Laboratorium jęz. programowania

ZASTOSOWANIA STRUKTUR DANYCH
w przykładowych algorytmach grafowych

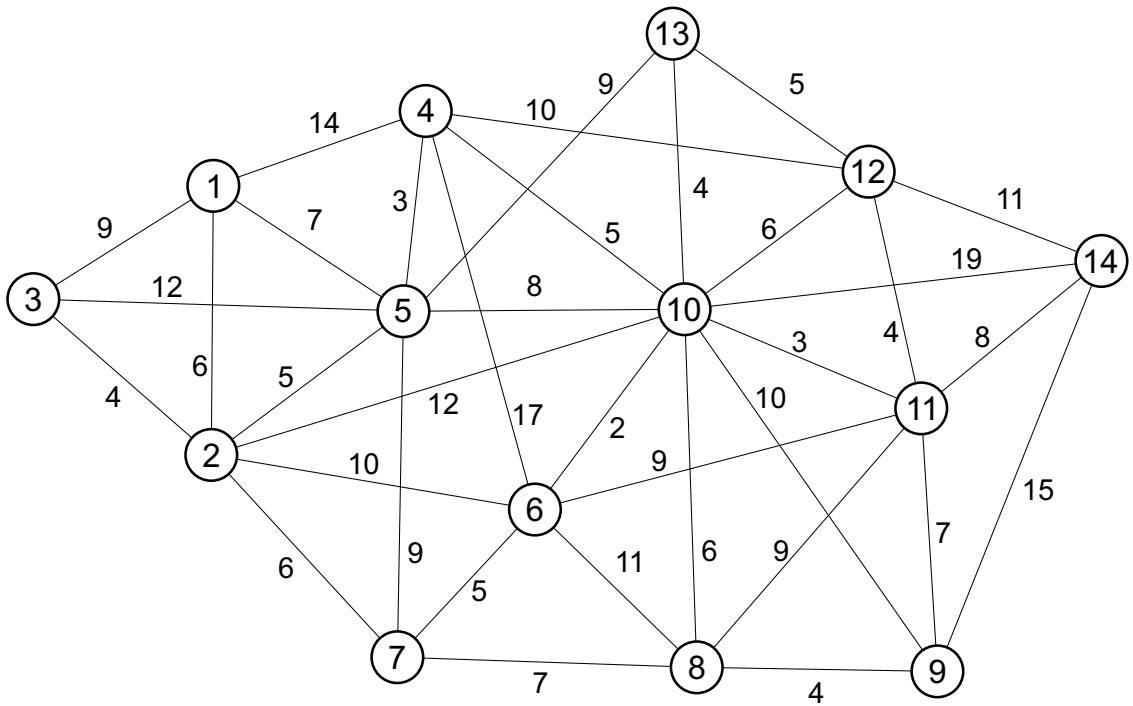
Miłosz Michalski

Institute of Physics
Nicolaus Copernicus University

Kwiecień 2024

Algorytm Kruskala – optymalne drzewo spinające

Opis zadania: Graf reprezentuje miejsca na mapie, które należy połączyć siecią światłowodą. Krawędzie oznaczają możliwy przebieg linii, dla każdej krawędzi podany jest szacunkowy koszt budowy odp. odcinka sieci. Należy wybrać wariant połączenia o minimalnym sumarycznym koszcie.



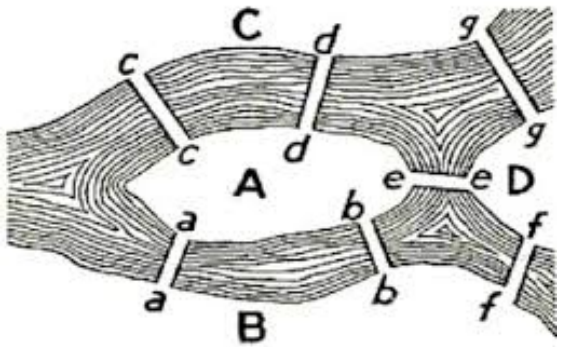
Algorytm Kruskala

1. Algorytm dodaje opisy krawędzi (jako pary połączonych wierzchołków (u,v) z wagami C(u,v) do kolejki priorytetowej Q, porządkując je rosnąco względem wag.
2. Krawędzie pobierane są kolejno z Q i dodawane do częściowego rozwiązania, jeśli nie powoduje to zamknięcia cyklu.
3. Po dodaniu n-1 krawędzi rozwiązanie jest kompletne.

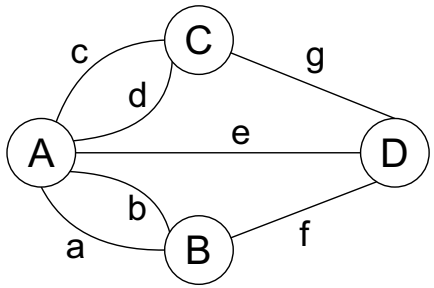
Algorytm Kruskala

```
while (!eof(input)) { czytaj(u,v,c); pushq(Q,u,v,c); }
m=0;  T=NULL;  koszt=0;
while (!empty(Q) && m<n-1){
    e = popq(Q);
    if (T+{e} nie zawiera cyklu){
        T = T+{e};
        koszt = koszt + e.c;
        m++;
    }
}
if (m==n-1) drukuj rozwiązanie;
```

Wyznaczanie cyklu Eulera w grafie

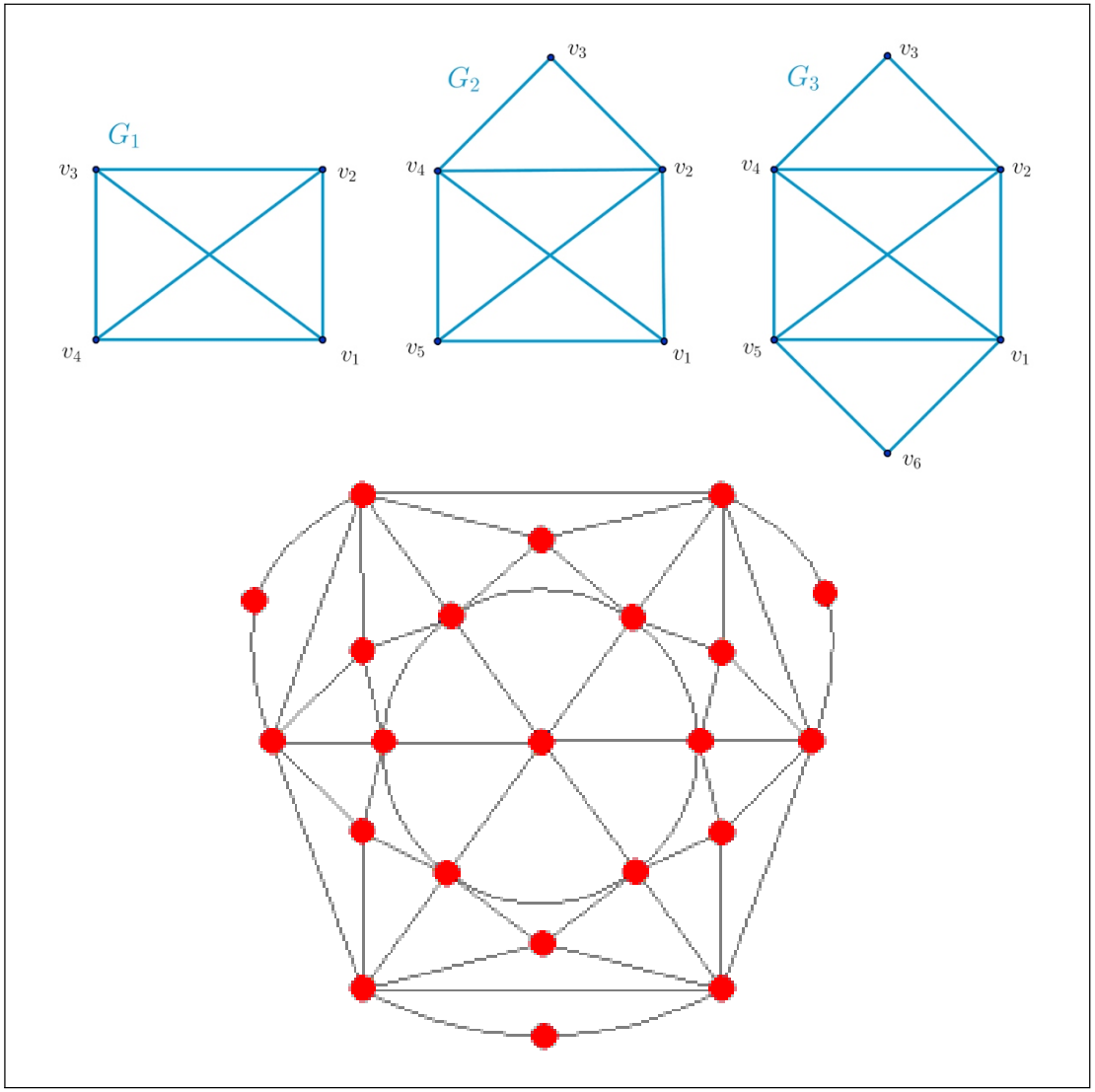


MOSTY KRÓLEWIECKIE (1736)



Twierdzenie Eulera: W grafie istnieje cykl zawierający wszystkie krawędzie wtedy i tylko wtedy gdy stopnie wszystkich wierzchołków są parzyste.

Istnienie cyklu Eulera



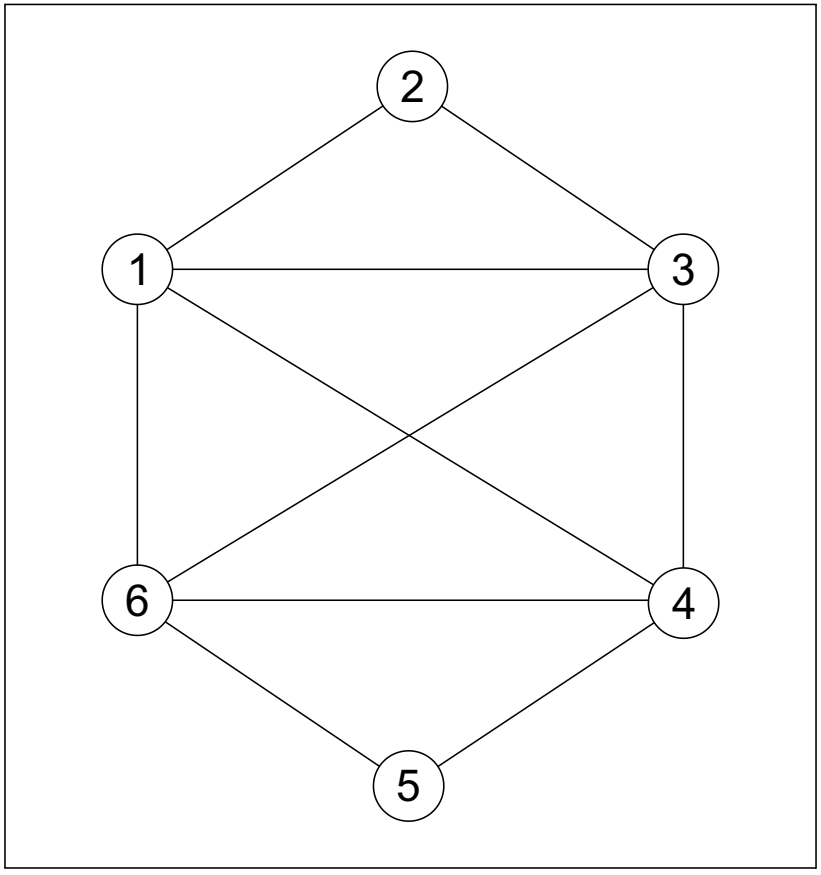
Algorytm wyznaczania cyklu Eulera

1. Wczytaj opis grafu jako listę krawędzi (u,v). Jeśli istnieją wierzchołki o nieparzystym stopniu — STOP.
2. Wybierz wierzcholek startowy u i zapisz go do stosu S.
3. Powtarzaj aż do opróżnienia stosu S:
   1. Jeśli z u wychodzi niewykorzystana krawędź (u,v), oznacz ją jako wykorzystaną, dodaj u do stosu S i u=v.
   2. W przeciwnym razie zapisz u do stosu T i u=pop(S);
4. Odczytaj rozwiązanie jako sekwencję kolejnych wierzchołków ze stosu T.

Algorytm wyznaczania cyklu Eulera

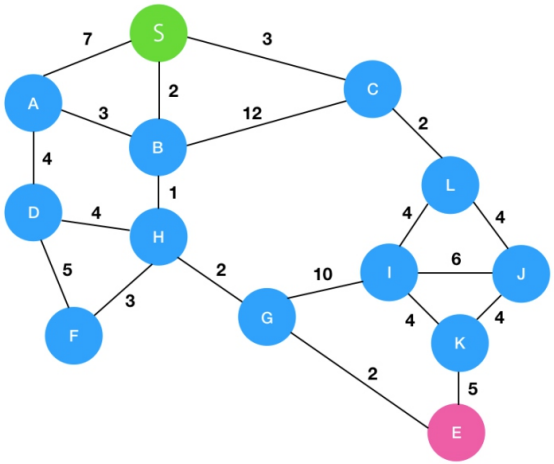
```
while (!eof(input)){ czytaj(u,v); D[u]++; D[v]++; }
for (u=1,...,n){if (D[u] nieparzyste) exit("brak rozwiązania");}
u = wierzchołek startowy;
do {
    if (istnieje niewykorzystana krawędź (u,v)) {
        push(S,u);
        u = v;
    } else {
        push(T,u);
        u = pop(S);
    }
} while (u!=0);
while (!empty(T)) print(pop(T));
```

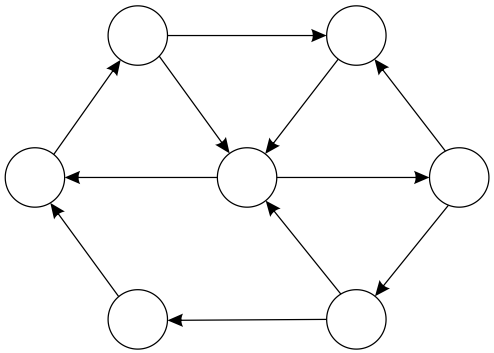
Wyznaczanie cyklu Eulera

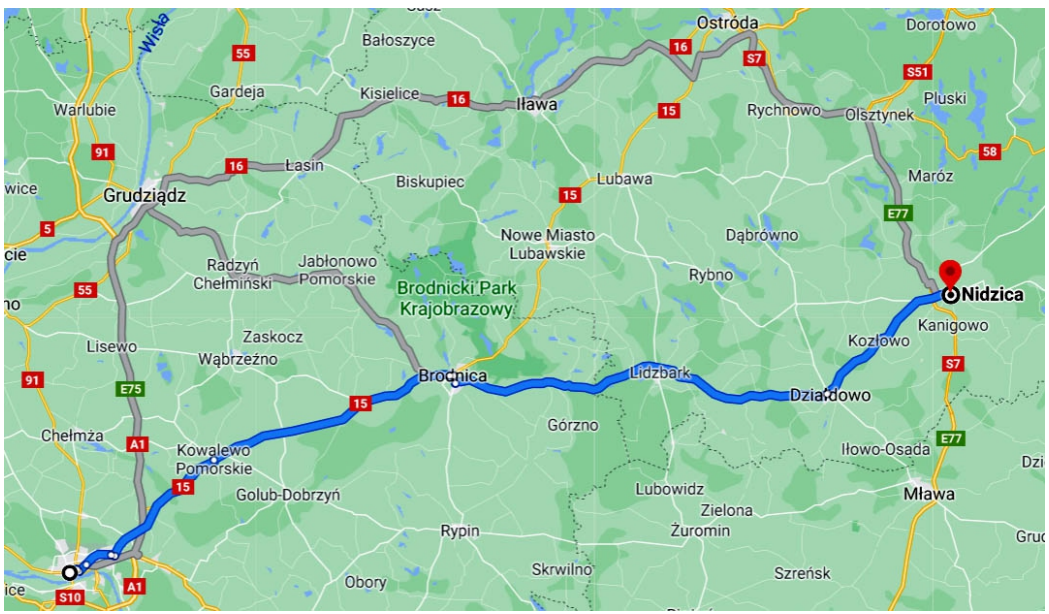


$$2 \to 3 \to 1 \to 2 \downarrow 4 \to 3 \to 6 \to 1 \downarrow 4 \to 5 \to 6$$

Najkrótsze drogi







Wyznaczanie najkrótszych dróg z wybranego wierzchołka s

1. Wczytujemy dane: wierzchołki 1,...,n, oraz odległości drogowe między nimi d(u,v). Brak połączenia drogowego oznaczamy jako $d(u,v)=\infty$.
2. Tworzymy kolejkę priorytetową Q wierzchołków u uporządkowaną wg długości D[u] najkrótszej znanej dotąd drogi z s do u. Początkowo $D[s]=0$, $D[u]=\infty$.
3. Stopniowo w pętli klasyfikujemy wierzchołki jako *odwiedzone*: kolejny wierzchołek przeznaczony do odwiedzenia pobieramy z kolejki Q – ma on najmniejszą wśród nieodwiedzonych wartość D[u].
4. Po pobraniu u z Q dokonujemy tzw. *relaksacji* wszystkich jego nieodwiedzonych sąsiadów v: $D[v]=\min\{ D[v], D[u]+d(u,v) \}$. Operecja ta wpływa na pozycję wierzchołków v w kolejce Q.
5. Przy relaksacji, jeśli $D[v]=D[u]+d(u,v)$, rejestrujemy $P[v]=u$. Tablica P przechowuje numer poprzednika u wierzchołka v na najkrótszej znanej dotąd drodze z s do v.
6. Gdy wszystkie wierzchołki zostaną odwiedzone, algorytm zatrzymuje się.
7. D[u] ma wartość optymalnej drogi z s do u. Przebieg tej drogi odczytujemy od końca z ciągu v, P[v], P[P[v]], ... , P[...[P[v]]...]=s.

Algorytm Dijkstry

```
while (!eof(input)) { czytaj(u,v,d); }
for (u=1,...,n) { D[u]=∞; P[u]=0; Odw[u]=0; pushq(Q,u,D[u]);}
czytaj(s); D[s]=0; modify(Q,s,D[s]);
while (!empty(Q)) {
    u = popq(Q); Odw[u]=1;
    for (v ∈ Sąsiedzi(u) && Odw[v]==0)
        if (D[v]>D[u]+d(u,v)){
            D[v]=D[u]+d(u,v);
            modify(Q,v,D[v]);
            P[v]=u;
        }
}
for (u=1,...,n){
    print("droga z s do u ma długość D[u]");
    v=u;
    while (v!=0) { print(v); v=P[v]; }
}
```