Matematyka dyskretna

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytet Gdański

sem. zimowy 2025/26

inf.ug.edu.pl/~amb/MaD

Algorytmy grafowe

Reprezentacja grafów

- Macierz incydencji:
  graf $G = (V, E)$, $V = \{0, \ldots, n-1\}$, $E \subseteq \mathcal{P}_2(V)$, $E = \{0, \ldots, k-1\}$
  macierz dwuwymiarowa $n \times k$: $G(i, j) = 1$ jeśli wierzchołek i jest końcem krawędzi j
  duża rozrzutność, każda kolumna ma tylko dwie jedynki
- Macierz sąsiedztwa: graf $G = (V, E)$, $V = \{0, \ldots, n-1\}$, $E \subseteq \mathcal{P}_2(V)$
  macierz dwuwymiarowa $n \times n$: $G(i, j) = 1$ jeśli wierzchołki i oraz j są połączone krawędzią
  (musi być symetryczna lub czytamy tylko indeksy $i < j$)
  zajmuje n^2 komórek nawet jeśli krawędzi jest b. mało

Reprezentacja grafów c.d.

- Listy incydencji:
  dla każdego wierzchołka v lista $L(v)$ jego sąsiadów
  - zajętość miejsca jest proporcjonalna do liczby krawędzi i wierzchołków
  - fakt: każda krawędź występuje dwa razy
    czyli algorytm usuwania krawędzi musi usunąć obie kopie, można utrzymywać powiązanie między kopiami
  - lista może być dwukierunkowa

Przeszukiwanie grafu wgłąb

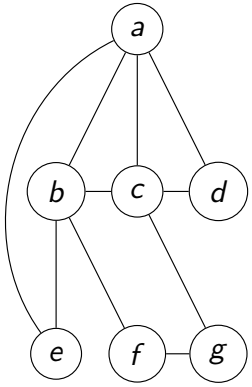
- przeszukaj (G,v)
  begin odwiedź v; odwiedzony(v)=true;
  for u in L(v) **if not** odwiedzony(u) przeszukaj(G,u);
  end
- dla całego grafu trzeba przejrzeć wszystkie wierzchołki:
  for v in V **if not** odwiedzony(v) przeszukaj(G,v);
  (na początku wszystkie są zaznaczone jako nieodwiedzone)
- każdy wierzchołek będzie odwiedzony, ale tylko raz
  (sprawdzanie może nastąpić wiele razy)
- przy okazji zaznaczane są składowe spójności – zaczynają się od wierzchołków, które są nieodwiedzone w głównej pętli
- jeśli kolejność przeglądania wierzchołków nie jest z góry ustalona, to algorytm jest niedeterministyczny

Przeszukiwanie grafu wgłąb – wersja ze stosem

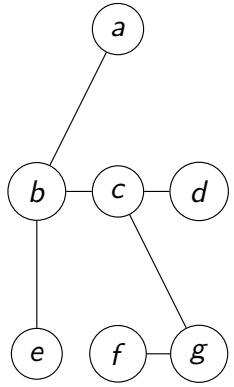
przeszukaj (G,v)
begin stos=∅; połóż(stos,v); odwiedź v; odwiedzony(v)=true;
  while stos≠∅
    begin u=wierzch(stos);
      szukamy na liście L(u) wierzchołka nieodwiedzonego
      if t jest takim wierzchołkiem
        begin połóż(stos,t); odwiedź t; odwiedzony(t)=true;
        end
      else zdejmij(stos,u);
    end

zawartość stosu w każdym momencie wskazuje na drogę od wierzchołka v do wierzchołka na stosie

Przeszukiwanie grafu wgłąb – przykład



u	t	stos
	a	a
a	b	a, b
b	c	a, b, c
c	d	a, b, c, d
d		a, b, c
c	g	a, b, c, g
g	f	a, b, c, g, f
f		a, b, c, g
g		a, b, c
c		a, b
b	e	a, b, e
e		a, b
b		a
a		$\varnothing$

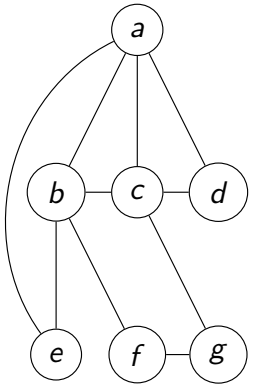


Przeszukiwanie grafu wszerz – kolejka

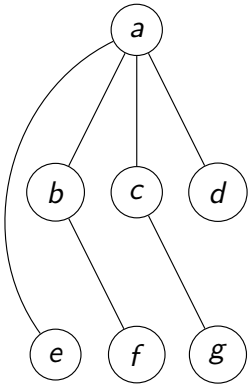
przeszukaj (G,v)
begin kolejka=∅; dodaj(kolejka,v); zaznaczony(v)=true;
  while kolejka≠∅
    begin u=pobierz(kolejka); odwiedź(u)
      for t ∈ L(u)
      if zaznaczony(t)=false
        begin dodaj(kolejka,t); zaznaczony(t)=true;
        end
    end
end

jeśli chcemy zapamiętać drogę od wierzchołka v do innych odwiedzanych, musimy utrzymywać tablicę poprzed(t)=u przy zaznaczaniu wierzchołka

Przeszukiwanie grafu wszerz – przykład



u	t	kolejka
	a	a
a	b	b
	c	b, c
	d	b, c, d
	e	b, c, d, e
b	f	c, d, e, f
c	g	d, e, f, g
d		e, f, g
e		f, g
f		g
g		$\varnothing$



Drzewa rozpinające

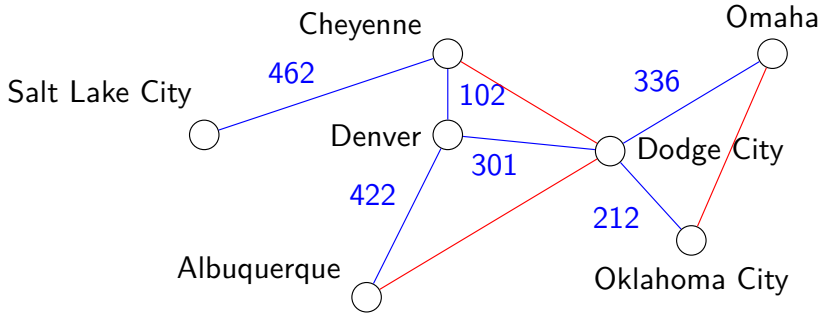
- def: drzewo rozpinające graf (V, E) jest to podgraf o tym samym zbiorze wierzchołków, (V, T) jest drzewem i $T \subseteq E$.
- budowa drzewa/lasu rozpinającego
  - z dowodu twierdzenia o istnieniu, dodawanie krawędzi
  - z dowodu twierdzenia o istnieniu, usuwanie krawędzi
  - przeszukiwanie wgłąb
  - przeszukiwanie wszerz
- tw.: przeszukiwanie wszerz znajduje najkrótszą drogę pomiędzy wierzchołkami
  (nie ma tej własności przeszukiwanie wgłąb)

Minimalne drzewo rozpinające

- założenie: krawędzie grafu mają wagi $\ell : E \to \mathbb{R}_+$
- cel: zbudować drzewo (las) rozpinający o minimalnej sumie wag $\Sigma\{\ell(e) \mid e \in T\}$
- algorytm: uporządkować krawędzie w/g wag zaczynając od najmniejszej
  w pętli rozważać kolejne krawędzie i dodawać do zbioru, o ile nie tworzy się cykl
- graf będzie na pewno acykliczny (tego pilnujemy)
- i będzie łączyć każde wierzchołki, które mają połączenie w grafie (bo nie dodajemy krawędzi jedynie, gdy końce już są połączone)
- i będzie miał najmniejszą możliwą sumę wag (bo rezygnacja z krawędzi o mniejszym koszcie wymagałaby zastąpienia ich krawędziami o większym koszcie)

Minimalne drzewo rozpinające – przykład

		2	3	4	5	6	Omaha
1	Denver	512	422	513	102	301	540
2	Salt Lake City		611	894	462	682	955
3	Albuquerque			525	522	425	895
4	Oklahoma City				706	212	454
5	Cheyenne					355	493
6	Dodge City						336

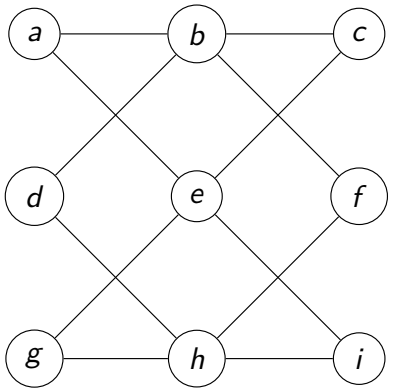


Cykle i ścieżki Eulera

- cykl/ścieżka zawierająca wszystkie krawędzie grafu (odwiedzając wierzchołki wielokrotnie)
- szukamy cyklu Eulera dla grafu bez wierzchołków stopnia nieparzystego zaczynając od wierzchołka v

CE=$\varnothing$;
while L(v)$\neq\varnothing$
  if L(v)=$\{u\}$
    begin V=V$\setminus v$; E=E$\setminus\{uv\}$; CE=CE$\bigcup\{uv\}$; v=u;
    end
  else if $|$L(v)$| > 1$
    begin wybieramy takiego sąsiada u, że operacja E=E$\setminus\{uv\}$; zostawi graf spójny
E=E$\setminus\{uv\}$; CE=CE$\bigcup\{uv\}$; v=u;
    end

Ścieżki i cykle Eulera – przykład



v	u	$\|L(v)\|$	
a	b	2	
b	c	3	
c	e	1	
e	g	3	ea niemożliwe
g	h	1	
h	f	3	hi niemożliwe
f	b	1	
b	d	1	
d	h	1	
h	i	1	
i	e	1	
e	a	1	

Cykle i ścieżki Hamiltona

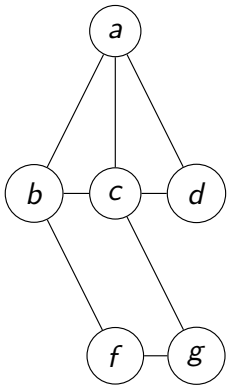
- cykl/ścieżka odwiedzająca wszystkie wierzchołki grafu (raczej nie korzysta z wszystkich krawędzi),
- problem komiwojażera: j.w. plus jeszcze zminimalizować długość ścieżki mając dane długości łuków,
- **nie ma** algorytmów znajdowania ścieżek Hamiltona o rozsądnej złożoności, tzn. gwarantującej w czasie wielomianowym znalezienie ścieżki,
- problem ten należy do klasy problemów *NP*-zupełnych, które są równoważne pod względem złożoności i dla żadnego z nich nie znamy algorytmu wielomianowego,
- nie przeszkadza to w opracowywaniu algorytmów, które prawie w każdym przypadku mogą być wykonane i przynieść szukany rezultat.

Cykle i ścieżki Hamiltona - algorytm z nawrotami

- szukamy ścieżki Hamiltona w grafie (V, E) zaczynając od wierzchołka v
- algorytm na pewno rozwiąże problem i ma b. dużą złożoność

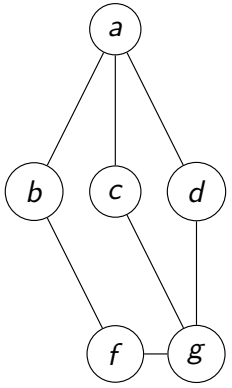
stos=$\{v\}$;
while stos $\neq\varnothing$
  begin u=wierzch(stos);
    szukamy sąsiada u, którego nie ma stosie i ma numer większy od ostatnio zdjętego
    if t jest takim wierzchołkiem
      begin połóż(stos,t);
        if na stosie leżą wszystkie wierzchołki – jest ścieżka
      end
    else zdejmij(stos,u);
  end
nie ma ścieżki Hamiltona (a stos = $\varnothing$)

Cykle i ścieżki Hamiltona – przykład



u	t	stos	
		a	
a	b	a, b	
b	c	a, b, c	
c	d	a, b, c, d	
d		a, b, c	
c	g	a, b, c, g	cd już było
g	f	a, b, c, g, f	
f		a, b, c, g	
g		a, b, c	gf już było
c		a, b	cd i cg już było
b	f	a, b, f	
f	g	a, b, f, g	
g	c	a, b, f, g, c	
c	d	a, b, f, g, c, d	cykl Hamiltona

Cykle i ścieżki Hamiltona – kontrprzykład



stos $= \varnothing$ i nie ma cyklu Hamiltona

u	t	stos	
		a	
a	b	a, b	
b	f	a, b, f	
f	g	a, b, f, g	
g	c	a, b, f, g, c	
c		a, b, f, g	
g		a, b, f, g, d	
d		a, b, f, g	
g		a, b, f	gc już było
f		a, b	
b		a	
a	c	a, c	
po kilku próbach wersja ac upadnie			
a	d	a, d	
po kilku próbach wersja ad upadnie			

Hiperkostka – ciągi bitów

- $Q^n = \{0,1\}^n$ – ciągi bitów długości n
  $E = \{\{u, v\} \mid u \oplus v$ ma jedną jedynkę$\}$
- $|Q^n| = 2^n$, $|E| = n \cdot 2^{n-1}$
- dla $n = 1$ jest jedna krawędź, która jest ścieżką Hamiltona, byłaby cyklem, gdyby pozwolić na powrót
- dla $n = 2$ cyklem jest 00, 01, 11, 11, 00
- jeśli $v_1, \ldots, v_{2^n}, v_1$ jest cyklem Hamiltona w Q^n, to $0v_1, \ldots, 0v_{2^n}, 1v_{2^n}, \ldots, 1v_1, 0v_1$ jest cyklem Hamiltona w Q^{n+1} (kod Graya)