alkanów C_nH_{2n+2} posiadających n atomów węgla, dla $n = 1, \ldots, 11$.

n	1	2	3	4	5	6	7	8	9	10	11
liczba alkanów	1	1	1	2	3	5	9	18	35	75	159

Rezultaty Cayleya wykorzystali i rozwinęli w swoich pracach inni, m.in. węgierski matematyk G. Pólya. W wyniku tych prac za pomocą metod teorii grafów zliczono wiele innych typów cząsteczek chemicznych.

Zagadnienie najkrótszej ścieżki

Każdej krawędzi e grafu G możemy przypisać pewną nieujemną liczbę $w(e)$ (zwaną **wagą** tej krawędzi). Wówczas taki graf jest nazywany **grafem z wagami**.

Grafy z wagami często występują w zastosowaniach teorii grafów [2]. Na przykład, w grafach modelujących relację znajomości waga może wskazywać na to, jak dobrze dane osoby znają się, a w grafach modelujących sieć połączeń komunikacyjnych – odległość pomiędzy dwoma punktami albo koszt wybudowania lub utrzymania takiego połączenia.

Zadanie polega na znalezieniu ścieżki pomiędzy dwoma wybranymi punktami (lub ich większej ilości), której suma wag krawędzi jest najmniejsza. Do rozwiązania tego problemu może posłużyć algorytm Dijkstry działający w czasie $O(E\log(V))$. Dla grafów planarnych istnieje szybszy algorytm, który działa w czasie liniowym [3].

Warto zwrócić w tym miejscu uwagę, że również trasowanie w sieci Internet i wybór odpowiedniej drogi dla pakietów zawdzięczamy teorii grafów (np. protokół OSPF korzysta z algorytmu Dijkstry).

Problem chińskiego listonosza

W swojej pracy listonosz pobiera listy z poczty, dostarcza je, po czym wraca do budynku poczty. Musi przejść każdą ulicę przynajmniej jeden raz. Zadanie polega na znalezieniu najkrótszej drogi dla listonosza. Zagadnienie to jest znane pod nazwą *problemu chińskiego listonosza*, ponieważ było po raz pierwszy rozpatrywane przez chińskiego matematyka Kuana (1962) [2, s. 62].

Dla grafów eulerowskich problem sprowadza się do znalezienia cyklu Eulera (ponieważ taki cykl przechodzi przez każdą krawędź dokładnie raz). Problem ten możemy łatwo rozwiązać w takim przypadku, np. stosując algorytm Fleury'ego.

Problem komiwojażera

Podróżujący sprzedawca chce odwiedzić daną listę miast i powrócić do punktu początkowego. Mając dane czasy podróży pomiędzy miastami, w jaki sposób sprzedawca powinien zaplanować podróż, żeby odwiedzić wszystkie dokładnie raz w jak najkrótszym czasie? Zagadnienie to jest znane pod nazwą *problemu komiwojażera* i jest równoważne ze znalezieniem takiego cyklu Hamiltona w danym grafie ważonym, w którym suma wag jest najmniejsza.

W przeciwieństwie do problemu chińskiego listonosza, nie jest znany efektywny[2] algorytm rozwiązujący problem komiwojażera. Dlatego często pożądane jest znalezienie odpowiednio dobrego (ale niekoniecznie najlepszego) rozwiązania [2, s. 65].

Problem najkrótszych połączeń

Pomiędzy miastami ma być wybudowana sieć połączeń kolejowych. Dane są koszty c_{ij} wybudowania połączenia pomiędzy miastami v_i i v_j. Zadanie polega na zaprojektowaniu sieci połączeń tak, aby zminimalizować koszt konstrukcji całej sieci.

Problem sprowadza się do obliczenia minimalnego drzewa rozpinającego na danym grafie, co możemy uzyskać stosując np. algorytm Kruskala.

Problem stworzenia niezawodnej sieci komunikacyjnej

Graf może reprezentować sieć komunikacyjną, którego wierzchołki to stacje komunikacyjne (lub którego krawędzie to połączenia komunikacyjne). Jaka jest minimalna liczba k stacji (lub połączeń), których awaria zaburzy komunikację w tej sieci (tj. rozspójni graf). Im większa ta liczba tym bardziej niezawodna jest sieć.

Dla $k = 1$ problem redukuje się do problemu najkrótszych połączeń. Dla $k > 1$ problem jest nierozwiązany i jest uważany za trudny (jednak dla grafów pełnych istnieje proste rozwiązanie) [2, s. 48].

Problem przydziału personelu

W pewnej firmie n pracowników $X_1, X_2, \ldots, X_n$ jest dostępnych do wykonania n zadań $Y_1, Y_2, \ldots, Y_n$. Każdy pracownik jest wykwalifikowany do wykonania jednego lub więcej z tych zadań. Czy można przypisać każdego pracownika do jednego zadania, do którego jest wykwalifikowany?

[2] tj. działający w czasie wielomianowym

Możemy utworzyć graf dwudzielny G z podziałem wierzchołków na rozłączne zbiory X i Y, gdzie $X = \{x_1, x_2, \ldots, x_n\}$ oraz $Y = \{y_1, y_2, \ldots, y_n\}$, w którym wierzchołek x_i jest połączony krawędzią z wierzchołkiem y_i, gdy pracownik X_i jest zdolny wykonać zadanie Y_j. Problem sprowadza się do sprawdzenia czy dany graf G posiada skojarzenie doskonałe (co możemy stwierdzić na mocy twierdzenia 5).

Problem rozkładu zadań

W szkole jest m nauczycieli $X_1, X_2, \ldots, X_m$ oraz n klas $Y_1, Y_2, \ldots, Y_n$. Każdy nauczyciel X_i powinien uczyć klasę Y_j przez p_{ij} godzin lekcyjnych. Zadanie polega na takim rozplanowaniu harmonogramu zajęć, aby zajęcia skończyły się jak najwcześniej.

Możemy utworzyć graf dwudzielny G z podziałem wierzchołków na rozłączne zbiory X i Y, gdzie $X = \{x_1, x_2, \ldots, x_m\}$ oraz $Y = \{y_1, y_2, \ldots, y_n\}$, w którym wierzchołek x_i jest połączony p_{ij} krawędziami z wierzchołkiem y_i. W danym momencie nauczyciel może uczyć co najwyżej jedną klasę oraz dana klasa może być uczona przez co najwyżej jednego nauczyciela.

Problem rozkładu zadań można rozwiązać stosując kolorowanie krawędzi – indeks chromatyczny odpowiada minimalnej sumarycznej liczbie godzin lekcyjnych, po których wszystkie klasy odbędą wymaganą ilość poszczególnych godzin lekcyjnych. W grafie dwudzielnym indeks chromatyczny jest równy maksymalnemu stopniowi wierzchołka [2, s. 93].

Problem magazynowania

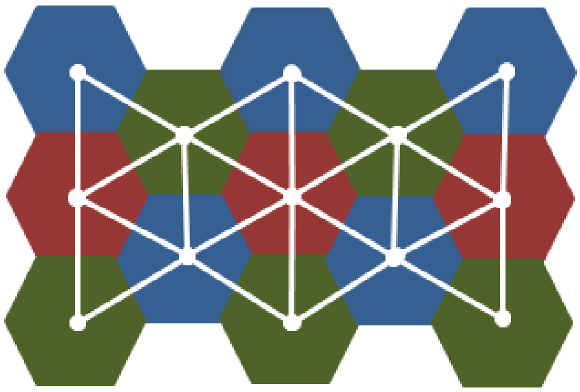
Firma produkuje n chemikaliów $C_1, C_2 \ldots, C_n$. Pewne pary tych chemikaliów nie są kompatybilne i mogą powodować eksplozje w przypadku kontaktu. Jako środek zapobiegawczy firma chce zrobić podzielić magazyny na przedziały i trzymać niekompatybilne chemikalia w osobnych przedziałach. Jaka jest minimalna liczba przedziałów, które powinny być utworzone?

Możemy utworzyć graf G ze zbiorem wierzchołków $\{v_1, v_2, \ldots, v_n\}$, w którym dwa wierzchołki v_i oraz v_j są połączone krawędzią, gdy chemikalia C_i oraz C_j nie są kompatybilne. Wówczas łatwo zauważyć, że minimalna liczba przedziałów, którą należy utworzyć jest równa liczbie chromatycznej grafu G.

Telekomunikacja

W sieci komórkowej obszar podzielony jest na komórki w sposób, który zależy od ukształtowania terenu, zabudowań oraz innych czynników mających wpływ na siłę i jakość odbieranego sygnału. Komórki te mają w przybliżeniu kształt sześciokątów, kwadratów lub okręgów, ale umownie przedstawiane są w postaci sześciokątów. W każdej komórce jest stacja bazowa, do której przypisany jest zakres używanych częstotliwości. Częstotliwości mogą być używane ponownie w innych komórkach pod warunkiem, że ta sama częstotliwość nie jest używana przez dwie sąsiadujące ze sobą komórki (ponieważ to mogłoby powodować zakłócenia sygnału – tzw. *przesłuch*).

W jaki sposób podzielić dostępne częstotliwości, aby spełniony był powyższy warunek? Problem ten możemy sprowadzić do zagadnienia kolorowania wierzchołków, co zostało przedstawione na rysunku 1.20 (inny kolor, oznacza inny zakres częstotliwości). Okazuje się, że cały zakres częstotliwości możemy podzielić na trzy rozłączne podzbiory, aby móc zgodnie z założeniem efektywnie pokryć cały obszar.



Rysunek 1.20: Schemat komórek w sieci GSM [8]

Planarność

Istnieje wiele praktycznych sytuacji, w których ważne jest stwierdzenie czy dany graf jest planarny i jeśli tak, to jak wygląda jego rysunek płaski. Na przykład, dana jest płytka elektroniczna, na której mają być wydrukowane przewody. Czy istnieje taki sposób ich rozmieszczenia, by połączenia nie przecinały się?

Problem ten możemy rozwiązać stosując algorytm skonstruowany przez Demoucrona, Malgrange'a i Pertuiseta (1964) [2, s. 163].

Inne zastosowania

Ponadto grafy znalazły zastosowanie w wielu innych dziedzinach, takich jak:

- systemy rekomendacji,
- wykrywanie oszustw,
- wykrywanie spamu,
- ranking stron w wyszukiwarce,
- drzewa przeszukiwań binarnych,
- bazy danych (B-drzewa),
- sieci przepływowe,
- sieci znajomych.

Rozdział 2

Przegląd istniejących aplikacji

W tym rozdziale przedstawię istniejące aplikacje internetowe [16] i desktopowe służące do tworzenia i wizualizacji grafów. Tabela 2.1 zawiera porównanie funkcjonalności opisywanych darmowych aplikacji internetowych oraz aplikacji *Graphy* – tworzonej w ramach tej pracy.

Głównie skupię się na aplikacjach internetowych ze względu na związek z tematem mojej pracy.

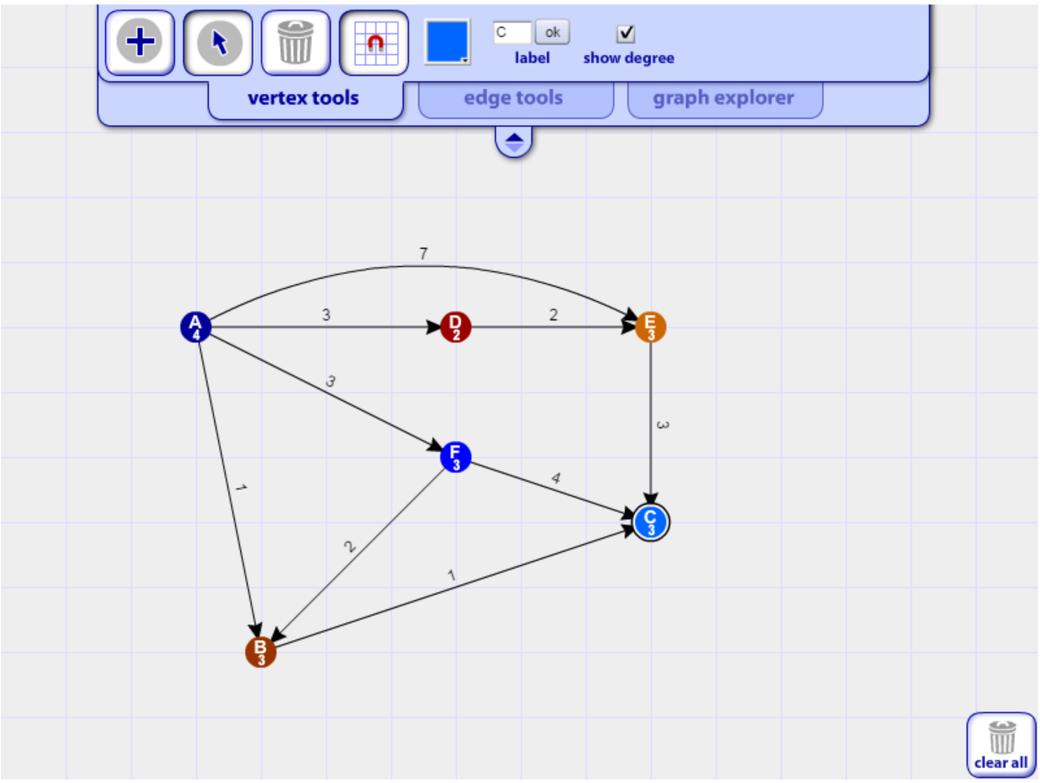
2.1 Aplikacje internetowe

Graph Creator

Adres URL	http://illuminations.nctm.org/Activity.aspx?id=3550
Autor	National Council of Teachers of Mathematics
Licencja	Darmowa

Aplikacja pozwala tworzyć grafy skierowane i nieskierowane. Posiada możliwość kolorowania wierzchołków, wyrównania ich do siatki oraz ustawienia wag na krawędziach i etykiet w wierzchołkach. Ponadto użytkownik może wyświetlić stopnie wierzchołków oraz wyginać krawędzie. Dodatkową funkcjonalnością jest możliwość zaznaczenia kilku wierzchołków na raz.

Graph Creator nie daje możliwości eksportowania i importowania grafów. Nie można również przesuwać widoku ani oddalać oraz przybliżać grafu. Aplikacja posiada ograniczenie liczby wierzchołków – maksymalna dozwolona ilość to 52 wierzchołki.



Rysunek 2.1: Zrzut ekranu z aplikacji Graph Creator

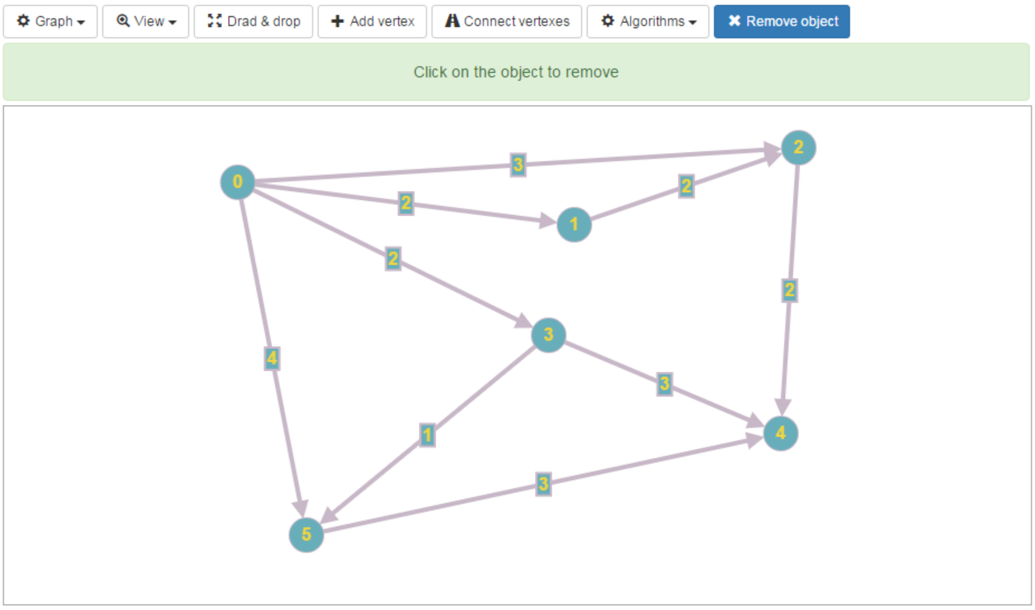
Graph Online

Adres URL	http://graphonline.ru/en/
Autor	Unick-soft
Licencja	Darmowa

Aplikacja również daje możliwość stworzenia grafów zarówno skierowanych jak i nieskierowanych. Podobnie jak poprzednia aplikacja pozwala na zmianę etykiet wierzchołków, nadanie wag krawędziom oraz na wyświetlenie stopnia wierzchołków. Ponadto użytkownik ma możliwość przesuwania widoku oraz jego przybliżania i oddalania. Dodatkowo *Graph Online* pozwala zapisać graf jako macierz sąsiedztwa lub incydencji oraz wczytać graf zapisany w takiej postaci. Użytkownik może również zapisać graf na serwerze – po zapisaniu wyświetlany jest ogólnodostępny adres URL do grafu. Ciekawą funkcjonalnością jest eksport grafu do obrazka (plik `PNG`).

Graph Online posiada możliwość wykonania podstawowych algorytmów na grafie, takich jak: znajdowanie najkrótszej ścieżki pomiędzy dwoma wierzchołkami, znajdowanie cyklu Eulera, znajdowanie spójnych składowych, znajdowanie minimalnego drzewa rozpinającego.

W przeciwieństwie do poprzedniej aplikacji nie mamy możliwości kolorowania wierzchołków, zaznaczania kilku wierzchołków na raz oraz wyginania krawędzi. Maksymalna dozwolona ilość wierzchołków to 299.



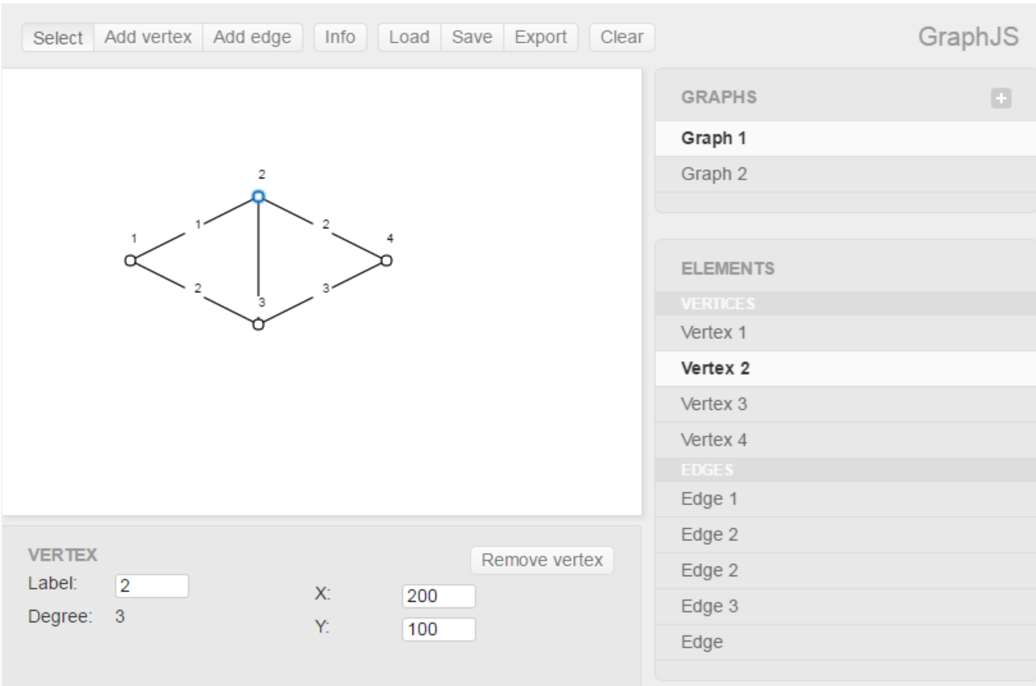
Rysunek 2.2: Zrzut ekranu z aplikacji Graph Online

GraphJS

Adres URL	https://dl.dropboxusercontent.com/u/4189520/GraphJS/graphjs.html
Autor	David Kofoed Wind
Licencja	Darmowa

Aplikacja pozwala na tworzenie grafów nieskierowanych. Podobnie jak w poprzednich aplikacjach możemy nadawać etykiety wierzchołkom i krawędziom. Niespotykaną funkcjonalnością jest możliwość stworzenia kilku grafów i przełączania się pomiędzy nimi oraz możliwość eksportu grafu do formatu LaTeX(pakiet Ti*k*Z). Ponadto użytkownik ma możliwość eksportu do własnego formatu `JSON` oraz importu grafu z tego formatu. Aplikacja posiada funkcjonalność zaznaczania wielu wierzchołków na raz.

W *GraphJS* nie ma możliwości przesuwania widoku oraz przybliżania i oddalania grafu. Nie ma również możliwości kolorowania wierzchołków oraz wyginania krawędzi. Aplikacja zdaje się nie mieć limitu na liczbę wierzchołków – udało się wczytać graf C_{1000} jednakże dodanie kolejnego wierzchołka zajmuje około 10 sekund.



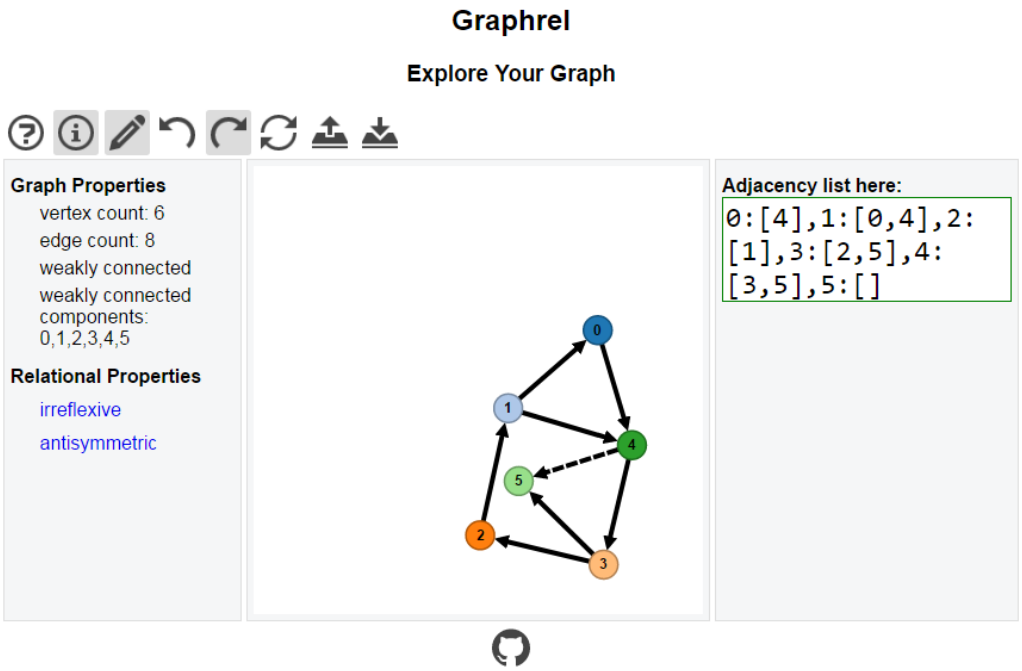
Rysunek 2.3: Zrzut ekranu z aplikacji GraphJS

Graphrel

Adres URL	https://yiboyang.github.io/graphrel/
Autor	Yibo Yang
Licencja	Darmowa

Aplikacja daje możliwość tworzenia grafów skierowanych. W przeciwieństwie do poprzednio opisywanych aplikacji posiada układ kierowany siłą (ang. *force-directed layout*), choć istnieje również opcja samodzielnego rozstawienia wierzchołków – poprzez przytrzymanie klawisza `Ctrl`. Użytkownik może zaimportować graf z formatu stworzonego przez aplikację (tablice list sąsiedztwa dla każdego wierzchołka). Bardzo przydatną i niespotykaną funkcjonalnością jest możliwość cofania oraz ponawiania ostatnich akcji.

W *Graphrel* nie możemy nadawać własnych etykiet na krawędziach ani w wierzchołkach, nie możemy przesuwać widoku ani zmieniać przybliżenia grafu. Nie ma również możliwości wyginania krawędzi, zaznaczania kliku wierzchołków na raz oraz kolorowania wierzchołków. Do aplikacji udało się wczytać graf C_{100}, przy próbie wczytania C_{101} pojawia się informacja o niepoprawnym formacie.

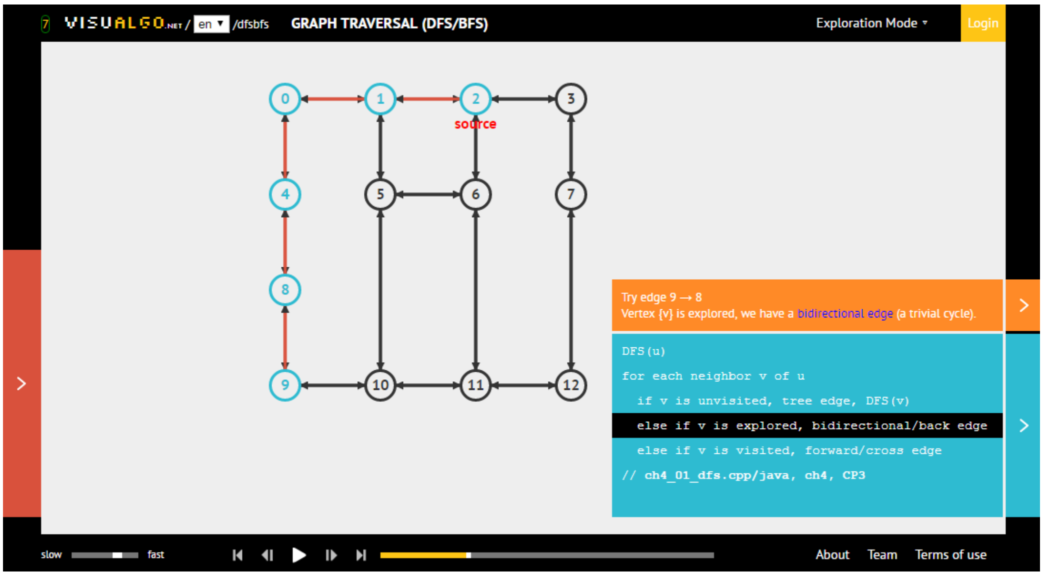


Rysunek 2.4: Zrzut ekranu z aplikacji Graphrel

VisuAlgo

Adres URL	https://visualgo.net/en/
Autor	Dr Steven Halim
Licencja	Darmowa

Aplikacja stworzona przez Dr Stevena Halima z National University of Singapore. Posiada możliwość tworzenia prostych grafów, jednak jej głównym celem nie jest tworzenie grafów, lecz wizualizacja algorytmów przez animację (nie tylko na grafach, ale również na strukturach danych). Użytkownik wraz z przebiegiem algorytmu może obserwować przebieg kodu, może zatrzymać się w dowolnym jego kroku, cofnąć się do kroku poprzedniego albo przejść do następnego.



Rysunek 2.5: Zrzut ekranu z aplikacji VisuAlgo

yEd Live

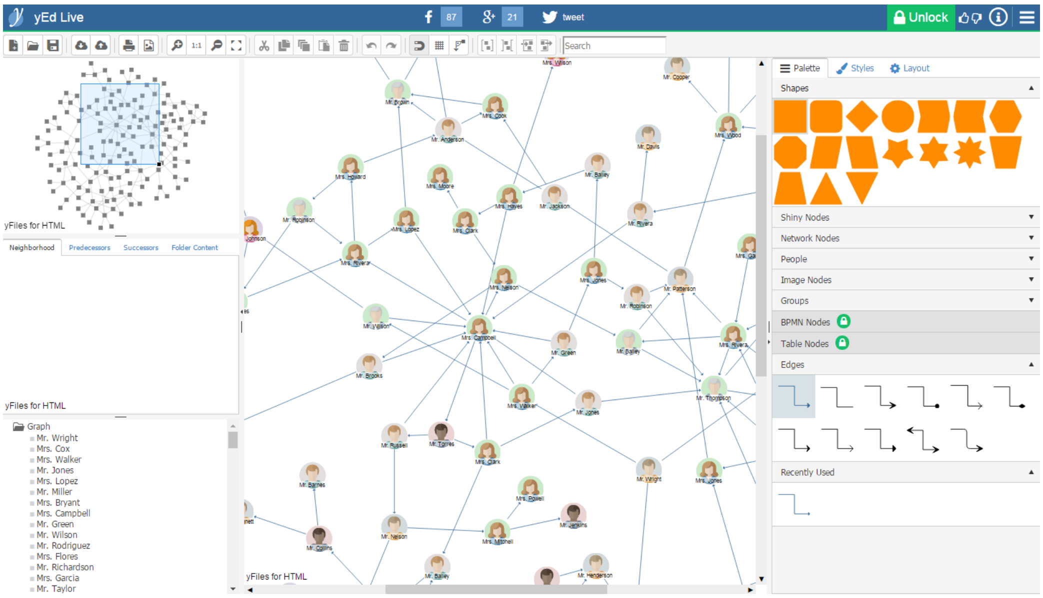
Adres URL	https://www.yworks.com/yed-live/
Autor	yWorks
Licencja	Darmowa i płatna

Internetowa wersja aplikacji yEd stworzona przez firmę yWorks. Pozwala na tworzenie dowolnych grafów (skierowanych, nieskierowanych). W aplikacji istnieje możliwość nadawania etykiet wierzchołkom i krawędziom. Twórcy dostarczyli również możliwość kolorowania wierzchołków, zmiany ich kształtu, a nawet ustawiania obrazków w wierzchołkach. Użytkownik może też dowolnie wyginać krawędzie. Niespotykaną funkcjonalnością jest możliwość grupowania wierzchołków.

Jeśli chodzi o wyświetlanie grafu, to *yEd Live* dostarcza mały podgląd grafu, dzięki któremu możemy przesuwać graf. Istnieje też opcja przybliżania i oddalania grafu. Dodatkową funkcjonalnością jest zmiana układu wierzchołków (m.in. na układ hierarchiczny, ortogonalny czy kołowy) oraz wyszukiwanie wierzchołków po etykiecie.

W *yEd Live* możemy zaimportować graf w formacie `GraphML` (z chmury, dysku lub adresu `URL`). Istnieje również możliwość eksportu do tegoż formatu oraz do pliku graficznego w formacie `PNG`.

Aplikacja dostępna jest w dwóch wersjach: darmowej i płatnej. Darmowa wersja posiada pewne ograniczenia, np. możemy zapisać do formatu `GraphML` graf mający maksymalnie 25 wierzchołków oraz wyeksportować graf do pliku `PNG` mający maksymalnie 50 wierzchołków.



Rysunek 2.6: Zrzut ekranu z aplikacji yEd Live

Linkurious

Adres URL	http://linkurio.us
Autor	Linkurious
Licencja	Płatna

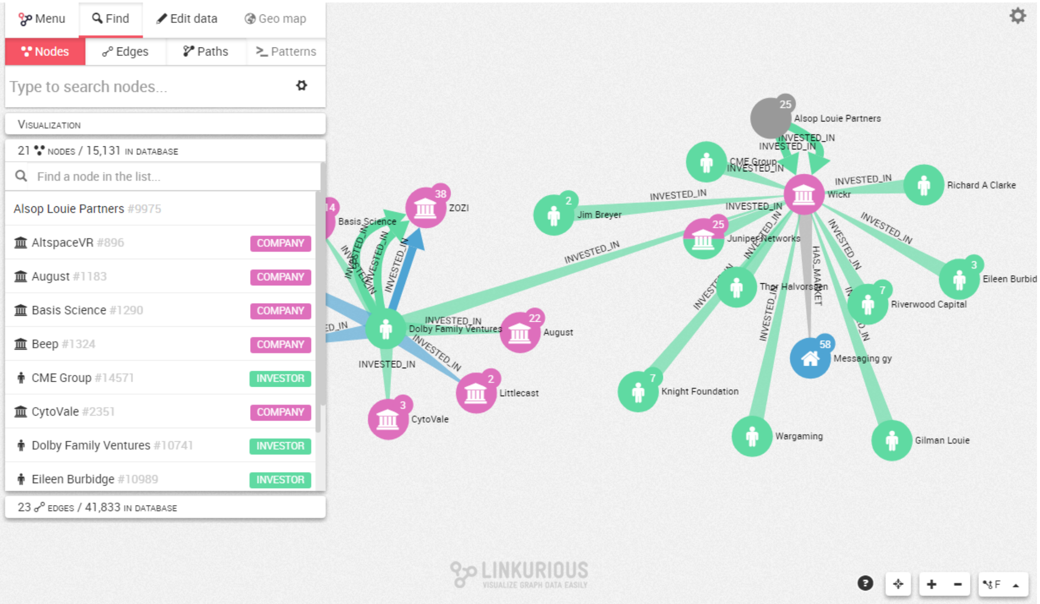
Komercyjna aplikacja internetowa służąca do wizualizacji i badania grafowych baz danych[1]. Jej współzałożycielem jest Dr Sébastien Heymann – współtwórca aplikacji desktopowej Gephi służącej również do wizualizacji i analizowania grafów.

Aplikacja wspiera duże zbiory danych – grafy z miliardami wierzchołków i krawędzi [20]. Pozwala na współpracę wielu użytkowników, m.in. przez możliwość udostępniania grafów czy publikowanie wizualizacji w czasie rzeczywistym.

Posiada kilka opcji układów grafów (kierowanych siłą i hierarchicznych). Dostarcza również możliwość widoku geoprzestrzennego.

[1]Wspierane bazy danych: *Neo4j*, *DataStax Enterprise Graph*, *Titan*, *AllegroGraph* oraz ich języki zapytań: *Cypher*, *Gremlin* i *SPARQL*.

W *Linkurious* istnieje możliwość dostosowania wyglądu elementów, np. wielkość wierzchołków, grubość krawędzi, zmiana ikon w wierzchołkach, tak aby wizualizacje były bogate w informacje. Użytkownik może również tworzyć filtry, by wyświetlić tylko istotne dane oraz tworzyć powiadomienia o podejrzanych połączeniach.



Rysunek 2.7: Zrzut ekranu z aplikacji Linkurious

Tablica 2.1: Porównanie darmowych aplikacji oraz aplikacji *Graphy* – tworzonej w ramach tej pracy

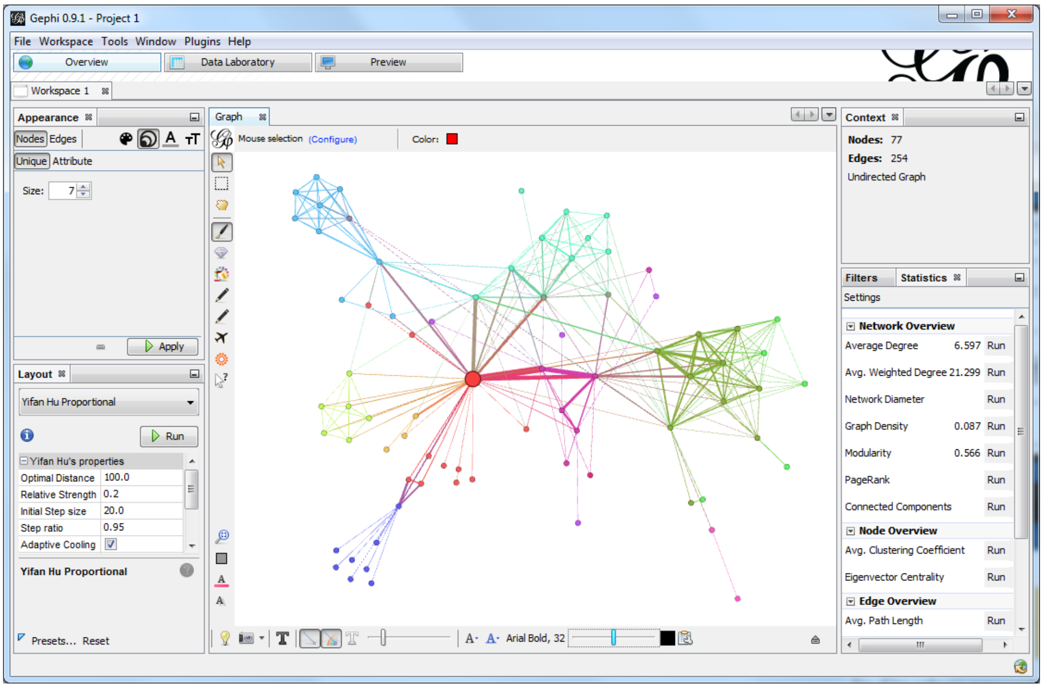
	Graph Creator	*Graph Online*	*GraphJS*	*Graphrel*	*yEd Live*	*Graphy*
graf nieskierowany	✓	✓	✓	–	✓	✓
graf skierowany	✓	✓	–	✓	✓	✓
multigraf	✓	–	✓	–	✓	✓
etykiety na krawędziach	✓	✓	✓	–	✓	✓
etykiety w wierzchołkach	✓	✓	✓	–	✓	✓
kolorowanie wierzchołków	✓	–	–	–	✓	✓
łuki jako krawędzie	✓	–	–	–	✓	✓
zaznaczanie kilku wierzchołków	✓	–	✓	–	✓	✓
grupowanie wierzchołków	–	–	–	–	✓	✓
przesuwanie widoku	–	✓	–	–	✓	✓
przybliżanie/oddalanie	–	✓	–	–	✓	✓
zapisywanie/wczytywanie	–	✓[1]	✓[2]	✓[3]	✓[4]	✓
algorytmy	–	✓	–	–	–	✓

[1] jako macierz sąsiedztwa lub jako obrazek
[2] własny format `JSON` lub jako LaTeX
[3] własny format (listy sąsiedztwa)
[4] `GraphML` lub jako obrazek (w wersji darmowej jest ograniczenie na wielkość zapisywanego grafu)

2.2 Aplikacje desktopowe

Istnieje również szereg aplikacji desktopowych służących do wizualizacji, analizy i edycji grafów. Są one bardziej rozbudowane i zakres ich funkcjonalności jest znacznie szerszy od aplikacji internetowych. Do najbardziej znanych należą [15]:

- Gephi
- GraphTea
- Cytoscape
- yEd Graph Editor



Rysunek 2.8: Zrzut ekranu z aplikacji Gephi

Rozdział 3

Narzędzia

3.1 Formaty zapisu grafów

Istnieje wiele formatów służących do opisu grafów. Do najpopularniejszych należą [7, 18]

- GML – *Graph Modeling Language*
- GraphML – *Graph Markup Language*
- GEXF – *Graph Exchange XML Format*
- JGF – *JSON Graph Format*
- DOT – format programu Graphviz
- DGML – *Directed Graph Markup Language*
- XGMML – *eXtensible Graph Markup and Modeling Language*

Graph Markup Language (GraphML)

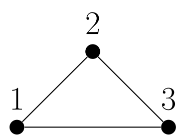
GraphML jest formatem zapisu grafów bazującym na składni XML. Format ten wspiera wszystkie typy grafów (skierowane, nieskierowane, mieszane). Wspiera również hipergrafy oraz grafy hierarchiczne. Dodatkowo umożliwia przypisywanie do wierzchołków i krawędzi atrybutów zawierających dane specyficzne dla aplikacji.

GraphML jest następcą formatu GML (nie będącego standardem XML). Prace nad formatem GML zostały zapoczątkowane przez społeczność *Graph Drawing* podczas Sympozjum Rysowania Grafów w 1995 roku w Pasawie w Niemczech. Pięć lat później przed 8. Międzynarodowym Sympozjum Rysowania Grafów w 2000 roku w Williamsburgu w USA ruszyły prace nad nowym formatem GraphML [19].

Plik w formacie GraphML zawiera element `graph` (graf), który może zawierać elementy `node` (wierzchołek), `edge` (krawędź) oraz `hyperedge` (hiperkrawędź). Każdy element `node` powinien zawierać unikalny atrybut `id` (identyfikator), który jest używany do definiowania krawędzi. Każda krawędź posiada atrybuty `source` (źródło) oraz `target` (cel), które odpowiadają identyfikatorom wierzchołków i oznaczają odpowiednio początek i koniec krawędzi. Najwyższym elementem w hierarchii jest `graphml`, który może zawierać serię elementów `key` (klucz) służących do definiowania atrybutów danych oraz elementów `graph`.

Z formatem tym związane są dwa rozszerzenia: *attribute extension* i *parseinfo extension*. Pierwsze z nich pozwala na wyspecyfikowanie typu atrybutu oraz jego nazwy. Drugie dodaje kilka atrybutów do elementów `graph` i `node`, takich jak ilość wierzchołków, ilość krawędzi, maksymalny stopień wierzchołka w grafie czy ilość krawędzi wychodzących dla wierzchołka. Metadane te mają na celu pomóc analizatorom składni (parserom) efektywniej przetwarzać pliki z grafami zapisanymi w formacie GraphML.

Format GraphML wspierają programy yEd Graph Editor oraz w ograniczonym stopniu Gephi (bez hipergrafów i grafów hierarchicznych).



Rysunek 3.1

Listing 3.1: Reprezentacja grafu z rysunku 3.1 w formacie GraphML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="1"/>
    <node id="2"/>
    <node id="3"/>
    <edge source="1" target="2"/>
    <edge source="2" target="3"/>
    <edge source="3" target="1"/>
  </graph>
</graphml>
```

Graph Exchange XML Format (GEXF)

Listing 3.2: Reprezentacja grafu z rysunku 3.1 w formacie GEXF

```xml
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
    <graph mode="static" defaultedgetype="directed">
        <nodes>
            <node id="0" label="1" />
            <node id="1" label="2" />
            <node id="2" label="3" />
        </nodes>
        <edges>
            <edge id="0" source="0" target="1" />
            <edge id="1" source="1" target="2" />
            <edge id="2" source="2" target="0" />
        </edges>
    </graph>
</gexf>
```

JSON Graph Format (JGF)

Listing 3.3: Reprezentacja grafu z rysunku 3.1 w formacie JGF

```
{
        "graph": {
                "nodes": [{
                        "id": "1",

                },
                {
                        "id": "2",

                },
                {
                        "id": "3",

                }],
                "edges": [{
                        "source": "1",
                        "target": "2"
                },
                {
                        "source": "2",
                        "target": "3"
                },
                {
                        "source": "3",
                        "target": "1"
                }]
        }
}
```

DOT Graphviz

Listing 3.4: Reprezentacja grafu z rysunku 3.1 w formacie DOT

```
graph graphname {
     1 -- 2 -- 3;
     3 -- 1;
}
```

3.2 Biblioteki do wizualizacji grafów w JavaScript

3.2.1 Cytoscape.js

Biblioteka z otwartym źródłem (ang. *open-source*) do analizy i wizualizacji grafów. Udostępniona na zasadach licencji MIT. Została napisana w czystym JavaScript i nie posiada zależności do żadnych innych bibliotek. Cytoscape.js jest następcą porzuconego projektu Cytoscape Web korzystającego z technologii Adobe Flash [14, s. 309].

Prawa własności intelektualnej posiada do niej Cytoscape Consortium – organizacja *non-profit*, która promuje rozwój i dystrybucję oprogramowania związanego z sieciami biologicznymi. Cytoscape.js została stworzona na University of Toronto. Jej głównym kontrybutorem jest Max Franz. Biblioteka została sfinansowana przez granty NRNB (*National Resource for Network Biology*) i NIH (*National Institutes of Health*). Kilka innych uniwersytetów oraz firm również pomagało w rozwoju biblioteki [17].

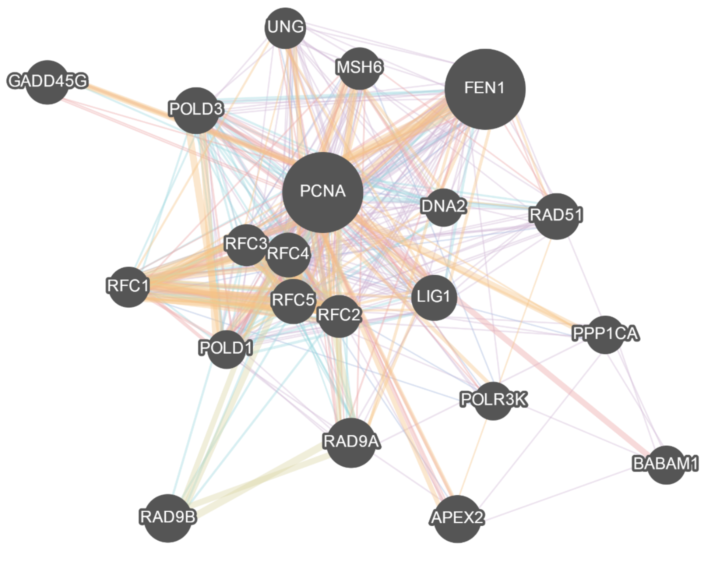
Cytoscape.js jest kompatybilny z kilkoma przydatnymi bibliotekami oraz środowiskami JavaScript, takimi jak: Node.js, Browserify, webpack, RequireJS czy Bower, co pozwala na integrację z szeroką gamą systemów opartych na JS.

Architektura Cytoscape.js pozwala na uruchomienie zarówno bez graficznego interfejsu użytkownika oraz jako komponent graficzny, którego implementacja bazuje na elemencie HTML5 Canvas (przykład przedstawiony jest na rysunku 3.2). Umożliwia to korzystanie z biblioteki zarówno po stronie klienta (np. przeglądarka internetowa), jak i po stronie serwera (np. Node.js).

Cytoscape.js wspiera różne typy grafów: skierowane, nieskierowane, multigrafy. Pozwala na dodawanie, usuwanie i modyfikację krawędzi oraz wierzchołków. Biblioteka dostarcza również możliwość grupowania wierzchołków.

W bibliotece jest zaimplementowanych kilka znanych algorytmów takich jak znajdowanie najkrótszej ścieżki, minimalnego drzewa rozpinającego czy minimalnego przekroju.

style, funkcje mapujące, wsparcie dla gestów myszy i urządzeń z ekranami dotykowymi przesuwanie wierzchołków, zmiana widoku przez przeciąganie lub przybliżanie/oddalanie wiązanie zdarzeń (ang. *event binding*) animacje import i eksport do obrazka (PNG/JPG), JSON (dodatek do GraphML) układ wierzchołków automatyczny: losowy, siatki (ang. *grid*), okręgu, koncentryczny, zdefiniowany przez przeszukiwanie grafu wszerz (ang. breadth-first search), cose (*Compound Spring Embedder* – układ korzystający z symulacji fizycznej) lub zdefiniowany przez programistę rozszerzalność – możliwość dopisania swoich własnych algorytmów, układów, itd. Wiele istniejących dodatków. wydajność – biblioteka jest w stanie obsłużyć i wyrenderować grafy posiadające tysiące elementów [14, s. 310], wydajność zależy od urządzenia, na którym jest uruchamiany kod, od silnika JS, rozmiaru grafu oraz użytych styli. W szczególności kosztowne do wyrenderowania są grawędzie, zwłaszcza w multigrafach ze względu na konieczność narysowania krzywych beziera. W dokumnetacji online jest wiele wskazówek dotyczących optymalizacji pod kątem wydajności (sekcja *Performance* [17]). Cytoscape.js posiada obszerną dokumentację online, która zawiera szczegółowy opis API (ang. *Application Programming Interface* – interfejs programistyczny), przykłady kawałków kodu oraz działające przykłady.



Rysunek 3.2: Przykład wizualizacji grafu w Cytoscape.js – krawędzie mogą mieć różny kolor i grubość, pomiędzy wierzchołkami może istnieć wiele krawędzi oraz wierzchołki mogą mieć różny rozmiar

3.2.2 sigma.js

3.2.3 VivaGraph.js

3.2.4 Linkurious.js

	Cytoscape.js	Sigma	VivaGraphJS
Licencja	MIT	MIT	BSD 3
Rozmiar	294	112,9	60,4
Renderowanie SVG HTML5 Canvas WebGL Canvas	 • • •	 tak tak tak	 • • •
Obsługiwane formaty	•	•	•
Rozszerzalność	•	•	•
•	•	•	•

Rozdział 4

Projekt aplikacji

W tym rozdziale przedstawię wszystkie wymagania funkcjonalne, które powinna spełniać aplikacja, aby użytkownik miał możliwość stworzyć graf dowolnego typu, wyświetlić go w optymalny sposób (wraz z możliwością zmiany widoku) oraz zmienić graf w dowolny sposób, np. poprzez dodawanie nowych wierzchołków i krawędzi, czy edycję etykiet.

Opiszę również wymagania niefunkcjonalne, aby praca z grafami była możliwie przystępna. Uwzględnię m.in.: wydajność, wspierane platformy, wygodną obsługę przez użytkownika oraz łatwą rozszerzalność dla programistów (co zostanie osiągnięte na przykład poprzez modułowowść kodu w JavaScript).

Na koniec rozdziału zaprezentuję prototyp graficznego interfejsu użytkownika uwzględniającego wszystkie wymagania funkcjonalne, który będzie obrazował jak powinna wyglądać aplikacja tworzona w ramach tej pracy.

4.1 Wymagania funkcjonalne

4.1.1 Tworzenie grafów

Podstawowym i oczywistym wymaganiem jest, aby użytkownik mógł stworzyć nowy, pusty graf skierowany oraz nieskierowany. Ponadto użytkownik powinien mieć możliwość zaimportowania istniejącego grafu oraz wygenerowania znanego grafu, np. cyklu lub grafu pełnego o zadanej ilości wierzchołków.

Importowanie grafów

Użytkownik powinien móc wczytać graf z komputera lub z chmury (np. Google Drive lub Dropbox) w trzech znanych formatach:

- `GraphML`,
- `GEXF`,
- `JGF`.

Opisy formatów znajdują się w sekcji 3.1.

Generowanie grafów

Użytkownik powinien mieć możliwość wygenerowania znanych grafów, dla zadanych parametrów wejściowych:

- grafu pustego,
- grafu liniowego,
- grafu cyklicznego,
- koła,
- grafu pełnego (lub turnieju dla grafów skierowanych),
- grafu pełnego dwudzielnego,
- grafu Petersena,
- drzewa (o zadanej wysokości i ilości dzieci)

Definicje i przykłady powyższych grafów znajdują się w sekcji 1.3.

Ponadto przydatnym dodatkiem w aplikacji będzie możliwość wygenerowania grafu losowego – o danej ilości wierzchołków oraz parametrem prawdopodobieństwa określającym, czy pomiędzy dwoma wierzchołkami istnieje krawędź.

4.1.2 Wizualizacja

Użytkownik powinien móc przesuwać widok, przybliżać i oddalać graf oraz rozmieszczać wierzchołki grafu w dowolny sposób. W aplikacji powinna istnieć możliwość zmiany układu grafu: układ oparty na oddziaływaniach (ang. *force-based layout*), układ siatki, układ okręgu, układ koncentryczny, układ hierarchiczny.

Użytkownik powinien być w stanie zmienić kategorię wierzchołka oraz typ krawędzi. Inne typy i kategorie powinny być oznaczone innym kolorem oraz powinna istnieć możliwość zmiany koloru.

Aplikacja powinna również dostarczać opcję wyszukiwania i filtrowania danych (np. tylko dany typ wierzchołków, wierzchołki o stopniu większym niż zadany parametr). Przydatną funkcjonalnością będzie wyświetlanie sąsiadów danego wierzchołka po najechaniu na niego kursorem myszy.

4.1.3 Edycja

W aplikacji powinien istnieć osobny tryb edycji. Gdy użytkownik jest w tym trybie, powinien móc dodawać oraz usuwać wierzchołki i krawędzie. Powinien być w stanie także dodawać oraz modyfikować etykiety wierzchołków i krawędzi.

Użytkownik powinien mieć możliwość zaznaczania wielu wierzchołków i krawędzi na raz. Użyteczną funkcjonalnością będzie również grupowanie (lub rozgrupowanie) zaznaczonych wierzchołków.

Aplikacja powinna wyświetlać ostatnio wykonaną akcję oraz udostępniać możliwość jej cofnięcia.

4.1.4 Przetwarzanie

Aplikacja powinna dawać możliwość wykonania podstawowych algorytmów na danym grafie:

- wyszukiwanie najkrótszej ścieżki pomiędzy dwoma wybranymi wierzchołkami,
- znajdowanie minimalnego drzewa rozpinającego,
- obliczanie algorytmu PageRank,
- znajdowanie (silnie) spójnych składowych oraz dwuspójnych składowych,
- znajdowanie cyklu Eulera,
- znajdowanie cyklu Hamiltona.

4.1.5 Eksportowanie

Użytkownik powinien mieć możliwość wyeksportowania do formatów, które zostały przedstawione w podsekcji 4.1.1.

Ponadto przydatną funkcjonalnością będzie możliwość wyeksportowania obecnego widoku do pliku graficznego, np. `PNG` lub `JPG`.

4.1.6 Udostępnianie grafu

W aplikacji powinna istnieć możliwość udostępniania grafu innym użytkownikom. Po wybraniu tej opcji, powinien zostać wygenerowany unikalny odnośnik do grafu. Po przejściu na ten adres (w podstawowej wersji) inni użytkownicy mogą wyświetlić i edytować graf.

4.2 Wymagania niefunkcjonalne

4.2.1 Wydajność

Aplikacja powinna być w stanie efektywnie wyświetlać oraz modyfikować grafy. Małe grafy (mające do około 50 wierzchołków) powinny być wyświetlane od razu, podobnie modyfikacja takich grafów powinna być odzwierciedlana natychmiast. Podczas wyświetlania oraz edycji grafów średnich (mających od 50 do 1000 wierzchołków) dozwolone jest niewielkie opóźnienie, mieszące się w graniach od 300-1500 ms.

Aplikacja powinna obsługiwać również duże grafy (mające np. 1 milion wierzchołków). W przypadku takich grafów dozwolone są opóźnienia jednakże ich postęp powinien być przedstawiany użytkownikowi, interfejs nie powinien być blokowany oraz użytkownik powinien mieć możliwość anulowania zbyt długo trwających operacji. Przydatną funkcjonalnością będzie również powiadamianie użytkownika o akcji, która może zająć dłuższy czas (np. powyżej 5 sekund).

System powinien być w stanie obsługiwać wielu użytkowników – wzrost liczby użytkowników nie powinien mieć większego wpływu na responsywność oraz szybkość odpowiedzi. Wymaganie to powinno być spełnione w łatwy sposób, ponieważ większość operacji (choć nie wszystkie) będzie wykonywana po stronie klienta (w przeglądarce internetowej). Dla tych operacji, które nie będą wykonywane na komputerze użytkownika, rozsądnym wymaganiem jest, aby część serwerowa była łatwo skalowalna.

4.2.2 Wspierane platformy

Część kliencka aplikacji powinna działać na wszystkich popularnych systemach operacyjnych (Windows, Linux, Mac OS) oraz przeglądarkach (Google Chrome, Mozilla Firefox, Internet Explorer, Safari, Opera). Jeśli chodzi o część serwerową, to również powinna istnieć możliwość uruchomienia jej pod dowolnym system operacyjnym.

Ze względu na wzrost znaczenia urządzeń mobilnych powinna być możliwość łatwego korzystania z aplikacji na ów urządzeniach (zwłaszcza na tabletach, które posiadają na tyle duży ekran, aby móc wygodnie wyświetlić graf i edytować go). By było to możliwe aplikacja musi: po pierwsze, automatycznie dostosowywać się do rozmiaru okna (ang. *Responsive Web Design*); po drugie, wspierać gesty obsługiwane przez urządzenia przenośne, np. przeciągnięcie, wykorzystanie dwóch palców, przytrzymanie elementu na ekranie.

4.2.3 Użyteczność

Aplikacja powinna spełniać kryteria użyteczności (ang. *usability*), aby praca z nią była jak najbardziej intuicyjna, prosta i przyjemna. Jakob Nielsen podaje 5 elementów, które wchodzą w skład użyteczności [12]:

- **Nauczalność** (ang. *learnability*) – jak łatwa jest dla użytkowników realizacja podstawowych zadań, gdy po raz pierwszy korzystają z aplikacji?

  Dla nowych użytkowników powinny wyświetlać się podpowiedzi, które pozwolą im jak najszybciej nauczyć się obsługi programu. Dla użytkowników, którzy uruchomią aplikację po raz pierwszy powinien otworzyć się krótki (opcjonalny) przewodnik, który oprowadzi ich po aplikacji i zapozna z wszystkimi dostępnymi funkcjonalnościami.

- **Efektywność** (ang. *efficiency*) – gdy użytkownicy znają program, jak szybko mogą wykonywać zadania?

  Aplikacja powinna oferować skróty klawiszowe, które przyspieszą pracę zaawansowanych użytkowników. W menu powinna być opcja wyświetlenia wszystkich skrótów klawiszowych oraz powinny być one podpowiadane użytkownikowi przy starcie lub podczas korzystania z programu.

- **Zapamiętywalność** (ang. *memorability*) – jak łatwo użytkownicy mogą przywrócić biegłość korzystania z aplikacji, gdy powracają do niej po dłuższej przerwie?

  Użytkownik powinien mieć możliwość ponownego włączenia podpowiedzi, przewodnika oraz wyświetlenia listy wszystkich dostępnych skrótów klawiszowych.

- **Błędy** (ang. *errors*) – jak wiele błędów popełniają użytkownicy, jak poważne są te błędy, jak łatwo mogą je poprawić?

  Możliwość popełnienia błędu powinna być zminimalizowana do zera, np. poprzez specjalny tryb edycji użytkownik nie jest w stanie przypadkowo dodać nowy wierzchołek. Ponadto po każdej akcji modyfikującej powinno wyświetlić się powiadomienie (ang. *toast*, *snack-bar*) mówiące o tym, w jaki sposób graf się zmienił oraz przycisk z opcją cofnięcia ostatnio wykonanej operacji. Powinna istnieć również opcja cofania ostatnich akcji i ponawiania ich korzystając ze znanych skrótów klawiszowych `Ctrl+Z` oraz `Ctrl+Y`.

- **Satysfakcja** (ang. *satisfaction*) – jak przyjemne jest korzystanie z programu?

  Wszystkie przedstawione powyżej wymagania funkcjonalne i niefunkcjonalne powinny przyczynić się do tego, że użytkownik będzie mógł w łatwy i przyjemny sposób tworzyć, wyświetlać i edytować swoje grafy.

4.2.4 Rozszerzalność

Aplikacja zostanie udostępniona na zasadach otwartego oprogramowania (ang. *open source*). Dlatego tez powinna w łatwy sposób dać się rozszerzać przez innych programistów, dając możliwość np. dodawania nowych algorytmów, sposobów importowania oraz eksportowania grafów, czy obsługi nowych formatów.

Rozszerzalność będzie zapewniona przede wszystkim przez:

- strukturę i modułowość kodu,
- dokumentację interfejsów aplikacji i opis architektury systemu,
- przykłady w jaki sposób zrealizować znane problemy,
- wysokie pokrycie testami jednostkowymi i integracyjnymi, które zagwarantują, że modyfikacja kodu nie zepsuje istniejących już funkcjonalności.

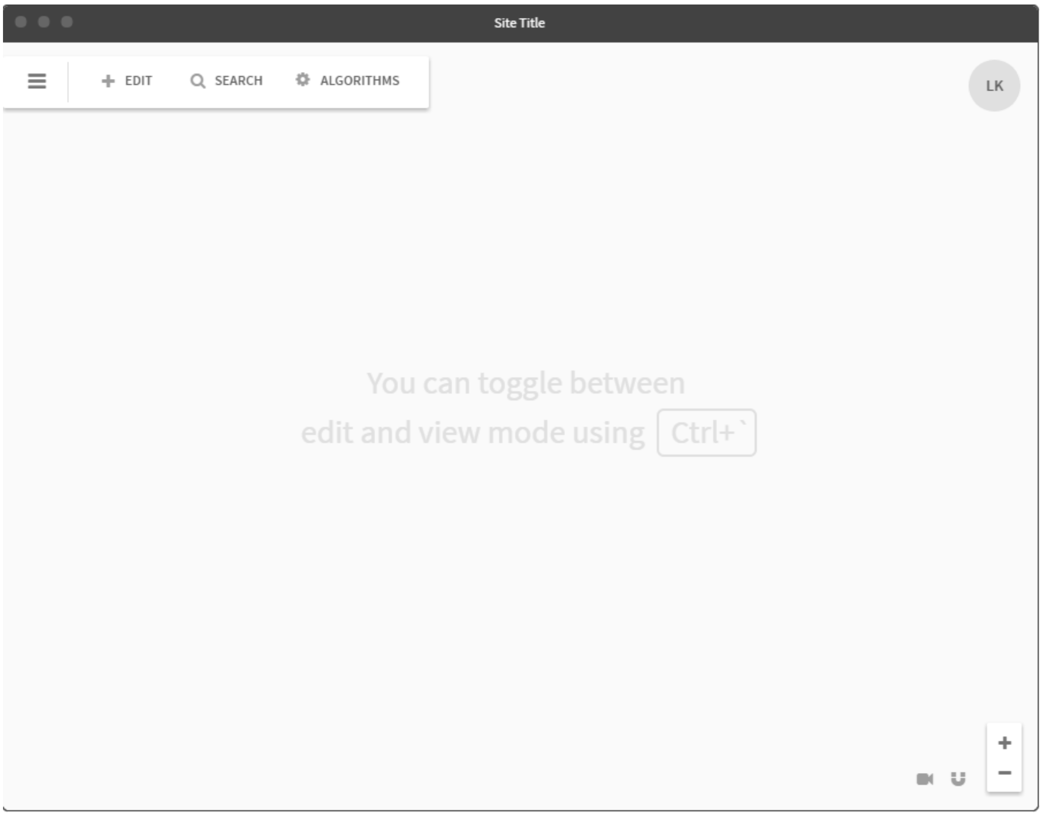
4.3 Prototyp interfejsu użytkownika

Tryb widoku grafu

Graf powinien być wyświetlony w całym oknie przeglądarki. U góry po lewej powinno znajdować się rozsuwane menu, a obok niego przyciski: *Edycja*, *Wyszukaj* i *Algorytmy*. Z kolei po stronie prawej powinien znajdować się pływający przycisk (ang. *floating button*) wyświetlający informację o zalogowanym użytkowniku i dający możliwość wyświetlić menu użytkownika.

Na dole po prawej stronie powinno znajdować się menu z opcjami zmieniającymi widok grafu (przybliżanie i oddalanie, zmiana układu, włączenie i wyłączenie oddziaływania pomiędzy wierzchołkami).

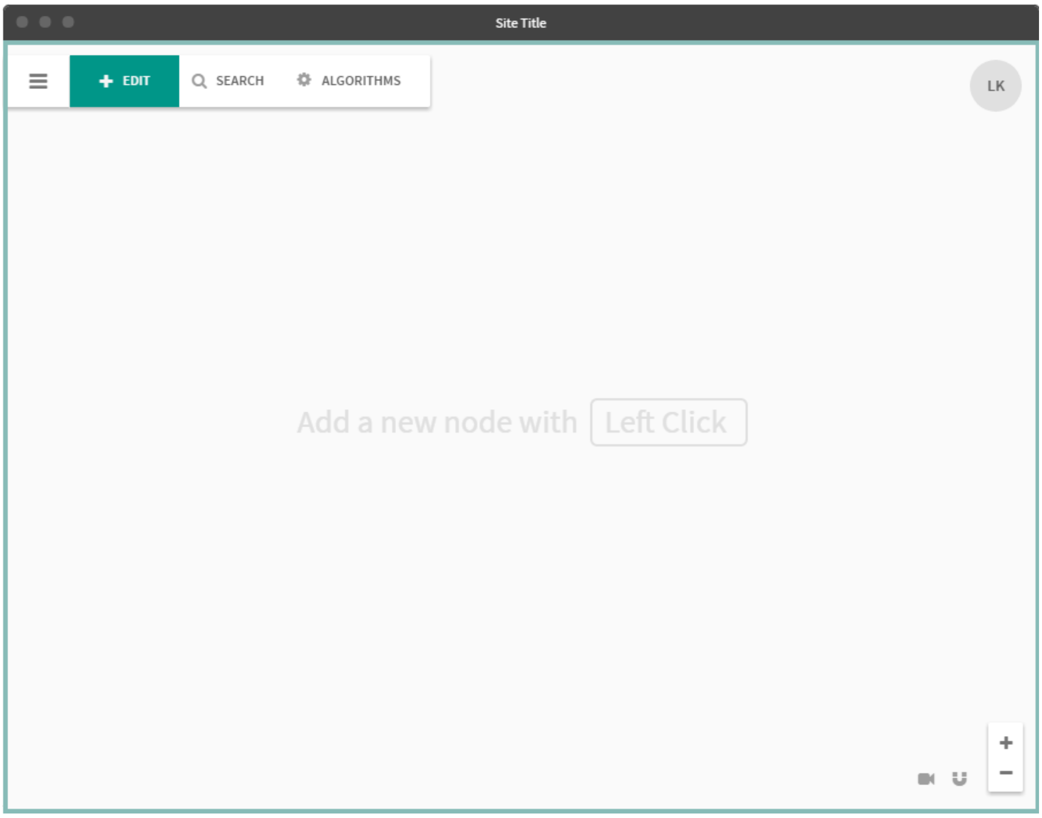
Gdy nie ma otwartego żadnego grafu, to w tle powinny pojawiać się podpowiedzi ze skrótami klawiszowymi, które co kilka sekund będą się zmieniać.



Rysunek 4.1: Tryb widoku grafu

Tryb edycji grafu

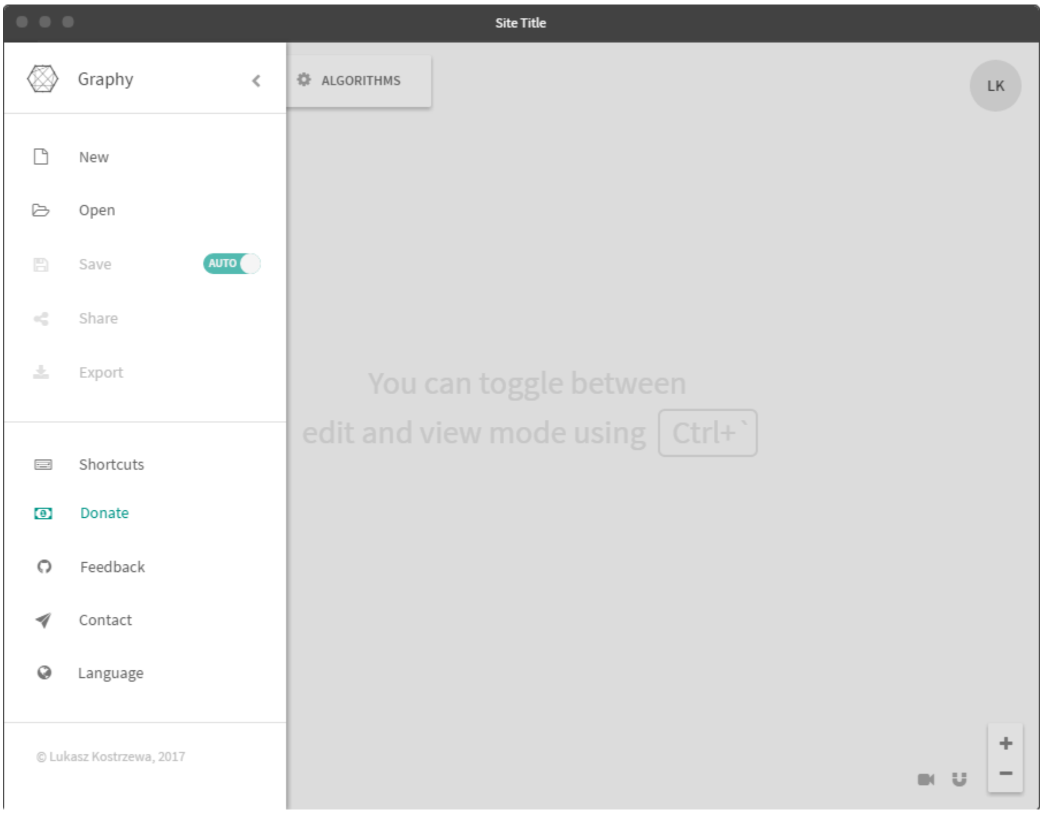
Użytkownik powinien być poinformowany, że jest w trybie edycji poprzez oznaczenie przycisku oraz obramowania okna wyróżniającym się kolorem.



Rysunek 4.2: Tryb edycji grafu

Widok menu

Menu powinno wysuwać się z boku, a pod nim powinna pojawić się warstwa z półprzezroczystym tłem. Powinno oferować podstawowe opcje tworzenia nowego grafu, otwierania zapisanego grafu, udostępniania czy eksportowania. W menu powinny znaleźć się też takie opcje jak: wyświetlenie listy skrótów, zmiana języka, kontakt czy możliwość zgłoszenia błędu w aplikacji oraz informacja o prawach autorskich.

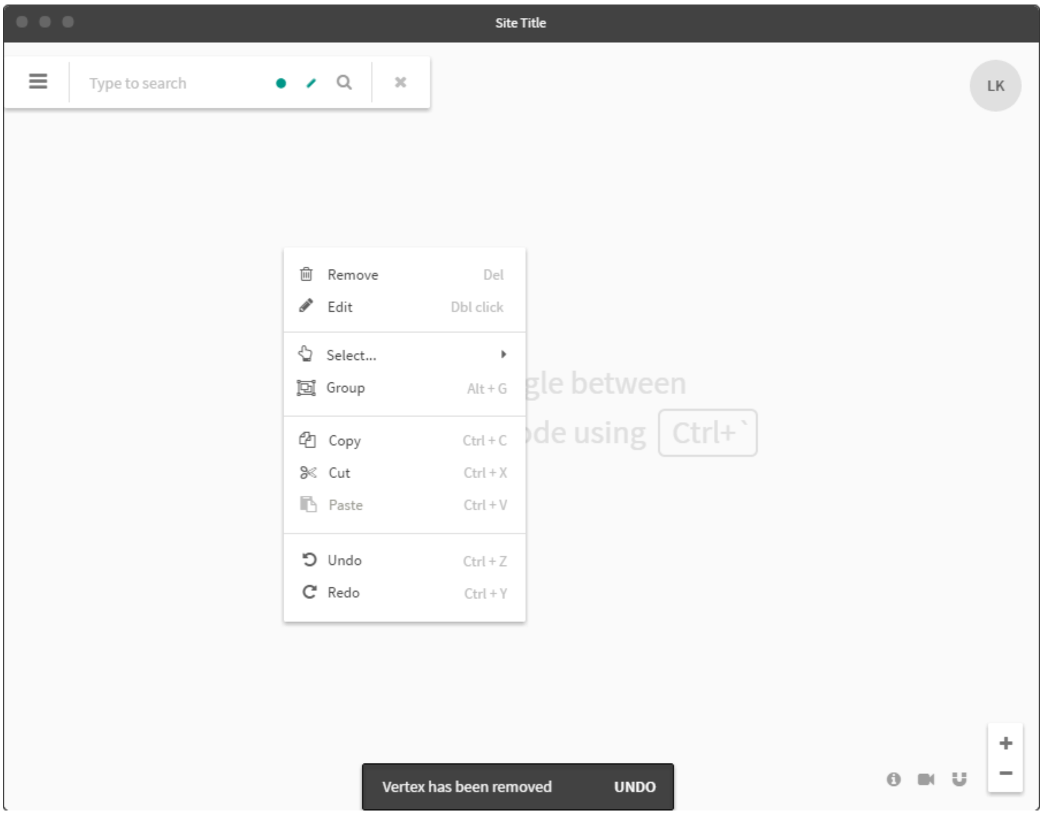


Rysunek 4.3: Widok menu

Menu kontekstowe i informacja o ostatniej akcji

Po kliknięciu prawego klawisza myszy powinno pojawić się niestandardowe menu kontekstowe zawierające dodatkowe opcje pozwalające na edycję, usuwanie, zaznaczanie, grupowanie, wycinanie, kopiowanie, wklejanie wierzchołków i krawędzi oraz na cofanie i ponawianie ostatniej akcji.

Po wykonaniu akcji modyfikującej graf na dole strony powinno wyświetlić się powiadomienie o wykonaniu tej akcji z przyciskiem umożliwiającym jej cofnięcie.



Rysunek 4.4: Menu kontekstowe i informacja o ostatniej akcji

Rozdział 5

Implementacja

5.1 Testy

Rozdział 6

Instrukcje dla użytkowników

6.1 Przypadek użycia

Rozdział 7

Wnioski

Bibliografia

[1] Robin J. Wilson i Lowell W. Beineke. *Applications of Graph Theory.* London: Academic Press, 1975. ISBN: 0-12-757840-4.

[2] J. A. Bondy i U. S. R. Murty. *Graph Theory with Applications.* 5 wyd. Elsevier Science Ltd/North-Holland, 1982. ISBN: 0-444-19451-7.

[3] Monika R. Henzinger i in. *„Faster Shortest-Path Algorithms for Planar Graphs"*. W: *Journal of Computer and System Sciences* 55 (sierp. 1997), s. 3–23. URL: http://www.sciencedirect.com/science/article/pii/S0022000097914938 (term. wiz. 18. 05. 2017).

[4] Lech Banachowski, Krzysztof Diks i Wojciech Rytter. *Algorytmy i struktury danych.* 2 wyd. Warszawa: Wydawnictwa Naukowo-Techniczne, 1999. ISBN: 83-204-2403-8.

[5] Joan M. Aldous i Robin J. Wilson. *Graphs and Applications: An Introductory Approach.* 1 wyd. Springer-Verlag London, 2000. ISBN: 978-1-85233-259-4.

[6] Brian Hopkins i Robin Wilson. *„The Truth about Königsberg"*. W: *College Mathematics Journal* 35 (maj 2004), s. 198–207. URL: https://www.maa.org/sites/default/files/pdf/upload_library/22/Polya/hopkins.pdf (term. wiz. 29. 04. 2017).

[7] S. Mohammed i M. Bernard. *Graph File Formats.* Spraw. tech. Mona, Kingston, Jamajka: Department of Mathematics and Computer Science, The University of the West Indies, 2004. URL: http://www2.sta.uwi.edu/~mbernard/research_files/fileformats.pdf (term. wiz. 29. 04. 2017).

[8] Shariefuddin Pirzada i Ashay Dharwadker. *„Applications of Graph Theory"*. W: *Journal of The Korean Society for Industrial and Applied Mathematics* 11 (kw. 2007), s. 19–38. URL: http://www.dharwadker.org/graph_theory_applications.pdf (term. wiz. 19. 05. 2017).

[9] Robin J. Wilson. *Wprowadzenie do teorii grafów.* 2 wyd. Warszawa: Wydawnictwo Naukowe PWN, 2007. ISBN: 978-83-01-15066-2.

[10] Ivan Gutman. *„The chemical formula C_nH_{2n+2} and its mathematical background"*. W: *The Teaching of Mathematics* 11 (lut. 2008), s. 53–61. URL: http://elib.mi.sanu.ac.rs/files/journals/tm/21/tm1121.pdf (term. wiz. 22. 05. 2017).

[11] Kenneth A. Ross i Charles R.B. Wright. *Matematyka dyskretna*. 5 wyd. Warszawa: Wydawnictwo Naukowe PWN, 2008. ISBN: 978-83-01-14380-0.

[12] Jakob Nielsen. *Usability 101: Introduction to Usability*. Nielsen Norman Group. 4 sty. 2012. URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (term. wiz. 12. 06. 2017).

[13] Thomas H. Cormen i in. *Wprowadzenie do algorytmów*. 7 wyd. Warszawa: Wydawnictwo Naukowe PWN, 2013. ISBN: 978-83-01-16911-4.

[14] Max Franz i in. *„Cytoscape.js: a graph theory library for visualisation and analysis"*. W: *Bioinformatics* 32.2 (2016), s. 309–311. URL: http://dx.doi.org/10.1093/bioinformatics/btv557 (term. wiz. 24. 05. 2017).

[15] Mathematics Stack Exchange. *Graph theory software*. Stack Overflow. URL: https://math.stackexchange.com/questions/58973/graph-theory-software (term. wiz. 16. 05. 2017).

[16] Mathematics Stack Exchange. *Online tool for making graphs (vertices and edges)*. Stack Overflow. URL: https://math.stackexchange.com/questions/13841/online-tool-for-making-graphs-vertices-and-edges (term. wiz. 02. 05. 2017).

[17] Max Franz. *Cytoscape.js*. Cytoscape Consortium. URL: http://js.cytoscape.org/ (term. wiz. 24. 05. 2017).

[18] Gephi. *Supported Graph Formats*. The Gephi Consortium. URL: https://gephi.org/users/supported-graph-formats/ (term. wiz. 29. 04. 2017).

[19] GraphML Working Group. *About GraphML*. Graph Drawing. URL: http://graphml.graphdrawing.org/about.html (term. wiz. 13. 06. 2017).

[20] Linkurious. *Find hidden insights in your graph data*. Linkurious SAS. URL: http://linkurio.us/product/ (term. wiz. 16. 05. 2017).