**Uniwersytet Warszawski**
Wydział Matematyki, Informatyki i Mechaniki

**Wojciech Dudek**
Nr albumu: 181255

# Praktyczny algorytm obliczania najliczniejszego skojarzenia w grafie

**Praca magisterska**
**na kierunku INFORMATYKA**

Praca wykonana pod kierunkiem
**dra hab. Krzysztofa Diksa**
Instytut Informatyki

Wrzesień 2004

Pracę przedkładam do oceny

Data Podpis autora pracy:

Praca jest gotowa do oceny przez recenzenta

Data Podpis kierującego pracą:

## Streszczenie

W pracy przedstawiono autorski algorytm o złożoności czasowej $O(\sqrt{n}m)$ dla problemu najliczniejszego skojarzenia w grafie. Obok opisu i dowodu poprawności podano implementację w języku C++. Omówione zostały także pewne heurystyki, które wydatnie przyspieszają działanie algorytmu.



## Słowa kluczowe

najliczniejsze skojarzenie, graf, ścieżka powiększająca, algorytm, złożoność obliczeniowa

## Klasyfikacja tematyczna

F.2.2 Analysis of Algorithms and Problem Complexity
G.2.2 Graph Theory

# Spis treści

# Wprowadzenie

Problem najliczniejszego skojarzenia w grafie jest uważany za jeden z ważniejszych w algorytmicznej teorii grafów. Jest on ściśle związany z wieloma innymi problemami, np. problemem stabilnych małżeństw, różnych reprezentantów, kolorowania krawędzi, maksymalnego przepływu.

Problem najliczniejszego skojarzenia został sformułowany i rozwiązany przez Claude'a Berge'a [1] w roku 1957. Podał on twierdzenie sprowadzające problem skojarzenia do sukcesywnego wyszukiwania ścieżek powiększających w grafie. Szukanie ścieżek powiększających jest relatywnie proste w grafach dwudzielnych, ale nie w grafach dowolnych. Dopiero w roku 1965 Edmonds [5] opublikował algorytm dla grafów dowolnych działający w czasie $O(n^4)$. (Jak pokazał w 1991 Blum [4] algorytm ten w rzeczywistości działał w czasie $O(n^3)$.) W swojej przełomowej pracy Edmonds wprowadził pojęcie kielicha (nieparzysty, naprzemienny cykl) i pokazał, w jaki sposób poprzez zwijanie kielichów można efektywnie znajdować ścieżki powiększające. Jeszcze w tym samym roku Wizgall i Zahn [19] podali modyfikację algorytmu Edmonds'a działającą w czasie $O(mn^2)$. W 1974 Kameda i Munro [12] ogłosili algorytm działający w czasie $O(mn)$, a w 1976 Lawler [14] i Gabow [8] podali własne algorytmy działające w trochę gorszym czasie – $O(n^3)$.

W roku 1973 Hopcroft i Karp [11] zaproponowali szukanie ścieżek powiększających w $O(\sqrt{n})$ fazach. W każdej fazie poszukiwano maksymalnego zbioru najkrótszych, rozłącznych wierzchołkowo ścieżek powiększających. Dodatkowo autorzy pokazali implementację fazy dla grafów dwudzielnych działającą w czasie $O(n + m)$ i w ten sposób otrzymali algorytm na skojarzenia o złożoności czasowej $O(\sqrt{n}m)$.

Nie trzeba było długo czekać, by powstały implementacje fazy dla grafów ogólnych. I tak: w 1975 Even i Kariv [6] osiągnęli czas $O(n^2)$ dla pojedynczej fazy, a w 1978 Bartnik podał alternatywny algorytm o tej samej złożoności czasowej. Wreszcie w 1980 udało się zbudować liniową implementację i osiągnąć czas $O(\sqrt{n}m)$ dla problemu najliczniejszego skojarzenia. Dokonali tego Micali i Vazirani [15], ale dowód Vazirani podał dopiero w roku 1994 [17]. Gabow i Tarjan [10] w 1989, a Blum [2, 3] w 1999 podali inne implementacje jednej fazy, działające też w czasie $O(m+n)$. (Bardziej wyczerpującą historię problemu podaje Galil [7].) Ciekawostką jest, że niedawno Mucha i Sankowski [16] przełamali barierę czasową $O(n^{2.5})$. Używając szybkiego mnożenia macierzy skonstruowali algorytm probabilistyczny działający w czasie $O(n^{2.38})$.

W tej pracy chcę pokazać mój własny algorytm rozwiązujący problem najliczniejszego skojarzenia w grafie. Motywacją mojej pracy był opis algorytmu Lawlera znajdujący się w [18]. Postanowiłem tak go zmodyfikować, by można było go połączyć z pomysłem $\sqrt{n}$ faz z pracy Hopcrofta-Karpa [11] i otrzymać w ten sposób oszacowanie czasowe $O(\sqrt{n}m)$. Dodatkowo opracowałem szybkie metody przetwarzania wstępnego, które wydatnie odciążają algorytm główny poprzez zmniejszanie wielkości danych wejściowych. Ostatecznie algorytm jest niezwykle szybki, a przy swojej prostocie okazuje się bardzo praktyczny.

Na początku (rozdz. 1) przedstawię problem, opiszę ogólny algorytm oraz redukcje problemu najliczniejszego skojarzenia do $O(\sqrt{n})$ faz poszukiwania maksymalnego zbioru najkrótszych ścieżek powiększających. Następnie w rozdziale 2 podam opis i dowód poprawności pojedynczej fazy algorytmu. W rozdziale 3 opiszę, w jaki sposób można przyspieszyć szukanie najliczniejszego skojarzenia wprowadzając przetwarzanie wstępne. Wreszcie w ostatniej części podam wyniki testów wydajnościowych dla omawianego algorytmu. Pełny kod źródłowy implementacji znajduje się na dołączonej do pracy płycie CD.

Chciałbym wyrazić podziękowania dla dra hab. Krzysztofa Diksa za wszelkie uwagi, motywowanie i za prowadzenie przedmiotu Algorytmika Praktyczna. Również podziękowania należą się wszystkim uczestnikom przedmiotu Algorytmika Praktyczna, których testy były użyte w mojej pracy. W szczególności chciałbym podziękować Krzysztofowi Onakowi, którego programu użyłem do porównywania wyników.

*pracę dedykuję*
*Jeshuah a Nozri'*

# Rozdział 1

# Definicje i ogólne metody

## 1.1. Definicje

Rozpoczniemy od podania podstawowych definicji i oznaczeń.

Niech $G = (V, E)$ będzie *grafem* składającym się ze skończonego zbioru wierzchołków $V$ i zbioru krawędzi $E$. Liczbę wierzchołków grafu oznaczamy przez $n$, liczbę krawędzi przez $m$. Graf $G$ może być skierowany lub nieskierowany. Dla grafu skierowanego (nieskierowanego) każda krawędź jest uporządkowaną (nieuporządkowaną) parą różnych wierzchołków. W obu przypadkach krawędź łączącą wierzchołki $x$ i $y$ oznaczamy jako $(x, y)$.

*Wersją nieskierowaną* skierowanego grafu $G = (V, E)$ nazywamy graf nieskierowany $G' = (V, E')$, który zawiera wszystkie krawędzie grafu $G$ z pominięciem skierowań.

*Wersją skierowaną* nieskierowanego grafu $G = (V, E)$ nazywamy graf skierowany $G' = (V, E')$, w którym każda krawędź nieskierowana $(u, v)$ grafu $G$ zostaje zastąpiona przez dwie krawędzie skierowane $(u, v)$ i $(v, u)$.

Graf jest *dwudzielny*, jeżeli zbiór wierzchołków $V$ może być podzielony na dwa rozłączne, niepuste zbiory $A$ i $B$ i takie, że dla każdej krawędzi $(u, v) \in E$, $u \in A$ i $v \in B$, lub na odwrót.

*Drogą* $W$ z $v$ do $w$ będziemy nazywać ciąg wierzchołków $v = v_1 \ldots v_k = w$ taki, że $k > 0$ i $(v_i, v_{i+1}) \in E$, dla $1 \leqslant i < k$. Droga jest prosta, jeżeli żaden wierzchołek nie występuje w $W$ więcej niż jeden raz. Często o drodze będziemy myśleli jako o ciągu krawędzi $(v_1, v_2), \ldots, (v_{k-1}, v_k)$. Każdej krawędzi możemy przypisać pewną wagę, która jest dodatnią liczbą całkowitą. Wszędzie tam, gdzie nie podano wartości wagi, będziemy przyjmować, że wynosi ona 1. Długością drogi jest suma wag jej krawędzi. Długość drogi $W$ oznaczamy przez $|W|$.

*Ścieżka* jest prostą drogą. Będziemy mówili, że ścieżka jest parzysta (nieparzysta) gdy jej długość jest parzysta (nieparzysta). Będziemy też mówili o parzystości ścieżki zamiast o parzystości długości ścieżki.

*Cykl* jest drogą $v_1, \ldots, v_k, v_1$ taką, że $k > 2$ i droga $v_1, \ldots, v_k$ jest prosta.

*Skojarzeniem* w grafie nieskierowanym nazywamy dowolny podzbiór $M$ zbioru jego krawędzi i taki, że żadne dwie krawędzie nie mają wspólnego wierzchołka. Skojarzenie w grafie skierowanym $G$ definiujemy jako skojarzenie w grafie będącym wersją nieskierowaną grafu $G$.

$M$ jest *maksymalnym skojarzeniem w sensie zawierania*, jeżeli nie istnieje taka krawędź $e \in E \setminus M$, że $M \cup \{e\}$ jest skojarzeniem.

$M$ jest *najliczniejszym skojarzeniem*, jeżeli jest skojarzeniem o największej możliwej liczności. Łatwo zauważyć, że najliczniejsze skojarzenie w grafie nie może zawierać więcej niż $\lfloor n/2 \rfloor$ krawędzi.

Dla danego nieskierowanego grafu $G = (V, E)$, *problem najliczniejszego skojarzenia* polega na znalezieniu najliczniejszego skojarzenia $M \subseteq E$.

Mówimy, że krawędź jest *skojarzona*, jeżeli należy do $M$, a *nieskojarzona*, jeżeli należy do $E \setminus M$. Wierzchołek $v$ jest *wolny (nieskojarzony)*, jeżeli każda krawędź wychodząca z $v$ jest nieskojarzona. Wierzchołek $v$ jest *skojarzony* z $u$, jeżeli krawędź $(u, v)$ jest skojarzona.

*Droga naprzemienna (ścieżka, cykl)* względem skojarzenia $M$, to droga $v_1, \ldots, v_k$, w której tylko co druga krawędź należy do $M$.

Ścieżkę naprzemienną z wierzchołka $u$ do wierzchołka $v$ będziemy oznaczali $u \to v$ albo $v \leftarrow u$. Będziemy używać strzałki $\hookleftarrow$ dla podkreślenia faktu, że ścieżka naprzemienna zaczyna się od krawędzi skojarzonej.

*Ścieżka powiększająca*, to ścieżka naprzemienna, której końce $v_1$, $v_k$ są wolne (pierwsza i ostatnia krawędź jest nieskojarzona). Jak łatwo zauważyć, ścieżki powiększające są zawsze nieparzystej długości.

Naprzemienna droga $v_1, \ldots, v_x, \ldots v_k, v_1$ jest *kielichem* wtedy i tylko wtedy, gdy $k$ jest nieparzyste oraz istnieje taki wierzchołek $v_x$, że skierowane drogi $(v_x, v_{x-1}) \ldots (v_2, v_1)$ i $(v_x, v_{x+1}) \ldots (v_k, v_1)$ są rozłącznymi krawędziowo ścieżkami.
*Kielich właściwy* [1] jest naprzemiennym, nieparzystym cyklem $v_1, \ldots, v_k, v_1$.
W obu definicjach *podstawą kielicha* jest wierzchołek $v_1$. Podstawę kielicha $C$ będziemy oznaczać przez $base(C)$. Łatwo zauważyć, że podstawa nie jest skojarzona z żadnym innym wierzchołkiem w kielichu.

*Krzak*, to skierowany graf, w którym każdy wierzchołek ma co najwyżej dwóch następników i istnieje dokładnie jeden wierzchołek, który nie ma następników - *korzeń krzaka*. Jak widać krzak jest uogólnieniem drzewa w ten sposób, że w drzewie wierzchołek ma co najwyżej jednego następnika, a nie dwóch. Zatem każde drzewo jest krzakiem. Zbiór krzaków będziemy nazywać *buszem*. Korzeń krzaka $T$ będziemy oznaczać $root(T)$. Będziemy pisać $T_x$ by wskazać krzak do którego należy wierzchołek $x$.

Niech $M$ będzie skojarzeniem. *Krzak naprzemienny* $T = (V, E)$ dla $M$ jest to krzak, w którym:

- korzeń krzaka jest nieskojarzony;
- każdy wierzchołek ma co najwyżej jedną krawędź wyjściową, która jest nieskojarzona;
- dla każdych dwóch wierzchołków $x, y \in V$ takich, że $(x, y) \in E$, istnieje naprzemienna ścieżka $x, y \to root(T)$;

Naprzemienną ścieżkę prowadząca do korzenia będziemy nazywali ścieżką powrotną.
Z definicji wynika, że każdy wierzchołek w krzaku naprzemiennym ma jedną albo dwie ścieżki powrotne, w zależności od stopnia wyjściowego wierzchołka. Co więcej, jeżeli wierzchołek ma dwie ścieżki powrotne, to jedna jest parzysta, a druga nieparzysta. Z powyższych dwóch obserwacji wynika następujący fakt.

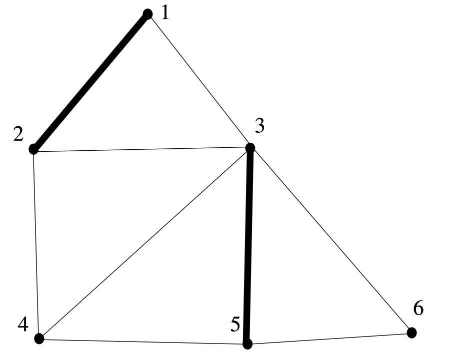
**Fakt 1.1** *Niech $l$ i $p$ będą ścieżkami powrotnymi w krzaku $T$. Jeżeli ścieżki $l$ i $p$ mają wspólną krawędź $(u, v)$, to mają wspólną ścieżkę $u, v \to root(T)$.*

Mówimy, że $l$ jest parzystości $k$, gdy $l \equiv k \pmod 2$.
*Sumę rozłączną (*`xor`*)* będziemy oznaczać przez $\oplus$.

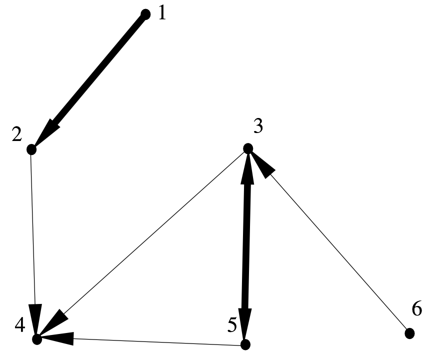
**Przykład 1** Na rysunku 1.1(a) krawędzie $(1, 2)$, $(3, 5)$ są skojarzone, a pozostałe nieskojarzone. Wierzchołki wolne to 4 i 6. Ścieżki naprzemienne, to np. $(1, 2, 3)$, $(1, 2, 3, 5)$, $(4, 2, 1)$.

[1] Kielich właściwy odpowiada definicji kielicha podanej przez Edmonds'a.

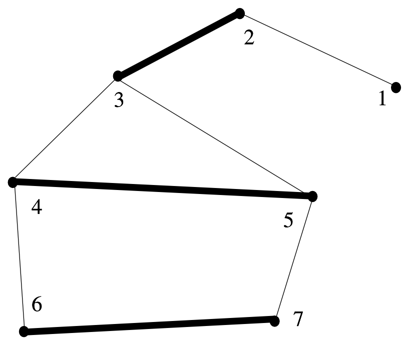
Natomiast ścieżkami powiększającymi są $(4,3,5,6)$ i $(4,2,1,3,5,6)$. Przykładem kielicha właściwego są $(4,5,3,4)$, $(4,5,3,1,2,4)$, $(3,2,3)$, i $(6,3,5,6)$. Graf z rysunku 1.1(c) ma dwa kielichy $(3,4,5,3)$ i $(3,4,5,7,6,4,5,3)$, przy czym tylko pierwszy z nich jest właściwy. Na rysunku 1.1(b) przedstawiono krzak naprzemienny o korzeniu w 4.




(a) przykładowy graf




(b) krzak naprzemienny




(c)

Rysunek 1.1: Rysunek dla przykładu 1

## 1.2. Twierdzenie Berge'a

Pokażemy teraz ogólny sposób podejścia do problemu najliczniejszego skojarzenia i udowodnimy jego poprawność. Zaczniemy od pewnej obserwacji.

**Lemat 1.2** *Niech $M_1$ i $M_2$ będą dowolnymi skojarzeniami w nieskierowanym grafie $G$. Wtedy $M_1 \oplus M_2$ jest sumą rozłącznych wierzchołkowo: ścieżek naprzemiennych oraz parzystych cykli naprzemiennych, zarówno względem $M_1$, jak i $M_2$.*

**Dowód:**
W $H = M_1 \oplus M_2$ są krawędzie, które są w $M_1$ lub w $M_2$, ale nie w obu na raz. Wierzchołek $v$ z $H$ jest incydentny albo z tylko jedną krawędzią z $M_1$, albo jedną krawędzią z $M_2$, albo jedną krawędzią z $M_1$ i jedną z $M_2$. Zatem stopień $v$ jest 1 lub 2. Czyli składowe grafu $H$ to, albo cykle parzystej długości, albo ścieżki, jedne i drugie naprzemienne względem $M_1$ i $M_2$. ■

**Twierdzenie 1.3 (Berge)** *Niech $G = (V, E)$ będzie nieskierowanym grafem i $M \subseteq E$ będzie skojarzeniem. Istnieje ścieżka powiększająca względem $M$ wtedy i tylko wtedy, gdy $M$ nie jest najliczniejsze.*

**Dowód:**
$\Rightarrow$ Niech ścieżką powiększającą będzie $p$ i rozważmy $H = M \oplus p$. Wszystkie wierzchołki skojarzone w $M$ są też skojarzone w $H$, bo $p$ jest naprzemienna. Dodatkowo $p$ jest powiększająca, zatem pierwszy i ostatni wierzchołek z $p$ są wolne w $M$, a w $H$ są skojarzone. Zatem $H$ jest skojarzeniem mocy większej niż $M$ i to dokładnie o 1.
$\Leftarrow$ Niech $M'$ będzie skojarzeniem większej mocy niż $M$. Zatem z lematu 1.2 wiemy, że $H = M \oplus M'$ jest sumą ścieżek i cyklów naprzemiennych względem $M'$ i $M$. W parzystych cyklach jest tyle samo krawędzi z $M$ co z $M'$; podobnie w parzystych ścieżkach. Jednak w $H$ jest więcej krawędzi z $M'$, zatem istnieje nieparzysta ścieżka naprzemienna $p$, której końce należą do $M'$, czyli $p$ jest powiększająca względem $M$. ■

Z powyższego twierdzenia mamy oczywisty wniosek:

**Wniosek 1.4** *Skojarzenie $M$ jest najliczniejsze wtedy i tylko wtedy, gdy w $G$ nie ma ścieżki powiększającej względem $M$.*

Z wniosku 1.4 wynika ogólny schemat algorytmu znajdowania najliczniejszego skojarzenia:

```
in: nieskierowany graf G i skojarzenie M (może być M = ∅)
out: najliczniejsze skojarzenie M

  while istnieje ścieżka powiększająca p względem M do
      M = M ⊕ p
```

**Alg. 1:** schemat Berge'a

Zatem problem najliczniejszego skojarzenia został sprowadzony do sukcesywnego wyszukiwania ścieżek powiększających. Od teraz będziemy zajmować się efektywnym sposobem owego wyszukiwania.

## 1.3. Twierdzenie Hopcrofta-Karpa

Podzielmy wyszukiwanie kolejnych ścieżek powiększających na fazy. W każdej fazie będziemy szukać zbioru *najkrótszych, wierzchołkowo rozłącznych ścieżek powiększających.* W ten sposób otrzymamy niezmiennik, że w każdej kolejnej fazie długość najkrótszej ścieżki powiększającej jest większa niż w poprzedniej fazie (lemat 1.6). Schemat algorytmu przedstawiamy w Alg. 2.

Dalej, jeżeli będziemy mówić o fazie, to będziemy rozumieć szukanie maksymalnego zbioru (w sensie zawierania) najkrótszych, wierzchołkowo rozłącznych ścieżek powiększających.

in: nieskierowany graf $G$ i skojarzenie $M$ (może być $M = \emptyset$)
out: najliczniejsze skojarzenie $M$

**while** istnieje ścieżka powiększająca $p$ względem $M$ **do**
  znajdź zbiór $N$ najkrótszych, wierzchołkowo rozłącznych ścieżek powiększających względem $M$
  dla każdego $p \in N$ wykonaj $M = M \oplus p$

**Alg. 2:** schemat Berge'a z fazami

**Lemat 1.5** *Niech $M$ będzie skojarzeniem, a $p$ najkrótszą ścieżką powiększającą względem $M$, $p'$ zaś ścieżką powiększającą względem $M \oplus p$. Zachodzi nierówność:*

$$|p'| \geqslant |p| + 2|p \cap p'|,$$

*gdzie $|p \cap p'|$ odnosi się do krawędzi, które są wspólne dla obu ścieżek.*

**Dowód:**
Niech $N = (M \oplus p) \oplus p'$, co jest równoważne $p \oplus p' = M \oplus N$. Z lematu 1.2 wiemy, że $M \oplus N$ składa się z rozłącznych wierzchołkowo cykli i ścieżek naprzemiennych. Ponieważ w każdym parzystym cyklu i w każdej parzystej ścieżce jest tyle samo krawędzi z $M$ co z $N$, i dodatkowo $N$ jest skojarzeniem większym od $M$ o 2, zatem istnieją dwie rozłączne nieparzyste ścieżki powiększające $p_1$, $p_2$ względem $M$. Co więcej, obie te ścieżki nie są krótsze od $p$, która była najkrótszą. Otrzymujemy równości:

$$|p \oplus p'| = |M \oplus N| \geqslant |p_1| + |p_2| \geqslant 2|p|$$

$$|p \oplus p'| = |p| + |p'| - 2|p \cap p'|$$

a po uproszczeniu:

$$|p'| \geqslant |p| + 2|p \cap p'|$$

■

**Lemat 1.6** *Rozważmy jedną fazę i niech $M$ będzie skojarzeniem przed tą fazą, natomiast niech $M'$ będzie skojarzeniem po wykonaniu tej fazy. Wtedy, o ile $M'$ nie jest najliczniejsze, najkrótsza ścieżka powiększająca względem $M'$ jest co najmniej o 2 dłuższa od najkrótszej ścieżki powiększającej względem $M$.*

**Dowód:**
Przypuśćmy, że w pewnej fazie powiększyliśmy bieżące skojarzenie $M$ o maksymalny zbiór $H$ najkrótszych rozłącznych ścieżek powiększających o długości $k$. Dostaliśmy nowe skojarzenie $M'$.
Jeżeli w $M'$ nie ma ścieżek powiększających, to mamy zgodnie z twierdzeniem 1.3 najliczniejsze skojarzenie. W przeciwnym wypadku istnieje ścieżka powiększająca $p'$ względem $M'$. Jeżeli $p'$ jest rozłączna wierzchołkowo z każdą ścieżką powiększającą z $H$, to musi mieć długość ostro większą niż $k$. (W przeciwnym wypadku $H$ moglibyśmy powiększyć dodając do niego $p'$, ale to przeczy jego maksymalności). Ponieważ ścieżki powiększające są nieparzystej długości, to $|p'| \geqslant k + 2$.
Jeżeli natomiast $p'$ ma wspólny wierzchołek z pewną ścieżką $p$ ze zbioru $H$, to jest to wierzchołek skojarzony, czyli $p$ i $p'$ mają wspólną krawędź. Wtedy z lematu 1.5 mamy: $|p'| \geqslant |p| + 2 = k + 2$. ■

Poniższe twierdzenie będzie potrzebne w dalszej części tej pracy:

**Twierdzenie 1.7 (Hopcroft-Karp)** *Wystarcza $O(\sqrt{n})$ faz by znaleźć najliczniejsze skojarzenie w $n$-wierzchołkowym grafie $G$.*

**Dowód:**
Załóżmy, że obliczamy najliczniejsze skojarzenie w fazach i wykonaliśmy już $x$ faz. Otrzymaliśmy skojarzenie $M$, które nie jest najliczniejsze. Z poprzedniego lematu wiemy, że od teraz każda ścieżka powiększająca ma długość co najmniej $2x+1$. Niech $M'$ będzie najliczniejszym skojarzeniem w $G$. Rozważmy $H = M' \oplus M$. Z lematu 1.2 wiemy, że jest to suma ścieżek i parzystych cykli naprzemiennych. W każdym parzystym cyklu (ścieżce) naprzemiennym jest tyle samo krawędzi z $M'$, co z $M$. Ponieważ $|M'| = |M| + k$, $k > 0$, istnieje $k$ rozłącznych nieparzystych ścieżek naprzemiennych względem $M$. Są to ścieżki powiększające, które zwiększają liczebność skojarzenia $M$ do $|M'|$. Dla ustalenia uwagi niech to będą ścieżki $p_1, ..., p_k$. Jeżeli teraz zsumujemy długości tych ścieżek, to wynik nie może być większy niż $n - k$, bo mamy $k$ składowych grafu $H$ o co najwyżej $n$ wierzchołkach. Otrzymujemy nierówność:

$$k(2x+1) \leqslant \sum_{i=1}^{k} |p_i| \leqslant n - k$$

po uproszczeniu:

$$k \leqslant \frac{n}{2x+2} \tag{1.1}$$

Jeżeli teraz za $x$ podstawimy $\sqrt{n}$, to otrzymamy, że po $\sqrt{n}$ fazach mamy do odnalezienia $\frac{n}{2(\sqrt{n}+1)} \leqslant \frac{\sqrt{n}}{2}$ ścieżek powiększających. Wystarczy dodatkowych $\frac{\sqrt{n}}{2}$ faz by to zrobić (bo w jednej fazie znajdujemy co najmniej jedną najkrótszą ścieżkę powiększającą). Zatem najliczniejsze skojarzenie można wyznaczyć w $O(\sqrt{n})$ fazach. ■

Jak już zostało powiedziane we wprowadzeniu, można podać algorytm implementujący pojedynczą fazę w czasie $O(n + m)$. Naszym celem jest przedstawienie własnej, efektywnej i praktycznej implementacji jednej fazy schematu Hopcrofta-Karpa.

# Rozdział 2

# Algorytm DMM

W tym rozdziale przedstawimy autorski algorytm znajdowania najliczniejszego skojarzenia w grafie – DMM (Dudek's Maximum Matching Algorithm). Jest on uszczegółowieniem algorytmu z fazami (patrz rozdział 1) i dlatego skoncentrujemy się na opisaniu pojedynczej fazy.

## 2.1. Opis algorytmu

### 2.1.1. Wprowadzenie

Rozpatrzmy nieskierowany graf $G = (V, E)$ z zadanym skojarzeniem $M \subseteq E$. *Ścieżka powrotna* z wierzchołka $x \in V$, to ścieżka naprzemienną od wierzchołka $x$ do dowolnego wierzchołka wolnego, którego będziemy nazywać *źródłem ścieżki powrotnej*. Szczególnie będzie nas interesować zbiór *najkrótszych parzystych ścieżek powrotnych* i zbiór *najkrótszych nieparzystych ścieżek powrotnych*. Zbiory te będziemy oznaczać odpowiednio $path_E(x)$ i $path_O(x)$. Długości ścieżek w tych zbiorach definiują głębokości wierzchołka $x$. Mianowicie głębokością parzystą $depth_E(x)$ jest długość ścieżek z $path_E(x)$, a głębokością nieparzystą $depth_O(x)$ jest długość ścieżek z $path_O(x)$. Jeżeli któryś ze zbiorów $depth_E(x)$, $depth_O(x)$ jest pusty, to odpowiadająca mu głębokość jest niezdefiniowana.

Jedna z głębokości wierzchołka $x$ musi być mniejsza i tą będziemy nazywać *pierwszą głębokością* $depth_1(x)$, a tą większą *drugą głębokością* $depth_2(x)$. Dodatkowo definiujemy zbiór $path(x) = path_E(x) \cup path_O(x)$. Każdą ścieżkę ze zbioru $path(x)$ będziemy nazywali *krótką ścieżką powrotną* z wierzchołka $x$. Zbiór $path(x)$ możemy podzielić na dwa rozłączne podzbiory $path_1(x)$ i $path_2(x)$. Pierwszy to ścieżki o długości $depth_1(x)$, a drugi to ścieżki o długości $depth_2(x)$.

Wśród ścieżek powrotnych niepustego zbioru $path_1(x)$ będziemy wyróżniać jedną ścieżkę $path_1^{Akt}(x)$. Tą ścieżkę powrotną będziemy nazywać *aktualną*. Pozostałe ścieżki powrotne z $path_1(x)$ będziemy nazywać *alternatywnymi*. W algorytmie DMM aktualną ścieżką powrotną jest najwcześniej znaleziona ścieżka ze zbioru $path_1(x)$. Podobnie definiujemy ścieżkę aktualną i ścieżki alternatywne dla $path_2(x)$.

Zauważmy, że dowolna ścieżka powiększająca $P = z \leftarrow u \rightarrow y$ w grafie $G$ składa się z dwóch rozłącznych wierzchołkowo ścieżek powrotnych z wierzchołka $u$, przy czym jedna jest ścieżką parzysta, a druga nieparzystą. Jak się później okaże, dla najkrótszych ścieżek powiększających zachodzi silniejsza własność. Mianowicie, każdą najkrótszą ścieżkę powiększającą można przedstawić jako $z \leftarrow u, v \rightarrow y$, gdzie $z \leftarrow u \in path(u)$, a $v \rightarrow y \in path(v)$, dla pewnych $u$ i $v$. Innymi słowy najkrótszą ścieżkę powiększającą da się podzielić na dwie krótkie ścieżki powrotne i krawędź.

Ideą algorytmu DMM jest obliczanie dla każdego wierzchołka $x$ jego głębokości, ścieżek $path(x)$ i odpowiednie sprawdzanie, czy z tych ścieżek można otrzymać ścieżkę powiększającą. Poszukiwania ścieżek powiększających są jednak tak synchronizowane, aby najpierw znajdować ścieżki najkrótsze.

### 2.1.2. Krzaki naprzemienne i ich rozbudowa

Zajmiemy się teraz sposobem szukania krótkich ścieżek powrotnych i wyznaczania głębokości wierzchołków. Przeglądając skierowaną wersję grafu $G$ budujemy krzaki naprzemienne (busz) względem skojarzenia $M$. Korzeniami krzaków są wierzchołki wolne. Każda ścieżka powrotna $x \to root(T)$ w dowolnym krzaku $T$ odpowiada aktualnej krótkiej ścieżce powrotnej ze zbioru $path(x)$. Długości ścieżek powrotnych z wierzchołka $x$ w krzaku $T$ są równe głębokościom $depth_E(x)$ i $depth_O(x)$. Oprócz ścieżek aktualnych będziemy też wyznaczać alternatywne ścieżki powrotne z $path(x)$.

Na początku każdy wierzchołek wolny tworzy jednowierzchołkowy krzak i otrzymuje głębokość parzystą równą 0. Rozbudowa dowolnego krzaka $T$ polega na przeglądaniu nierozpatrywanych krawędzi incydentnych z wierzchołkami krzaka, które otrzymały nową głębokość. Mianowicie, dla każdej nowej, nierozpatrywanej do tej pory głębokości nadanej pewnemu wierzchołkowi $x$ rozważamy nierozpatrywaną do tej pory krawędź incydentną $(y, x)$.

Jeżeli wierzchołek $y$ nie ma do tej pory nadanej głębokości, to wierzchołek $y$ i krawędź $(y, x)$ dodajemy do krzaka. Wierzchołkowi $y$ nadajemy głębokość o jeden większą niż odpowiednia głębokość wierzchołka $x$.

Zauważmy, że aby budowany krzak był naprzemienny, względem skojarzenia $M$, to *dla parzystej głębokości należy rozpatrywać jedynie krawędzie nie należące do skojarzenia, a nieparzystej głębokości tylko krawędź skojarzoną*. Powyższą zasadę przeglądania będziemy nazywać *regułą naprzemienności*.

### 2.1.3. Kielichy i ich obróbka

Załóżmy, że podczas rozbudowy krzaka naprzemiennego $T$ rozpatrujemy krawędź $(y, x)$, a oba wierzchołki $x$ i $y$ są już w krzaku $T$. Oba wierzchołki mają nadane pierwsze głębokości $depth_1(x)$ i $depth_1(y)$ oraz zostały znalezione odpowiadające tym głębokościom aktualne ścieżki krótkie.

Jeżeli oba wierzchołki mają głębokości o różnej parzystości, to ścieżka naprzemienna $P = y, x \to root(T)$ jest ścieżką powrotną w grafie $G$, której parzystość jest taka sama jak parzystość $depth_1(y)$. Jeżeli dodatkowo $P$ ma długość dokładnie $depth_1(y)$, to $P$ jest alternatywną ścieżką krótką.

Rozpatrzmy teraz przypadek kiedy oba wierzchołki mają głębokości o tej samej parzystości. W krzaku $T$ mamy zatem dwie ścieżki powrotne $L = x \to root(T)$ i $P = y \to root(T)$. Niech $W = w \to root(T)$ będzie najdłuższą ścieżką naprzemienną wspólną dla obu ścieżek powrotnych. Ścieżka $W$ jest parzysta, bo w przeciwnym wypadku wierzchołek $w$ byłby skojarzony z dwoma różnymi poprzednikami. Ze względu na fakt 1.1 ścieżki $x \to w$ i $y \to w$ są rozdzielne krawędziowo. Zatem naprzemienna droga $w \dots x, y \dots w$ jest kielichem w grafie $G$, a $w$ jest jego podstawą. Oznaczmy ten kielich jako $C$.

W ogólności wierzchołki grafu mogą należeć do wielu zagnieżdżających się kielichów. Mając dwa kielichy $C_1$ i $C_2$ mówimy, że $C_1$ jest *zewnętrznym* kielichem w stosunku do $C_2$, jeżeli $base(C_2) \in C_1$. *Najbardziej zewnętrzny* kielich, to taki względem którego nie istnieje kielich zewnętrzny.

W naprzemiennym kielichu $T$ budowanym na podstawie grafu $G$ definiujmy wartość $base(v)$ i $base^*(v)$ następująco:

$$base(v) = \begin{cases} v & v \text{ nie należy do żadnego ze znalezionych kielichów} \\ base(C) & v \text{ należy do znalezionego, najbardziej zewnętrznego kielicha } C \end{cases}$$

$$base^*(x) = base^k(x) \quad \text{dla takiego } k \text{, że: } base^k(x) = base^{k+1}(x)$$

Podczas przeglądania grafu $G$ będziemy aktualizować wartość *base* dla wszystkich wierzchołków z budowanego buszu i tym samym będziemy znali wartości *base*$^*$.

Powróćmy do analizowanego przypadku. Podstawową obserwacją jest, że wierzchołek należący do kielicha, z wyjątkiem wierzchołka będącego podstawą tego kielicha, ma dwie głębokości. Mianowicie, wierzchołek $z \in C \setminus base(C)$ ma już znalezioną krótką ścieżkę powrotną $z \to root(T)$. Natomiast drugą ścieżką powrotną jest $z \leftarrow x, y \to w \to root(T)$ dla $z \in L$ lub $z \leftarrow y, x \to w \to root(T)$ dla $z \in P$. Jest ona innej parzystości niż pierwsza ścieżka. W dowodzie algorytmu DMM pokażemy, że w rzeczywistości nowo znalezione ścieżki powrotne są ścieżkami krótkimi i wyznaczają drugą głębokość.

Obróbka kielicha polega na nadaniu wszystkim wierzchołkom w kielichu drugich głębokości i wyznaczeniu im aktualnych ścieżek krótkich, poprzez dodanie do krzaka naprzemiennego odpowiednich krawędzi. Przetwarzamy wierzchołki kielicha $C$ leżące na ścieżce $L$ i na ścieżce $P$. Dla przykładu rozpatrzmy przetwarzanie wierzchołków ze ścieżki $L$.

Niech $v_{q+1}, v_q \dots v_1, v_0$ będzie naprzemienną ścieżką taką, że $v_q \dots v_0$ jest kawałkiem ścieżki $L$, a $v_{q+1} = y$, $v_q = x$, $v_0 = w$. Przetwarzanie zaczynamy od $v_q$, a kończymy na wierzchołku $v_1$. Rozpatrzmy wierzchołek $v_s$. Naprzemienna ścieżka $F = v_s, v_{s+1} \dots v_q, y \to w \to root(T)$ jest ścieżką powrotną dla $v_s$. Możliwe są trzy przypadki:

1. $v_s$ ma nadaną tylko pierwszą głębokość i nie jest podstawą żadnego kielicha.
   Nadajemy drugą głębokość równą $|F|$. Dodajemy do krzaka $T$ krawędź $(v_s, v_{s+1})$

2. $v_s$ ma nadaną tylko pierwszą głębokość i jest podstawą pewnego kielicha $C' \neq C$.
   Nadajemy drugą głębokość równą $|F|$. Niech $v_t$ będzie ostatnim wierzchołkiem w ciągu $v_s, v_{s+1} \dots v_q$, który zawarty jest w kielichu $C'$. Dodajemy do krzaka $T$ specjalną krawędź $(v_s, v_{t+1})$ o wadze $|v_s \dots v_{t+1}|$ i z etykietą $v_t$.

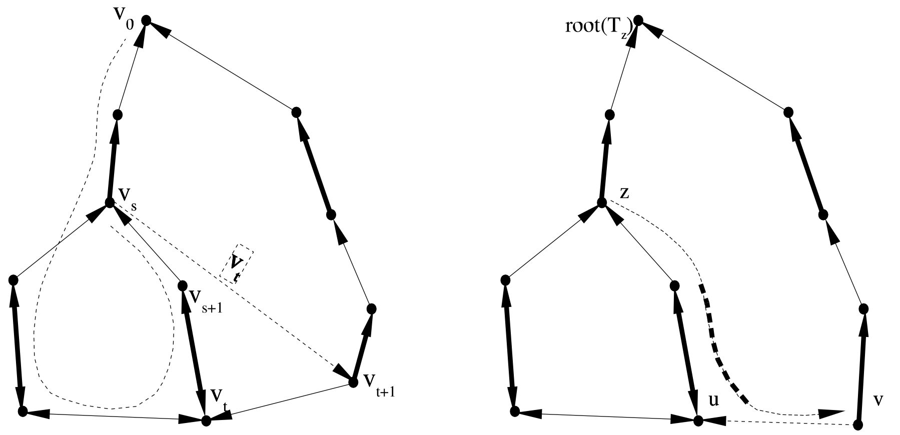
3. $v_s$ ma już nadane dwie głębokości.
   Jeżeli $|F|$ jest równe $depth_2(v_s)$, to $F$ jest alternatywną ścieżką krótką.

Następnie zmieniamy wartość $base(v_s)$ na $w$.

W przypadku 2 nie możemy dodać do krzaka krawędzi $(v_s, v_{s+1})$, gdyż spowodowałoby to stworzenie drogi naprzemiennej $v_s, v_{s+1} \to v_t \to v_s \to root(T)$, która nie jest ścieżką (patrz rysunek 2.1(a)). Jednakże, krawędź $(v_s, v_{t+1})$ jest „sztuczna”, gdyż nie koniecznie należy do grafu $G$. Mówi ona nam jedynie, że istnieje ścieżka naprzemienna $v_s \to v_{t+1}$ w grafie $G$. W algorytmie DMM, ze względów implementacyjnych, zamiast dodawać do buszu krawędź $(v_s, v_{t+1})$ z etykietą $v_t$ zapamiętujemy dla wierzchołka $v_s$ parę $(v_t, v_{t+1})$, którą będziemy nazywali mostem.

W ogólności most $(u, v)$ z wierzchołka $z$ będzie oznaczać, że ścieżka powrotna wygląda następująco $z \hookleftarrow u, v \hookrightarrow root(T_z)$, gdzie $T_z$ jest krzakiem naprzemiennym do którego należy wierzchołek $z$ (patrz rysunek 2.1(b)). Jest to ważne by potem móc odtworzyć ścieżkę powrotną w grafie $G$. Zauważmy, że długość ścieżki $z \hookleftarrow u$ jest równa $depth_E(u) - depth_E(z)$.

**Wniosek 2.1** *Niech $C$ będzie znalezionym kielichem, wtedy $depth_E(base(C)) = depth_1(base(C)) < depth_1(v) < depth_2(v)$ dla każdego $v \in C \setminus base(C)$.*




(a) Zła droga powrotna i specjalna krawędź

(b) Ścieżka powrotna przez most

Rysunek 2.1: Specjalna krawędź i most

**Wniosek 2.2** *Niech $C$ będzie znalezionym kielichem. Obie aktualne ścieżki powrotna dla wierzchołka $v \in C \setminus base(C)$ mają postać $v \to base(C) \to root(T_v)$, przy czym wierzchołki z $v \to base(C)$ należą do $C$.*

Kielich jest najważniejszym pojęciem występującym w algorytmach znajdowania najliczniejszego skojarzenia w dowolnych grafach. W grafach dwudzielnych kielichów nie ma. Pojawienie się kielichów wśród dróg naprzemiennych w dowolnym grafie jest tym, co komplikuje algorytm w porównaniu z przypadkiem grafów dwudzielnych.

### 2.1.4. Powiększanie skojarzeń

Pozostał do rozpatrzenia przypadek, w którym podczas rozbudowy krzaka naprzemiennego $T$ rozpatrywaną krawędzią jest $(y, x)$ i taka, że wierzchołek $x$ należy do $T$, a wierzchołek $y$ należy do innego krzaka naprzemiennego $T'$. Wynika z tego, że oba wierzchołki mają nadane pierwsze głębokości $depth_1(x)$ i $depth_1(y)$.

Jeżeli oba wierzchołki mają głębokości o różnej parzystości, to ścieżka naprzemienna $P = y, x \to root(T)$ jest ścieżką powrotną w grafie $G$, której parzystość jest taka sama jak parzystość $depth_1(y)$. Jeżeli dodatkowo $P$ ma długość dokładnie $depth_1(y)$, to $P$ jest alternatywną ścieżką krótką.

Rozpatrzmy teraz przypadek kiedy oba wierzchołki mają głębokości o tej samej parzystości. Ścieżki $x \to root(T)$ i $y \to root(T')$ są rozłączne, bo należą do różnych krzaków. Ścieżka $P = root(T) \leftarrow x, y \to root(T')$ jest zatem ścieżką powiększającą w grafie $G$ względem skojarzenia $M$. Zgodnie z Alg. 2 należy teraz skojarzenie $M$ zmodyfikować na $M \oplus P$. Jednakże wszystkie ścieżki aktualne w krzakach $T$ i $T'$ staną się wtedy nieaktualne. Musimy zatem wszystkie je usunąć, a dla wierzchołków z krzaków $T$ i $T'$ wybrać inne ścieżki aktualne ze zbioru ścieżek alternatywnych. Jeżeli wierzchołek nie ma już ścieżek alternatywnych, wtedy usuwamy go z buszu. Jest to tak zwane *topologiczne usuwanie*.

Z powyższych powodów nie możemy przypisywać wierzchołka na stałe do jednego krzaka. Jeżeli chcemy sprawdzić do jakiego krzaka należy wierzchołek idziemy wzdłuż aktualnej ścieżki powrotnej aż do źródła tej ścieżki. Kluczowy dla efektywności algorytmu jest fakt, że aktualna ścieżka powrotna przecinająca znaleziony wcześniej kielich musi przechodzić przez podstawę tego kielicha (wniosek 2.2). Obserwacja ta pozwala „przeskakiwać” kielichy w poszukiwaniu korzenia krzaka.

### 2.1.5. Kolejki

W celu uzyskania odpowiedniej synchronizacji w przeszukiwaniu grafu i budowaniu buszu będziemy używać dwóch kolejek priorytetowych: kolejkę przeszukiwania i kolejkę łączników.

Do kolejki przeszukiwania będziemy wstawiać krawędź $(v, u)$ i głębokość $d$ z priorytetem $2d$, jeżeli wierzchołek $u$ otrzymał nową głębokość równą $2d$.

Łącznikiem będziemy nazywać krawędź, której rozpatrzenie powoduje powstanie kielicha lub odnalezienie ścieżki powiększającej. Do kolejki łączników będziemy wstawiać łącznik $(v, u)$ i głębokość $d$ z priorytetem $p$, jeżeli rozpoznano łącznik $(v, u)$ podczas rozpatrywania głębokości $d$ nadanej wierzchołkowi $u$, natomiast długość drogi naprzemiennej $root(T_u) \leftarrow u, v \rightarrow root(T_v)$ jest równa $p$.

### 2.1.6. Synchronizacja pojedynczej fazy

Teraz już możemy przedstawić w całości fazę algorytmu DMM.

Mamy zadany graf $G(V, E)$ i skojarzenie $M \subseteq E$. Dla skierowanej wersji grafu $G$ budujemy busz $B$ względem skojarzenia $M$.

Na początku wierzchołkom wolnym nadajemy parzystą głębokość równą 0 i wstawiamy do kolejki przeszukiwania z priorytetem 0 wszystkie krawędzie incydentne do wierzchołków wolnych. Następnie w krokach wykonujemy na zmianę część Search i Connect, aż do opróżnienia obu kolejek – przeszukiwania i łączników.

Search:
W $k$-tym kroku rozpatrujemy krawędzie wchodzące do wierzchołków o głębokości $k$ wyjmując z kolejki przeszukiwania pewną krawędź $(v, u)$ i głębokość $k$ o priorytecie $2k$. Wierzchołek $u$ ma głębokość równą $k$. Może zdarzyć się kilka możliwości:

S1. **Wierzchołek $v$ nie ma nadanej żadnej głębokości.** Oznacza to, że nie był jeszcze odwiedzony i nie należy do żadnego krzaka. Aktualna ścieżka z $v$ wygląda następująco $v, u \rightarrow root(T_u)$. Dodajemy krawędź $(v, u)$ do buszu $B$. Nadajemy wierzchołkowi $v$ głębokość $k + 1$. Wszystkie krawędzie $(a, v) \in E$ spełniające zasadę naprzemienności wstawiamy wraz z głębokością $k + 1$ do kolejki przeszukiwania nadając im priorytet $2k + 2$.

S2. **Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k$ i $base^*(u) \neq base^*(v)$.** Oznacza to, że znaleźliśmy ścieżkę powiększającą lub kielich. Krawędź $(v, u)$ jest łącznikiem i zostanie rozpatrzona później w części Connect. Łącznik $(v, u)$ i głębokość $k$ wstawiamy z priorytetem $d + k + 1$ do kolejki łączników.

S3. **Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k + 1$ i nie ma nadanej głębokości o parzystości $k$.** Jeżeli $d = k + 1$, to ścieżka powrotna $v, u \rightarrow root(T_u)$ jest alternatywną ścieżką dla $v$.

S4. **Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k$ i $base^*(u) = base^*(v)$.**. Jeżeli $d = k + 1$, to ścieżka powrotna $v, u \rightarrow root(T_u)$ jest alternatywną ścieżką dla $v$. Jeżeli $k = d + 1$, to ścieżka powrotna $u, v \rightarrow root(T_v)$ jest alternatywną ścieżką dla $u$.

Connect:

Na końcu każdego kroku rozpatrujemy krawędzie z kolejki łączników o priorytecie $2k+1$, gdzie $k$ jest numerem kroku. Niech łącznikiem pobranym z kolejki będzie $(v, u)$, a towarzysząca mu głębokość wynosi $d$. Sprawdzamy czy krawędź $(v, u)$ nadal jest łącznikiem. Jeżeli nie, to pomijamy tą krawędź. (Krawędź przestaje być łącznikiem, jeżeli jeden z wierzchołków $u$ lub $v$ został usunięty lub wierzchołki $u$ i $v$ są w tym samym kielichu lub nie mają głębokości o parzystości $d$.) W przeciwnym wypadku sprawdzamy, do których krzaków naprzemiennych należą wierzchołki $u$ i $v$. Niech to będą odpowiednio $T_u$ i $T_v$. Mamy dwie możliwości:

C1. **Jeżeli** $T_u \neq T_v$, to znaleźliśmy ścieżkę powiększającą $P = root(T_u) \leftarrow u, v \rightarrow root(T_v)$ o długości $2k+1$. Usuwamy z buszu naprzemiennego $B$ wszystkie aktualne ścieżki powrotne, które przecinają się z $P$. Robimy to w następujący sposób. Przeszukujemy busz $B$ przy pomocy przeszukiwania BFS zaczynając od wierzchołków ze ścieżki $P$. Załóżmy, że podczas przeszukiwania rozpatrywany jest pewien wierzchołek $x$ do którego doszliśmy w $c$ krokach. Usuwamy aktualną ścieżkę powrotną o parzystości $c$ dla wierzchołka $x$. Jeżeli $x$ ma zapamiętaną alternatywną ścieżkę powrotną o parzystości $c$ i jest ona nie usunięta, to od tej pory ta ścieżka alternatywna staję się aktualną ścieżką powrotną dla $x$. Natomiast jeżeli wierzchołek straci wszystkie swoje krótkie ścieżki, to jest on usuwany z buszu $L$.

Ostatecznie zmieniamy bieżące skojarzenie $M$ na $M \oplus P$.

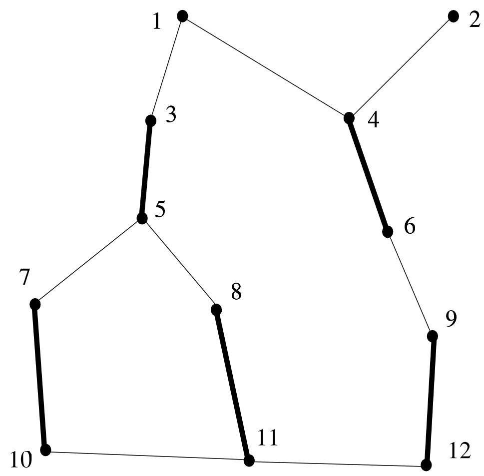
C2. **Jeżeli** $T_u = T_v$, to znaleźliśmy kielich $C$. Obrabiamy kielich $C$ tak, jak zostało to wcześniej opisane w podpunkcie 2.1.3. Niech $N$ będzie ciągiem wierzchołków, które podczas obrabiania kielicha otrzymały nową, parzystą głębokość. Ciąg $N$ jest uporządkowany zgodnie z kolejnością nadawania wierzchołkom nowych głębokości.

Dla każdego wierzchołka $i \in N$ o pewnej parzystej głębokości $depth_E(i)$ rozpatrujemy krawędź $(j, i) \in E$ taką, że $base^*(j) \neq base^*(i)$.

- Jeżeli wierzchołek $j$ ma parzystą głębokość, to krawędź $(j, i)$ i głębokość $depth_E(i)$ wstawiamy do kolejki łączników z priorytetem $depth_E(i) + depth_E(j)$.

  Jeżeli ponadto ścieżka $i, j \rightarrow root(T_j)$ jest alternatywną ścieżką dla wierzchołka $i$ (zachodzi to, gdy $depth_E(j) + 1 = depth_O(i)$), to łącznik $(j, i)$ jest wstawiany na początek grupy łączników, które są w kolejce i mają priorytet $depth_E(i) + depth_E(j)$.
- Jeżeli wierzchołek $j$ nie ma parzystej głębokości, to krawędź $(j, i)$ i głębokość $depth_E(i)$ wstawiamy do kolejki przeszukiwania z priorytetem $depth_E(i)$.

**Przykład 2** Prześledźmy działanie algorytmu na przykładzie grafu z rysunku 2.2. Do kolejki przeszukiwania wstawiamy wierzchołki wolne 1, 2. Następnie badamy krawędzie $(1, 3)$, $(1, 4)$. Nadajemy wtedy wierzchołkom 3, 4 głębokość 1. Krawędź $(2, 4)$ daje wierzchołkowi 4 alternatywną ścieżkę powrotną. Kolejne krawędzie: $(3, 5)$, $(4, 6)$, $(5, 7)$, $(5, 8)$, $(6, 9)$, $(7, 10)$, $(8, 11)$ $(9, 12)$ zostaną przebadane i wszystkie wierzchołki w grafie otrzymają swoje głębokości. Przetwarzanie krawędzi $(10, 11)$ spowoduje rozpoznanie łącznika i włożenie go do kolejki łączników. Podobnie się staje z $(11, 10)$, $(11, 12)$, $(12, 11)$. W części Connect wyciągamy łącznik $(10, 11)$. Okazuje się, że 10 i 11 są w tym samym krzaku, zatem mamy kielich. Nadajemy drugie głębokości wierzchołkom 10, 7, 11, 8 i ustalamy im wartość *base* na 5. Łącznik $(11, 10)$ pomijamy, bo 10 i 11 są w tym samym kielichu. Kolejny łącznik $(11, 12)$ tworzy kielich $1, 3, 5, 8, 11, 12, 9, 6, 4, 1$. Wierzchołek: 11 otrzymuje informacje o alternatywnej ścieżce




Rysunek 2.2: Przykład

powrotnej; 3 ma ścieżkę powrotną przez 5; 12, 9, 6, 4 też otrzymują swoje drugie głębokości. Wszystkim wierzchołkom ustawiamy *base* na 1. Podczas obsługi kielicha krawędź $(4, 2)$ zostanie włożona do kolejki łączników i w tym samym kroku zostanie rozpatrzona. W ten sposób znajdziemy ścieżkę powiększającą. Następnie wszystkie wierzchołki zostaną usunięte i algorytm się zatrzyma.

**Uwagi:**

1. W $k$-tym kroku wyjmujemy z kolejki przeszukiwania obiekty o priorytecie $2k$, a z kolejki łączników obiekty o priorytecie $2k + 1$. Zatem, w implementacji wystarczy jedna kolejka priorytetowa, która dla parzystych priorytetów będzie trzymała zawartość kolejki przeszukiwania, a dla nieparzystych zawartość kolejki łączników.

2. W S1 zamiast wstawiać do kolejki przeszukiwania wszystkie krawędzie $(a, v) \in E$ spełniające zasadę naprzemienności możemy wstawiać parę $(*, v)$. Po wyciągnięciu takiej pary z kolejki, będziemy sprawdzać wszystkie krawędzie $(a, v) \in E$ zgodnie z zasadą naprzemienności. W ten sposób zmniejszy się nieco ilość pamięci potrzebnej dla kolejki.

3. Na początku obróbki kielicha $C$ szukamy wierzchołka $w$ takiego, że $base(C) = w$. Jednakże, w implementacji jaką podamy będziemy liczyć $base^*(w)$, gdyż łatwiej jest znaleźć właśnie tą wartość. Okazuje się, że taka podmiana nie psuje algorytmu.

## 2.2. Struktury danych i procedury

W tym punkcie przedstawiono szczegółowy zapis algorytmu.

### 2.2.1. Struktury danych

W algorytmie używamy następujących struktur danych:

*mate* tablica skojarzeń; $mate[x]$ to wierzchołek skojarzony z $x$.

$bfsQ$ kolejka priorytetowa, zrealizowana na tablicy list. Wymagane operacje to: wstawianie do kolejki elementu z pewnym priorytetem ( push ), wstawianie na początek kolejki priorytetowej ( push_front ), wyjmowanie z kolejki elementu o najmniejszym priorytecie ( get ). Kolejka $bfsQ$ obsługuje kolejkę przeszukiwań i kolejkę łączników.

*bloom* implementacja struktury zbiorów rozłącznych. Wymagane operacje to: stwórz zbiór jednoelementowy ( init ), znajdź reprezentanta ( find ), dołącz do zbioru zbiór ( linkToSet ). Wartość *bloom*.find($x$) jest równa $base^*(x)$.

*depth* tablica głębokości; $depth[x][t]$ oznacza głębokość o parzystości $t$ dla wierzchołka $x$.

$from$ tablica list z informacją o ścieżkach powrotnych; zawiera następniki czyli albo wierzchołki, albo mosty. Pierwszy element listy $from[x][t]$ informuje o *aktualnej najkrótszej ścieżce powrotnej* dla $x$ o parzystości $t$, reszta listy to następniki *alternatywnych* ścieżki powrotnych o parzystości $t$.

*bridges* tablica list. Lista $bridges[x]$, to wierzchołki których ścieżki powrotne zaczynają się od ścieżki zbudowanej przez most.

Na listach określone są następujące operacje: wstawienie na koniec ( push_back ), wstawienie na początek ( push_front ), podanie wartości pierwszego elementu ( front ), usunięcie pierwszego elementu ( pop_front ), sprawdzenie czy lista jest pusta ( empty ). Listy przeglądamy zawsze w kierunku od pierwszego elementu do ostatniego.

Na kolejce FIFO określone są operacje: dodanie elementu ( push ), wyciągnięcie elementu ( get ), sprawdzenie czy kolejka jest pusta ( empty ).

Dla jasności prezentacji algorytmu używamy prostych makr:

`GUARD` wierzchołek atrapa.

`EVEN, ODD` wartość parzysta i nieparzysta — 0 i 1.

`TYPE(d)` parzystość głębokości $d$; zwraca `EVEN` lub `ODD`.

`OPOSITE(d)` parzystość głębokości $d+1$; zwraca `EVEN` lub `ODD`.

`HAS_TYPE(x,t)` czy wierzchołek $x$ ma głębokość o parzystości $t$

`IS_CONNECTOR(pr)` czy krawędź pobrano z kolejki łączników, gdzie *pr* to informacja zwracana przez kolejkę $bfsQ$.

`CONNECTORQ, SEARCHQ` kolejka łączników, kolejka przeszukiwania.

### 2.2.2. Procedury

AlterPath(*a*,*b*,*dep*) Sprawdza czy ścieżka $a, b \to root(T_b)$ o długości *dep* jest alternatywną ścieżką dla $a$. Jeżeli tak, to fakt ten zapamiętuje na liście $from[a]$.

Check(*u*,*v*,*depthU*,*pr*) Dla krawędzi $(v,u)$ i głębokości *depthU* rozpoznaje przypadki S1-S4, C1-C2. Zmienna *pr* informuje z której kolejki pobrano krawędź.

Init() Procedura inicjująca struktury.

Erase(*q*,*y*) Funkcja pomocnicza dla TopErase, usuwa nieparzyste ścieżki dla wierzchołka $y$.

TopErase($l1$,$l2$) Topologiczne usuwanie. Usuwa wszystkie aktualne ścieżki powrotne, które przecinają się z wierzchołkami znajdującymi się w listach $l1$, $l2$.

SuperBacktrack($u$,$v$,$t$) Idzie wzdłuż aktualnych ścieżek powrotnych o parzystości $t$ dla wierzchołków $u$ i $v$, „przeskakując" kielichy. Zwraca `GUARD` jeżeli wierzchołki są w różnych krzakach. Jeżeli $u$ i $v$ są w tym samym krzaku $T$, to zwraca wierzchołek $base^*(w)$ taki, że ścieżka $w \rightarrow root(T)$ jest najdłuższą ścieżką naprzemienną, wspólną dla obu ścieżek aktualnych z $u$ i $v$ o parzystości $t$.

AddEdges($l$) Dodaje krawędzie i głębokości do kolejek w sposób opisany w S2 dla wierzchołków w ciągu $l$.

RelabelBloom($x$,$y$,$type$,$depthBX$) Obróbka kielicha.

```
in: nieskierowany graf G = (V, E)
out: najliczniejsze skojarzenie M – mate

procedure AlterPath(a,b,dep)
    if depth[a][TYPE(dep)] = dep then from[a][TYPE(dep)].push_back(b)

procedure Check(u,v,depthU, pr)
    typeU =TYPE(depthU)
    if bloom.find(u) = bloom.find(v) then
        AlterPath(v, u, depth[u][EVEN] + 1)
        AlterPath(u, v, depth[v][EVEN] + 1)
    else if IS_CONNECTOR(pr) then
        if (not HAS_TYPE(u, typeU)) or (not HAS_TYPE(v, typeU)) then return
        w:=SuperBacktrack(u, v, typeU)
        if w =GUARD then
            niech l1 będzie aktualną ścieżką powrotną u o parzystości typeU
            niech l2 będzie aktualną ścieżką powrotną v o parzystości typeU
            TopErase(l1, l2)
            M := M ⊕ (l1 ⊕ l2 ⊕ (v, u))                         ▷ modyfikuje mate
        else
            l1:=RelabelBloom(u, v, w, TYPE(depthU + 1), depth[v][typeU] + 1)
            l2:=RelabelBloom(v, u, w, TYPE(depthU + 1), depth[u][typeU] + 1)
            AddEdges(l1)
            AddEdges(l2)
    else if HAS_TYPE(v, typeU) then
        bfsQ.push((u, v, depthU), depthU + depth[v][typeU]+1, CONNECTORQ)
    else if HAS_TYPE(v, OPOSITE(typeU)) then
        AlterPath(v, u, depthU + 1)
    else
        from[v][OPOSITE(depthU)].push_back(u)
        depth[v][OPOSITE(depthU)]:=depthU + 1
        bfsQ.push((v, *, depthU + 1), 2 * (depthU + 1), SEARCHQ)

Policz dowolne skojarzenie M                                  ▷ M może być puste
repeat
    Init()
    while not bfsQ.empty() do
        (u, v, depthU) = bfsQ.get(pr)          ▷ pr informuje z której kolejki pobrano
        if v ≠ * then
            Check(u, v, depthU, pr)
        else if depthU jest parzyste then
            for all x takiego, że (u, x) ∈ E jest krawędzią nieskojarzoną do
                Check(u, x, depthU, pr)
        else
            Check(u, mate[x], depthU, pr)
until nie znaleziono żadnej ścieżki powiększającej
```

**Alg. 3:** DMM

```
procedure Init
    bfsQ.makeEmpty()
    przywróć wszystkie skasowane wierzchołki
    for all x ∈ V do
        bloom.init(x)
        opróżnij listy from[x][ODD] i from[x][EVEN]
        if x nie jest skojarzone then
            from[x][EVEN].push_back(GUARD)
            depth[x][EVEN]:=0
            bfsQ.push((x, *, 0), 0,SEARCHQ)
```

**Alg. 4:** Init

```
function SuperBacktrack(u,v,t)
    typeU:=t; typeV:=t
    while u ≠ v and u ≠ GUARD do
        if depth[u][typeU] ⩾ depth[v][typeV] then
            u:=from[u][typeU].front()
            tmp:=bloom.find(u)
            if u = tmp then typeU:=OPOSITE(typeU)
            else typeU:=ODD; u:=tmp
        else
            v:=from[v][typeV].front()
            tmp:=bloom.find(v)
            if v = tmp then typeV:=OPOSITE(typeV)
            else typeV:=ODD; v:=tmp
    return u
```

**Alg. 5:** SuperBacktrack

```
procedure Erase(q,y)
    niech l będzie listą from[y][ODD]
    while (not l.empty())
        and (l.front() jest mostem or from[l.front()][EVEN].empty()) do
        l.pop_front()
    if l.empty() then
        q.push(mate[y])
        if from[y][EVEN].empty() then usuń wierzchołek y

procedure TopErase(l1, l2)
    niech q będzie kolejką FIFO
    for all x z l1 i l2 do
        q.push(x)
        opróżnij listę from[x][ODD]
    while not q.empty() do
        x:=q.get()
        opróżnij listę from[x][EVEN]
        if from[x][ODD].empty() then usuń wierzchołek x
        for all y takiego, że (x, y) ∈ E i from[y][ODD]=x do Erase(q,y)
        for all y z listy bridges[x] do Erase(q,y)
```

**Alg. 6:** TopErase

```
procedure AddEdges(l)
    for all i z listy l do
        d:=depth[i][EVEN]
        for all j takiego, że (i, j) ∈ E i bloom.find(i) ≠ bloom.find(j) do
            if HAS_TYPE(j,EVEN) then
                if depth[j][EVEN] + 1 = depth[i][ODD] then
                    bfsQ.push_front((i, j, d), d + depth[j][EVEN] + 1, CONNECTORQ)
                else
                    bfsQ.push((i, j, d), d + depth[j][EVEN] + 1, CONNECTORQ)
            else
                bfsQ.push((i, j, d), 2 * d, SEARCHQ)

function RelabelBloom(x,y,w,type,depthBX)
    niech l będzie pustą listą
    baseX:=bloom.find(x)
    while true do
        if x ≠ baseX then AlterPath(x, y, depthBX)
        if baseX = w then break
        if x ≠ baseX then
            bridges[y].push(baseX)
            type:=ODD
            y staje się mostem (x, y)
            depthBX += (depth[x][EVEN] − depth[baseX][EVEN])
        from[baseX][type].push_front(y)
        depth[baseX][type]:=depthBX
        bloom.linkToSet(baseX, w)
        if type =EVEN then l.push_back(baseX)
        type:=OPOSITE(type)
        y:=baseX
        x:=from[baseX][type].front()
        depthBX += 1
        baseX:=bloom.find(x)
    return l
```

**Alg. 7:** RelabelBloom i AddEdges

## 2.3. Dowód poprawności

**Lemat 2.3** *Niech $H = x \to v$ będzie ścieżką naprzemienną w grafie $G$, wierzchołek $x$ ma nadane dwie głębokości i $x$ otrzymał drugą głębokość podczas obróbki kielicha $C_x$. Niech dla każdego wierzchołka $b \in H$ zachodzi $depth_1(b) + depth_2(b) \leqslant 2k+1$, $b \notin r \to base(C_x)$ i wierzchołkowi $b$ nadano pierwszą głębokość. Wtedy krawędzie należące do $H$, a nie będące w budowanym buszu zostaną przetworzone jako łączniki najpóźniej w $k$-tym kroku, aż do napotkania łącznika ścieżki powiększającej.*

**Dowód:**
Wystarczy pokazać, że dla dowolnej krawędzi $(a, b)$ leżącej na ścieżce $H$, jeżeli wierzchołki $x \dots a$ otrzymały drugą głębokość najpóźniej w $k$-tym kroku, to $b$ też ją otrzyma.

Niech kielich $C$ będzie tym, podczas obróbki którego $a$ otrzymał drugą głębokość.

- Jeżeli krawędź $(a, b)$ jest skojarzona, to z faktu, że $a \in C \setminus base(C)$ wynika $b \in C \setminus base(C)$. Wierzchołek $b$ otrzymuje drugą głębokość razem z $a$.
- Jeżeli $(a, b)$ jest nieskojarzona, to albo $b \in C$ albo $b \notin C$. Rozpatrzmy przypadek kiedy $b \notin C$. Mamy $depth_1(a) + depth_2(a) = 2x+1 \leqslant 2k+1$ i $depth_1(b) + depth_2(b) = 2y+1 \leqslant 2k+1$

  Zachodzi $depth_1(b) = depth_E(b)$. Mamy teraz dwa przypadki.

  Jeżeli $depth_2(a) = depth_E(a)$, to mamy łącznik $(a, b)$. Policzmy jego priorytet. Wierzchołek $b$ nie należy do $path_2^{Akt}(a)$. Ścieżka $b, path_2^{Akt}(a)$ jest nieparzysta, zatem $depth_O(b) = depth_2(b) \leqslant depth_2(a) + 1$. Priorytet łącznika wynosi:

  $$depth_1(b) + depth_2(a) + 1 = 2y + 1 - depth_2(b) + depth_2(a) + 1 \leqslant 2y + 1 \leqslant 2k + 1.$$

  Zatem łącznik przetworzony będzie najpóźniej w $k$-tym kroku.

  Jeżeli $depth_1(a) = depth_E(a)$, to mamy łącznik $(a, b)$. Policzmy jego priorytet. Wierzchołek $b$ nie należy do $path_1^{Akt}(a)$. Ścieżka $b, path_1^{Akt}(a)$ jest nieparzysta, zatem $depth_O(b) = depth_2(b) \leqslant depth_1(a) + 1$. Priorytet łącznika wynosi:

  $$depth_1(b) + depth_1(a) + 1 = 2y + 1 - depth_2(b) + depth_1(a) + 1 \leqslant 2y + 1 \leqslant 2k + 1.$$

  Zatem łącznik przetworzony będzie najpóźniej w $k$-tym kroku.

  Jeżeli $b \in C$, to albo $base(C) \neq b$, albo $base(C) = b$. W pierwszym przypadku $b$ otrzymuje drugą głębokość razem z $a$. Zatem niech $base(C) = b$. Krawędź $(a, b)$ należy do $H$ i $C$. Niech $(t', t)$ będzie pierwszą krawędzią (idąc od $x$) należącą do $H$, taką że $t' \in C$ i $t \notin C$. Istnieje taka krawędź, bo jeżeli $C = C_x$, to mamy sprzeczność z $b \notin r \to base(C_x)$. Krawędź $(t', t)$ jest łącznikiem i była już przetworzona.

■

**Twierdzenie 2.4** *Mamy zadany graf $G$ i skojarzenia $M$. Niech $2t+1$ będzie długością najkrótszej ścieżki powiększającej w $G$ względem $M$. Dla $k = 0 \dots t$ w algorytmie DMM zachodzą następujące własności:*

**N0** *Wyznaczane drogi dla wierzchołków są ścieżkami krótkimi.*

**N1** *W części* Search *$k$-tego kroku wierzchołkowi, dla którego zachodzi $depth_1(v) = k+1$, zostaje nadana pierwsza głębokość.*

**N2** *Dla $k < t$, w części* Connect *$k$-tego kroku wierzchołek $v$ dla którego zachodzi $depth_1(v) + depth_2(v) = 2k + 1$ otrzymuje swoją drugą głębokość.*

**N3** *Dla $k = t$, w $k$-tym kroku wierzchołek dla którego zachodzi $depth_1(v) + depth_2(v) = 2k+1$ otrzymuje swoją drugą głębokość lub zostaje usunięty, gdy należy do ścieżki powiększającej, lub gdy wszystkie ścieżki z $path(x)$ zostały usunięte.*

**N4** *Jeżeli istnieje najkrótsza ścieżka powiększająca w $G$ względem $M$, o długości $2t + 1$, rozłączna wierzchołkowo z dotychczas znalezionymi ścieżkami $P_1 \dots P_p$, to algorytm w $t$-tym kroku wyznaczy jedną z nich.*

**Dowód:**
Dla dowodu N0, N1 i N2 prowadzimy indukcję ze względu na $k$.

N0 Dowód podajemy w lematach 2.5 i 2.6

N1 Krok inicjacji ma numer $-1$. Dla $k = -1$ niezmiennik jest prawdziwy. Procedura Init inicjuje na 0 głębokość wszystkich wierzchołków wolnych.

Krok indukcyjny: Z założenia mamy, że każdemu wierzchołkowi głębokość $\leqslant k$ została nadana najpóźniej w $k - 1$ kroku. Mianowicie, jeżeli dla pewnego wierzchołka $u$ mamy $depth_1(u) \leqslant k$, to wnioskujemy to z N1; jeżeli $depth_2(u) = k$, to mamy $depth_1(u) + depth_2(u) \leqslant 2k - 1$ i wnioskujemy to z N2.

Załóżmy teraz, że $(v, u)$ jest pierwszą krawędzią na ścieżce powrotnej z $v$ o długości $depth_1(v) = k + 1$. Wierzchołek $u$ ma głębokość równą $k$. Ponieważ w części Search w $k$-tym kroku DMM przeszukuje wszystkie krawędzie wchodzące do wierzchołków o głębokości $k$, krawędź $(v, u)$ zostanie rozpatrzona. Ze względu na N0, wierzchołek $v$ otrzyma poprawną głębokość.

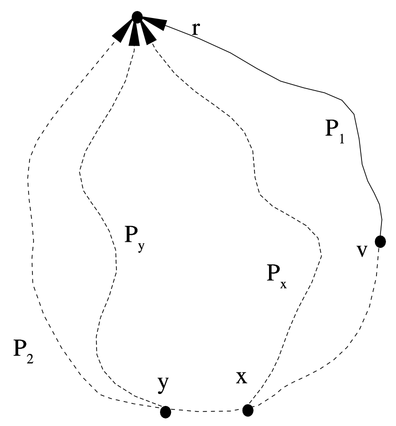
N2 Z definicji pierwszej głębokości wiemy, że $depth_1(v) \leqslant k$. Z własności N1, wierzchołek otrzymuje swoją pierwszą głębokość w części Search najpóźniej w $k - 1$-tym kroku. Niech $P_1 = v \xrightarrow{1} r$ będzie aktualną ścieżki powrotną $path_1^{Akt}(v)$.

Zakładając $2k + 1 < 2t + 1$, to wszystkie krótkie ścieżki powrotne wierzchołków dla których zachodzi $depth_1 + depth_2 \leqslant 2k + 1$ mają to same a $v$ mają źródło w $r$. W przeciwnym wypadku mielibyśmy ścieżkę powiększającą krótszą niż $2t + 1$.

Zatem wszystkie ścieżki z $path_2(v)$ mają źródło w $r$. Weźmy dowolną ścieżkę $P_2 = v \xrightarrow{2} r$ ze zbioru $path_2(v)$. Ze złączenia ścieżek $P_1$ i $P_2$ otrzymujemy drogę naprzemienną $r \xleftarrow{1} v \xrightarrow{2} r$ o długości $2k + 1$.

Na ścieżce $P_2$ znajdujemy dwa wierzchołki $x, y$ takie, że $|r \xleftarrow{1} v \xrightarrow{2} x| = k$ i $|r \xleftarrow{1} v \xrightarrow{2} x, y| = k+1$, czyli $|y \to r| = k$. Oczywiście, droga naprzemienna $r \xleftarrow{1} v \xrightarrow{2} x, y$ jest ścieżką. (W przeciwnym wypadku zawierałaby kielich i $v$ miałby drugą głębokość mniejszą niż $|P_2|$). Wierzchołki $x, y$ mają głębokości o parzystości $k$ i to w dodatku nie większe niż $k$. Zatem pierwsze głębokości wierzchołków $x, y$ zostają nadane w części Search najpóźniej w $k - 1$-tym kroku. Niech nadane wierzchołkom $x, y$ aktualne ścieżki powrotne o parzystości $k$ będą odpowiednio $P_x$ i $P_y$. Wierzchołki $x, y$ mają głębokości parzystości $k + 1$ i to w dodatku nie większe niż $k + 1$.

Zauważmy, że ścieżki $P_x, P_y$ mają to samo źródło $z$. W przeciwnym wypadku ścieżka $root(T_x) \leftarrow x, y \to root(T_y)$, byłaby ścieżką powiększającą krótszą niż $2t + 1$. Używając tego samego argumentu zauważamy, że $z = r$.



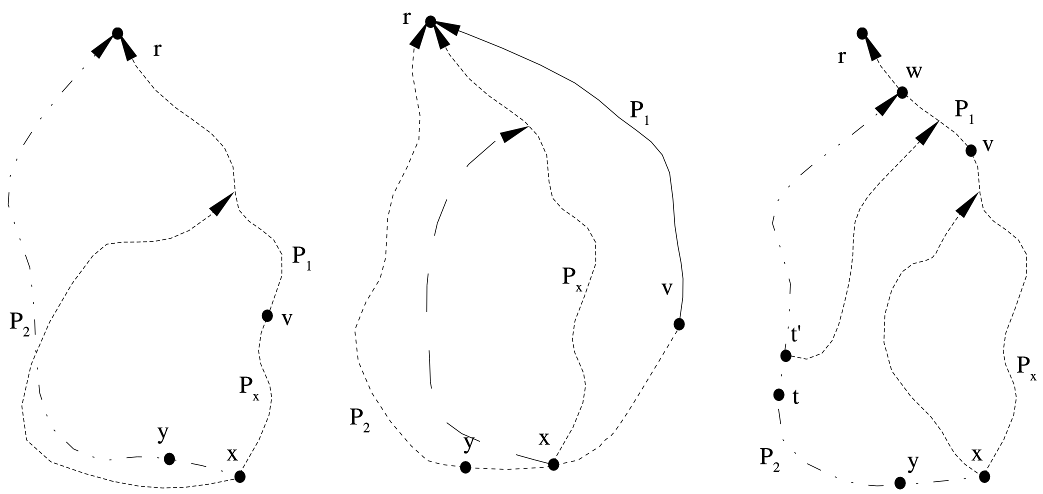

Rysunek 2.3: Ilustracja dla dowodu N2

Jeżeli jedna ze ścieżek $P_x$, $P_y$ jest krótsza od $k$, to $depth_1(x) + depth_2(x) \leqslant 2k - 1$ i $depth_1(y) + depth_2(y) \leqslant 2k - 1$, czyli z N2 mamy, że wierzchołkom $x, y$ nadano drugą głębokość najpóźniej w $k - 1$ kroku.

Sytuacja kiedy $P_x$ i $P_y$ są równe $k$ i obie są ścieżkami drugimi oznacza (patrz N2), że wierzchołkom $x, y$ nadano głębokość najpóźniej w $k - 1$ kroku.

Natomiast jeżeli $P_x$ i $P_y$ są równe $k$ i co najmniej jedna z nich jest pierwszą ścieżką powrotną, to krawędzie $(x, y)$, $(y, x)$ zostaną rozpoznane najpóźniej w części Search w $k$-tym kroku jako łączniki. W części Connect $k$ tego kroku łącznik $(x, y)$ lub $(y, x)$ spowoduje obróbkę kielicha. Wierzchołki $x, y$ otrzymają swoje drugie głębokości.

Pokazaliśmy zatem, że $x, y$ otrzymują drugie głębokości najpóźniej w $k$-tym kroku.




(a) (b) (c)

Rysunek 2.4: Rysunki dla dowodu N2

Niech $C_x$ będzie znalezionym kielichem takim, że wierzchołek $x$ otrzymał podczas obróbki $C_x$ drugą głębokość. Mamy trzy przypadki.

(a) Jeżeli $v \in C_x \setminus base(C_x)$, to $v$ otrzymał drugą głębokość razem z $x$, czyli najpóźniej w $k$-tym kroku.

(b) Rozpatrzmy przypadek kiedy $v \notin C_x$ i $v \notin base(C_x) \to r$. Dla każdego wierzchołka $a$ na ścieżce $v \xrightarrow{2} x$ zachodzi $depth_1(a) + depth_2(a) \leqslant 2k+1$. Z lematu 2.3 mamy, między innymi, że $v$ otrzyma swoją drugą głębokość najpóźniej w $k$-tym kroku.

(c) Pozostał przypadek kiedy $v \notin C_x$ i $v \in base(C_x) \to r$. Niech $(t, t')$ będzie pierwszą krawędzią na $P_2$ taką, że $v \notin path_1^{Akt}(t')$ i niech $w$ będzie pierwszym wierzchołkiem wspólnym dla ścieżek $path_1^{Akt}(t')$ i $base(C_x) \to r$. Zachodzi zatem $v \in path_1Akt(t)$. Wszystkie wierzchołki na $P_2$ mają już nadaną pierwszą głębokość ze względu na N1 i wszystkim zostanie nadana druga głębokość z godnie z lematem 2.3 najpóźniej w $k$-tym kroku. Między innymi krawędź $(t', t)$ zostanie włożona do kolejki łączników i zostanie przetworzony kielich $w \leftarrow t', t \to w$. Wierzchołek $v$ należy do tego kielicha i nie jest w nim podstawą zatem zostanie nadana mu druga głębokość.

N3 Dowód prowadzimy podobnie jak dla własności N2. Zachodzi $depth_1(v) \leqslant t$, $depth_2(v) \leqslant t+1$. Wierzchołek $v$ otrzymał już swoją pierwszą głębokość. Niech $P_1 = v \xrightarrow{1} r_1$ będzie aktualną ścieżki powrotną $path_1^{Akt}(v)$.

Weźmy dowolną ścieżkę $P_2 = v \xrightarrow{2} r_2$ ze zbioru $path_2(v)$. Ze złączenia ścieżek $P_1$ i $P_2$ otrzymujemy drogę naprzemienną $r_1 \xleftarrow{1} v \xrightarrow{2} r_2$ o długości $2t+1$.

Na ścieżce $P_2$ znajdujemy dwa wierzchołki $x, y$ takie, że $|r_1 \xleftarrow{1} v \xrightarrow{2} x| = t$ i $|r_1 \xleftarrow{1} v \xrightarrow{2} x, y| = t+1$, czyli $|y \to r_2| = t$. Oczywiście, droga naprzemienna $r_1 \xleftarrow{1} v \xrightarrow{2} x, y$ jest ścieżką. (W przeciwnym wypadku zawierałaby kielich i $v$ miałby drugą głębokość mniejszą niż $|P_2|$). Wierzchołki $x, y$ mają głębokości o parzystości $t$ i to w dodatku nie większe niż $t$. Zatem pierwsze głębokości wierzchołków $x, y$ zostają nadane w części Search najpóźniej w $t-1$ kroku. Niech nadane wierzchołkom $x, y$ aktualne ścieżki powrotne o parzystości $t$ będą odpowiednio $P_x$ i $P_y$, a źródła tych ścieżek $r_x$ i $r_y$. Wierzchołki $x, y$ mają głębokości parzystości $t+1$ i to w dodatku nie większe niż $t+1$.

Jeżeli $r_x \neq r_y$, to musi być $|P_x| = |P_y| = t$. Łącznik $(y, x)$ zostanie rozpoznany w $k$-tym kroku, a następnie spowoduje znalezienie ścieżki powiększającej $r_x \leftarrow x, y \to r_y$. Zatem albo wierzchołek $v$ zostanie usunięty jeżeli leży na tej ścieżce, albo ścieżka $P_2$ staje się nieaktualna, bo krawędź $(y, x)$ zostanie usunięta.

Jeżeli $r_x = r_y$, to wierzchołek $x$ i $y$ otrzymają swoje drugie głębokości najpóźniej w $t$-tym kroku. Argumentujemy to tak samo jak w dowodzie własności N2.

Niech $C_x$ będzie znalezionym kielichem takim, że wierzchołek $x$ otrzymał podczas obróbki $C_x$ drugą głębokość.

Mamy teraz dwa przypadki.

(a) Zachodzi $r_1 = r_x$. Jeżeli $v \in C_x \setminus base(C_x)$ lub $v \notin C_x$ i $v \notin base(C_x) \to r_1$, to argumentujemy tak samo jak w N2 (przypadek a i b), że $v$ otrzyma drugą głębokość najpóźniej w $t$-tym kroku.

Rozpatrzmy przypadek $v \notin C_x$ i $v \in base(C_x) \to r_1$. Niech $(t, t')$ będzie pierwszą krawędzią na $P_2$ taką, że $v \notin path_1^{Akt}(t')$, czyli $v \in path_1Akt(t)$ i obie ścieżki powrotna wierzchołka $t$ zawierają $v$. Jeżeli ścieżka $path_1^{Akt}(t')$ ma źródło równe $r_1$, to mamy sytuację taką samą jak w przypadku c dowodu N2. Jeżeli natomiast $path_1^{Akt}(t')$ ma inne źródło niż $r_1$, to krawędź $(t', t)$ zostanie włożona do kolejki łączników i zostanie znaleziona ścieżka powiększająca $root(T_{t'}) \leftarrow t', t \to r_1$. najpóźniej w $t$-tym kroku.

(b) Zachodzi $r_1 \neq r_x$. Z lematu 2.3 zostanie przetworzona cała ścieżka $v \to x$ i zostanie znaleziona ścieżka powiększająca $r_x \leftarrow x \to v \to r_1$ najpóźniej w $t$tym kroku.

N4 Niech $P_1' \dots P_r'$ będą ścieżkami powiększającymi w grafie $G$ względem skojarzenia $M$ każda o długości $2t + 1$ i każda rozdzielna wierzchołkowo z dotychczas znalezionymi ścieżkami $P_1 \dots P_p$. Weźmy dowolny wierzchołek wolny $v \in \bigcup_i P_i'$. Zachodzi $depth_1(v) + depth_2(v) = 0 + (2t + 1) = 2t + 1$. Zgodnie z N3 wierzchołek $v$ albo otrzyma drugą głębokość, albo zostanie usunięty razem ze ścieżką powiększającą o długości $2t + 1$ lub jeżeli zostaną usunięte wszystkie ścieżki krótkie dla $v$.

Wierzchołek otrzymuje drugą głębokość tylko podczas obrabiania kielicha. Wierzchołek $v$ jest nieskojarzony, zatem może być co najwyżej podstawą w kielichu. Nie otrzyma zatem drugiej głębokości.

Założymy, że wierzchołkowi $v$ usunięto wszystkie ścieżki. Zatem usunięto ścieżkę $v$, czyli $v \in \bigcup_i P_i$. Sprzeczność.

Zatem wierzchołek $v$ został usunięty razem z pewną ścieżką powiększającą.

■

Pozostało nam tylko udowodnić własność N0.

**Lemat 2.5** *Wyznaczane w części* Search *drogi są ścieżkami krótkimi.*

**Dowód:**
Załóżmy, że wierzchołkowi $u$ została nadana nowa głębokość o wartości $k$ i niech $u \to root(T_u)$ będzie aktualną ścieżką krótką o długości $k$. W części Search $k$-tego kroku (procedura CHECK) będzie rozpatrywana pewna krawędź $(v, u)$ z grafu $G$ zgodnie z zasadą naprzemienności. Dlatego droga $v, u \to root(T_u)$ jest naprzemienna. Mamy kilka przypadków, wszystkie przedstawiamy na rysunku 2.5

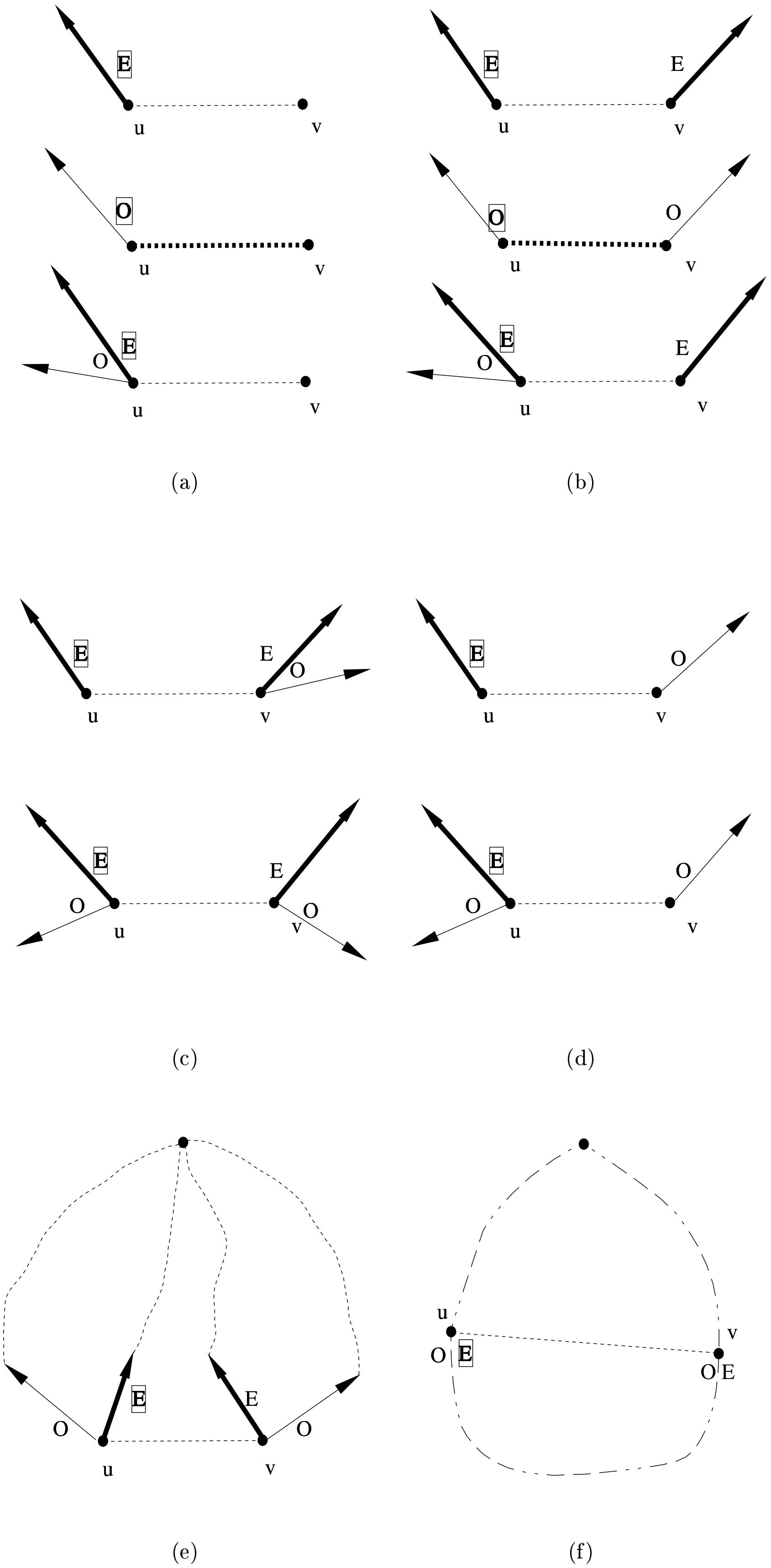
a) Wierzchołek $v$ nie ma żadnej nadanej głębokości (rys. 2.5(a)).

   Droga $P = v, u \to root(T_u)$ jest ścieżką, bo $v$ nie występuje w $u \to root(T_u)$. W przeciwnym razie $v$ otrzymałby głębokość mniejsza niż $k$ (patrz N1). Z tego samego powodu $P$ jest najkrótszą ścieżką powrotną. Przypadek ten jest rozpatrywany w Alg. 3.24-27.

b) Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k$ oraz $base^*(u) \neq base^*(v)$. (rys. 2.5(b) i 2.5(c)).

   Mamy dwie ścieżki $u \to root(T_u)$, $v \to root(T_v)$ o parzystości $k$, czyli droga $root(T_u) \leftarrow u, v \to root(T_v)$ jest naprzemienna. Krawędź $(u, v)$ jest łącznikiem.

c) Wierzchołek $v$ ma nadaną głębokość o parzystości $k + 1$ i nie ma nadanej głębokości o parzystości $k$ (rys. 2.5(d)).






Rysunek 2.5: Przypadki z dowodu lematu 2.5

Wierzchołek $v$ ma jedną głębokość. Załóżmy, że $depth_1(v) = depth_E(v)$, czyli $k$ jest nieparzyste. Krawędź $(v,u)$ zgodnie z zasadą naprzemienności jest skojarzona. Wierzchołek $v$ nie jest zatem korzeniem krzaka, czyli aktualna ścieżka krótka z $v$ ma postać $v, x \to root(T_v)$. Nie może być $x \neq u$, bo wtedy $v$ byłby skojarzony jednocześnie z $v$ i $x$. Jeżeli natomiast $x = u$, to krawędź $(v,u)$ była już rozpatrywana. Dochodzimy do wniosku, że $v$ nie może mieć parzystej głębokości, czyli ma nieparzystą, a krawędź $(u,v)$ jest nieskojarzona.

Teraz, jeżeli $depth_O(v) = k+1$, to droga naprzemienna $v, u \to root(T_u)$ jest nieparzystą ścieżką krótką. Mianowicie jeżeli przyjąć, że $v$ należy do aktualnej ścieżki powrotnej z $u$, to mielibyśmy $depth_O(v) < k$ – sprzeczność. Powyższy przypadek jest sprawdzany w Alg. 3.22-23.

Jeżeli $depth_O(v) < k+1$, to droga $v, u \to root(T_u)$ nie jest najkrótsza. Natomiast przypadek $depth_O(v) > k+1$ nie może zajść ze względu na N1.

d) Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k$ oraz $base^*(u) = base^*(v)$. Wierzchołkom $u, v$ nadano drugie głębokości podczas obróbki dwóch różnych kielichów $C_u$,$C_v$, ale $base(C_u) = base(C_v) = w$ (rys. 2.5(e)).

   Krawędź $(u,v)$ nie należy ani do $C_u$, ani do $C_v$, zatem jest to krawędź nieskojarzona. Co więcej, $C = w \hookleftarrow u, v \hookrightarrow w$ jest kielichem. i $(u,v)$ jest łącznikiem nowego kielicha.

   Jednakże wszystkie wierzchołki w kielichu $C$ mają już dwie głębokości i $base^*$ mają już zmienione na $w$. Jedynie co pozostaje, to sprawdzić czy wierzchołek $v$ może mieć ścieżkę alternatywną. Jeżeli $depth_0(v) = depth_E(u)+1$, to droga naprzemienna $v, u \to root(T_u)$ jest alternatywną ścieżką powrotną, bo w przeciwnym wypadku byłoby $v \in C_u$. Ten przypadek jest sprawdzany w Alg 3.5-6 i w razie powodzenia jest dodawana ścieżka alternatywna.

   Jeżeli $depth_O(v) < k+1$, to droga $v, u \to root(T_u)$ nie jest najkrótsza. Natomiast przypadek $depth_O(v) > k+1$ nie może zajść ze względu na N1.

e) Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k$ oraz $base^*(u) = base^*(v)$. Wierzchołkom $u$, $v$ nadano drugie głębokości podczas obróbki kielicha $C$ i z tego powodu $base^*(u) = base^*(v)$ (rys. 2.5(f)).

   Z definicji wszystkie wierzchołki w kielichu, oprócz podstawy, są skojarzone, natomiast podstawa nie jest skojarzona z żadnym z wierzchołków kielicha. Krawędź $(u,v)$ nie należy do kielicha $C$ (jakby należała, to byśmy jej drugi raz nie przeglądali), zatem jest to krawędź nieskojarzona. Pokażemy, że żadna krótka ścieżka nie przechodzi przez $(u,v)$.

   Jeżeli $depth_E(u) = depth_1(u)$, to z N1 i N2 wiemy, że krawędź $(u,v)$ została sprawdzona w części Search kroku $depth_1(u)$, a kielich $C$ przetwarzany był w części Connect w kroku $(depth_1(u) + depth_2(u) - 1)/2 \geqslant depth_1(u)$. Czyli $C$ jeszcze nie istnieje - sprzeczność.

   Jeżeli $depth_E(u) = depth_2(u)$ i $depth_O(v) = depth_1(v)$, to mamy $depth_2(u) > depth_1(v)$. (W przeciwnym wypadku równość $depth_1(u) + depth_2(u) = depth_1(v) + depth_2(v)$ nie byłaby spełniona, a przecież to prosty wniosek z N2.) Zatem takie założenie nie daje żadnej najkrótszej drogi.

   Dla $depth_E(u) = depth_2(u)$ i $depth_O(v) = depth_2(v)$ rozpatrzmy najkrótszą parzystą ścieżkę powrotną z $u$. Przechodzi ona albo przez $v$, wtedy droga $v, u \hookrightarrow v \to root(T_u)$ nie jest ścieżką; albo nie przechodzi przez $v$, wtedy krawędź $(u,v)$ należy do kielicha – sprzeczność.

f) Wierzchołek $v$ ma nadaną głębokość $d$ o parzystości $k$ oraz $base^*(u) = base^*(v)$. Jeden z wierzchołków $u$, $v$ ma tylko jedną głębokość. Zatem albo $base^*(u) = u$, albo $base^*(v) = v$. Pociąga to za sobą, że jeden z wierzchołków zmieniono wartość *base* i nadano mu drugą głębokość podczas obróbki pewnego kielicha. Krawędź $(v, u)$ jest wiec nieskojarzona.

   Jeżeli $base^*(u) = u$, to $depth_E(u) = depth_1(u)$. Krawędź $(u, v)$ została sprawdzona w części Search kroku $depth_1(u)$, a wierzchołkowi $v$ nadano drugą głębokość w części Connect w kroku $(depth_1(u) + depth_2(u) - 1)/2 \geqslant depth_1(u)$. Czyli $v$ jeszcze nie nadano drugiej głębokości – sprzeczność.

   Jeżeli $base^*(v) = v$, to korzystamy z wniosku 2.1 i mamy $depth_2(u) > depth_1(u) > depth_1(v)$. Nie zachodzi zatem $d = k + 1$.

■

**Lemat 2.6** *Po obróbce kielicha $C$ w $k$-tym kroku, każdy wierzchołek $v \in C \setminus base(C)$ ma nadaną drugą głębokość i ustaloną ścieżkę ze zbioru $path_2(v)$.*

**Dowód:**
Niech $C = w \leftarrow x, y \rightarrow w$ będzie kielichem, gdzie krawędź $(u, v)$ jest łącznikiem o priorytecie $2k + 1$. Niech $L = x \rightarrow w \rightarrow root(T_w)$, $P = y \rightarrow w \rightarrow root(T_w)$, będą aktualnymi ścieżkami krótkimi odpowiednio dla wierzchołka $x$ i $y$. Zatem $|root(T_w) \leftarrow w \leftarrow x, y \rightarrow w \rightarrow root(T_w)| = 2k + 1$. Jeżeli teraz weźmiemy sobie dowolny wierzchołek $v \in C \setminus base(C)$, to $v$ należy do jednej ze ścieżek $L$ albo $P$. Stąd ścieżka $v \rightarrow w \rightarrow root(T_w)$ jest aktualną ścieżką krótką, a jeżeli $v$ nie miał wcześniej nadanej drugiej głębokości, to jest ścieżką pierwszą. Dla ustalenia uwagi niech $v \in L$ i $v$ nie miał do tej pory drugiej głębokości.

Droga naprzemienna $Q = v \leftarrow x, y \rightarrow w \rightarrow root(T_w)$ jest ścieżką. Mianowicie, niech $u$ będzie tak jak $v$, ale w dodatku najbliższym $y$ takim, że $u \leftarrow y \rightarrow w \rightarrow root(T_w)$ nie jest ścieżką. Jeżeli $u \in w \rightarrow root(T_w)$, to sprzeczność, bo $u \in C \setminus base(C)$. Jeżeli $u \in y \rightarrow w$, to albo $u$ jest podstawą kielicha, albo $depth_1(u) + depth_2(u) < 2k + 1$ – ma już znalezioną drugą głębokość.

Co więcej, $Q$ jest nie dłuższa niż $depth_2(v)$. W przeciwnym wypadku: $2k+1 = |root(T_w) \leftarrow w \leftarrow v \leftarrow x, y \rightarrow w \rightarrow root(T_w)| = depth_1(v) + |Q| < depth_1(v) + depth_2(v)$. Wierzchołek $v$ ma znalezioną drugą głębokość – sprzeczność. ■

**Twierdzenie 2.7** *Algorytm DMM znajduje najliczniejsze skojarzenie w grafie.*

**Dowód:**
Ze względu na N4, w pojedynczej fazie zostaną znalezione wszystkie rozłączne najkrótsze ścieżki powiększające. Algorytm DMM szuka ścieżek powiększających dopóki ich już nie ma, zatem na podstawie twierdzenia 1.3 znajduje najliczniejsze skojarzenie. ■

## 2.4. Oszacowanie złożoności

Każda skierowana krawędź grafu jest przeglądana co najwyżej 6 razy. W procedurze CHECK krawędź skierowaną $(a, b)$ może być rozpatrzona raz w części Search i raz w części Connect.

Podczas RelabelBloom krawędź jest przeglądana tylko raz, podczas konstruowania ścieżki powiększającej raz oraz w TopErase też raz.

Przeanalizujmy czas zużyty przez SuperBacktrack. Rozpatrzmy kielich $C$, przetwarzany w $k$-tym kroku. Oczywiście wszystkie aktualne ścieżki powrotne wierzchołków z $C$ przechodzą przez podstawę kielicha. Niech $x \in C$ będzie wierzchołkiem, który ma zapisaną „na boku" alternatywną ścieżkę powrotną $x, y \ldots$ i będzie miał największą drugą głębokość. Ze względu na sposób wkładania łączników do kolejki łączników (Alg 7.3-10) krawędź $(x, y)$ będzie rozpatrzona natychmiast po przetworzeniu kielicha $C$. Jeżeli $(x, y)$ jest łącznikiem ścieżki powiększającej, to TopErase usunie cały kielich. Jeżeli $(x, y)$ jest łącznikiem pewnego kielicha, to alternatywna ścieżka powrotna ma część wspólną ze ścieżką powrotną $base(C) \to root(T_{base(C)})$. Czyli jeżeli się zdarzy, że usuwana ścieżka powiększająca przecina $C$, to wierzchołek $x$ też zostanie usunięty.

Zatem SuperBacktrack będzie przeglądał krawędź $(a, b)$ tylko raz. Albo ta krawędź leży na ścieżce powiększającej i zostanie usunięta, albo należy do pewnego kielicha i wtedy będzie pomijana.

Całkowity koszt obsługi struktur danych takich jak lista i tablica jest liniowy. Strukturę dla zbiorów rozłącznych (*bloom*) też można zaimplementować tak, żeby czas był liniowy (patrz [9]). Koszt inicjalizacji (patrz Init) fazy wynosi $O(n)$. Ostatecznie jedna faza algorytmu działa w czasie $O(n + m)$. Zgodnie z twierdzeniem 1.7 złożoność czasowa DMM wynosi $O(\sqrt{n}m)$.

# Rozdział 3

# Przyspieszające heurystyki

W tym rozdziale zastanowimy się, jak zbudować dowolne skojarzenie $M$ w grafie $G$ w krótkim czasie i tak, żeby liczność skojarzenia była jak największa. Metoda taka może okazać się bardzo pomocna jako przetwarzanie wstępne w algorytmie DMM (patrz Alg. 3.28).

Zaczniemy od pewnej obserwacji.

**Lemat 3.1** *Każdy nieizolowany wierzchołek może być skojarzony względem najliczniejszego skojarzenia.*

**Dowód:**
Niech $v$ będzie nieizolowanym wierzchołkiem i $M$ będzie dowolnym najliczniejszym skojarzeniem. Jeżeli $v$ jest skojarzony w $M$, to jest dobrze. Rozpatrzmy zatem przypadek kiedy $v$ jest wolny.

Wierzchołek $v$ ma incydentną krawędź $(v, u)$, przy czym $u$ jest skojarzone. W przeciwnym wypadku $(v, u)$ powiększa najliczniejsze skojarzenie $M$ - sprzeczność. Niech wierzchołkiem skojarzonym z $u$ będzie $w$. Skojarzenie $M' = M \oplus \{(v, u), (u, w)\}$ jest tak samo liczne jak $M$.
■

Nasuwa się oczywisty wniosek:

**Wniosek 3.2** *Ażeby znaleźć najliczniejsze skojarzenie w grafie $G$, wierzchołek $v$ stopnia* 1 *można jednoznacznie skojarzyć z $u$ i rozpatrzyć problem najliczniejszego skojarzenia w grafie $G$ bez $u$ i $v$.*

Taka obserwacja pozwala zbudować poniższy algorytm.

in: nieskierowany graf $G = (V, E)$
out: skojarzenie $M$

Niech $X = \{x \in V : deg(x) = 1\}$
**while** istnieje nieizolowany wierzchołek w $G$ **do**
  **if** istnieje wierzchołek $v \in X$ o stopniu 1 **then**
    skojarz $v$ z jego sąsiadem $u$
    usuń $u$,$v$ z $G$ i $X$
  **else**
    skojarz dowolne dwa wierzchołki $v$,$u$ i usuń je
  wstaw do $X$ sąsiadów $u$,$v$, którzy mają stopień równy 1

**Alg. 8:** H0

Jeżeli zawsze będzie spełniony warunek z 3-go wiersza to skojarzenie $M$ będzie najliczniejsze. Dzieje się tak na przykład dla grafów, które są drzewami.

Czas działania H0 jest oczywiście liniowy. Przeanalizujmy teraz o ile algorytm się może pomylić, jeżeli kojarzy wierzchołki o stopniu większym od 1. Niech $M$ będzie skojarzeniem w $G$, w którym H0 źle skojarzył wierzchołek $v$ w 7. wierszu. Niech krawędzią skojarzoną będzie $(u, v)$. Zatem nie istnieje najliczniejsze skojarzenie, w którym krawędź $(u, v)$ jest skojarzona. Weźmy dowolne najliczniejsze skojarzenie $H$, w którym $v$ jest skojarzone z $w$. Na podstawie lematu 1.2 $M \oplus H$ jest zbiorem ścieżek powiększających i parzystych cykli. Przez krawędzie $(u, v), (v, w)$ przechodzi dokładnie jedna ścieżka powiększająca, bo parzysty cykl nie zwiększa skojarzenia. Najkrótsze ścieżki powiększające w $M \oplus H$ mają długość 3. Czyli w najgorszym przypadku, w każdej ścieżce powiększającej ze zbioru $M \oplus H$ mamy jedną krawędź z $M$ i dwie z $H$. Innymi słowy, na jedną skojarzoną krawędź z $M$ przypada co najwyżej dwie skojarzone krawędzie z $H$.

**Wniosek 3.3** *Jeżeli najliczniejsze skojarzenie w grafie $G$ wynosi $z$, to algorytm H0 konstruuje skojarzenie o mocy co najmniej $z/2$.*

Zauważmy, że efektywność H0 jest nie mniejsza niż efektywność pierwszej fazy algorytmu DMM. W obu przypadkach znajdowane jest skojarzenie mniej liczne o co najwyżej $n/4$ od najliczniejszego skojarzenia. Przewagą H0 jest to, że jest tańszy, bo przegląda krawędź grafu tylko raz i kojarzy prawidłowo wierzchołki o stopniu 1.

Okazuje się, że w praktyce algorytm H0 można poprawić wybierając w 7. wierszu nie dowolną krawędź, ale taką której wierzchołki mają najmniejszy stopień. W ten sposób „odkładamy na później" wierzchołki, które mają duży stopień, a tym samym większą szanse na skojarzenie. Warte uwagi są dwie poniższe poprawki algorytmu H0. Stopień wierzchołka $x$ oznaczamy $deg(x)$.

H1. Wybieramy krawędź $(u, v)$, dla której $deg(u)$ jest najmniejsze.

H2. Wybieramy krawędź $(u, v)$, dla której wartość $(deg(u)-1) \cdot (deg(v)-1)$ jest najmniejsza.

Za pomocą kopców z operacją decreaseKey (kopiec binarny, kopiec Fibonacciego) można zbudować implementacje działającą w czasie $O(D(m) + m)$ [1], gdzie $D(x)$ to czas $x$ operacji decreaseKey na zbiorze $x$-elementowym. Dla kopca binarnego mamy $D(x) = O(x \log x)$, a dla kopca Fibonacciego $D(x) = O(x)$. Zatem, jeżeli użyjemy kopca Fibonacciego heurystyki H1 i H2 działają w czasie liniowym ze względu na rozmiar grafu.

Dla przykładu podajemy implementację H1.

[1] dla H1 zamiast krawędzi możemy wstawiać do kopca wierzchołki, zatem czas spada do $O(D(n) + m)$

```
procedure H1(G = (V, E))
    niech deg będzie tablicą stopni wierzchołków grafu G
    zbuduj kopiec Q
    for all x ∈ V do
        wstaw x do Q z kluczem deg[x]
    while not Q.empty() do
        x:=Q.getMin()
        if deg[x] < 1 then continue
        niech y wierzchołkiem, takim że (x, y) ∈ E i deg[y] ≠ 0
        skojarz x z y
        for all j, takiego że (x, j) ∈ E i deg[j] ≠ 0 do
            deg[j]− = 1
            Q.decreaseKey(j,deg[j])
        for all j, takiego że (y, j) ∈ E i deg[j] ≠ 0 do
            deg[j]-=1
            Q.decreaseKey(j,deg[j])
        deg[x]=0
        deg[y]=0
```

**Alg. 9:** Heurystyka H1

# Rozdział 4

# Omówienie testów i podsumowanie

## 4.1. Testy

Aby sprawdzić praktyczną efektywność algorytmu DMM i heurystyk przetwarzania wstępnego, przeprowadzono testy implementacji umieszczonej w dodatku ??. Program został skompilowany kompilatorem `gcc 3.2.2` z włączoną optymalizacją `-O2`. Testy przeprowadzono na komputerze z procesorem Duron 650.

Przetestowano kilka odmian algorytmów obliczania najliczniejszego skojarzenia:

**DMM** sam DMM bez dodatków

**DMM$^-$** wersja DMM, która przerywa obliczanie fazy po znalezieniu wszystkich najkrótszych rozłącznych ścieżek powiększających, a nie tak jak DMM, po opróżnieniu kolejki przeszukiwania i kolejki łączników.

**PRE** DMM poprzedzony znajdowaniem dowolnego maksymalnego skojarzenia.

**H0** DMM z heurystyką H0.

**H1** DMM z heurystyką H1.

Testy podzielona na pięć grup. W ramach grupy testy różnią się liczbą wierzchołków i liczbą krawędzi.

**P** Grafy losowe gęste – $m/n$ jest rzędu $\sqrt{n}$.

**Rzad0** Grafy losowe rzadkie – $m/n$ jest około 1-1.5.

**Rzad1** Grafy losowe rzadkie – $m/n$ jest około 1.5-4.

**Rzad2** Grafy losowe mniej rzadkie – $m/n$ jest około 3-21.

**Edg** Grafy losowe o 7000 wierzchołków sparametryzowane gęstością $m/n$

**W** Grafy nieregularne, ale nie losowe. Wierzchołki odpowiadają w nich najczęściej występującym pięcioliterowym słowom angielskim, a krawędzie parom słów różniących się dokładnie jedną literą. Testy wygenerowano przy pomocy procedur z Standford GraphBase [13] poleceniem `words(x,NULL,0,0)`, gdzie $x$ jest parametrem określającym liczbę słów.

**S** Grafy regularne. Wierzchołkami są trójki punktów $(x_0, x_1, x_2)$, gdzie $0 \leqslant x_0 \leqslant 10x$, $0 \leqslant x_1 \leqslant 20x$, $0 \leqslant x_2 \leqslant 10x$ i $x_0 + x_1 + x_2 \leqslant 20x$, a $x$ jest parametrem testu. Krawędzie łączą wierzchołki które różnią się na dokładnie dwu składowych o $\pm 1$. Testy uzyskano przy pomocy Standford GraphBase poleceniem `simplex(2x,x,2x,x,0,0)`.

Wyniki testów zamieszczamy poniżej. Czas wykonania podano w sekundach.

| n | 500 | 1000 | 2000 | 5000 | 7000 | 10000 | 15000 | 20000 | 30000 | 40000 |
|---|---|---|---|---|---|---|---|---|---|---|
| DMM | 0.07 | 0.19 | 0.62 | 3.10 | 5.30 | 10.06 | 19.18 | 30.55 | 57.88 | 93.04 |
| $DMM^-$ | 0.05 | 0.18 | 0.66 | 3.12 | 5.31 | 9.84 | 19.33 | 30.74 | 58.15 | 92.80 |
| PRE | 0.04 | 0.11 | 0.38 | 1.78 | 2.99 | 5.68 | 10.85 | 18.06 | 33.91 | 54.85 |
| H0 | 0.06 | 0.12 | 0.38 | 1.82 | 3.17 | 5.89 | 11.53 | 18.90 | 36.00 | 57.42 |
| H1 | 0.04 | 0.16 | 0.40 | 1.83 | 3.33 | 6.23 | 12.59 | 20.71 | 40.72 | 66.30 |

Tabela 4.1: Czasy wykonania dla grafów gęstych - P

| n | 500 | 1000 | 2000 | 5000 | 7000 | 10000 | 15000 | 20000 | 30000 | 40000 |
|---|---|---|---|---|---|---|---|---|---|---|
| DMM | 0.00 | 0.01 | 0.07 | 0.20 | 0.22 | 0.44 | 0.69 | 0.88 | 1.50 | 2.06 |
| $DMM^-$ | 0.02 | 0.03 | 0.06 | 0.31 | 0.41 | 0.78 | 1.27 | 1.70 | 3.36 | 4.56 |
| PRE | 0.01 | 0.02 | 0.06 | 0.16 | 0.19 | 0.36 | 0.57 | 0.68 | 1.21 | 1.65 |
| H0 | 0.01 | 0.02 | 0.02 | 0.09 | 0.09 | 0.12 | 0.29 | 0.29 | 0.47 | 0.68 |
| H1 | 0.02 | 0.02 | 0.02 | 0.10 | 0.15 | 0.21 | 0.31 | 0.42 | 0.79 | 0.89 |

Tabela 4.2: Czasy wykonania dla grafów rzadkich - Rzad0

| n | 500 | 1000 | 2000 | 5000 | 7000 | 10000 | 15000 | 20000 | 30000 | 40000 |
|---|---|---|---|---|---|---|---|---|---|---|
| DMM | 0.02 | 0.04 | 0.09 | 0.24 | 0.33 | 0.44 | 1.09 | 0.86 | 1.45 | 2.72 |
| $DMM^-$ | 0.03 | 0.05 | 0.07 | 0.27 | 0.40 | 0.50 | 0.97 | 1.11 | 1.75 | 2.43 |
| PRE | 0.02 | 0.02 | 0.05 | 0.19 | 0.28 | 0.33 | 0.85 | 0.62 | 1.04 | 2.07 |
| H0 | 0.01 | 0.03 | 0.08 | 0.22 | 0.31 | 0.22 | 0.51 | 0.44 | 0.67 | 0.96 |
| H1 | 0.03 | 0.04 | 0.07 | 0.23 | 0.31 | 0.27 | 0.79 | 0.59 | 0.90 | 1.36 |

Tabela 4.3: Czasy wykonania dla grafów rzadkich - Rzad1

| n | 500 | 1000 | 2000 | 5000 | 7000 | 10000 | 15000 | 20000 | 30000 | 40000 |
|---|---|---|---|---|---|---|---|---|---|---|
| DMM | 0.00 | 0.06 | 0.08 | 0.40 | 0.36 | 0.58 | 0.94 | 1.97 | 2.98 | 4.14 |
| $DMM^-$ | 0.02 | 0.04 | 0.11 | 0.34 | 0.40 | 0.64 | 0.99 | 1.84 | 2.83 | 3.93 |
| PRE | 0.02 | 0.05 | 0.05 | 0.33 | 0.26 | 0.39 | 0.63 | 1.55 | 2.39 | 3.14 |
| H0 | 0.01 | 0.05 | 0.05 | 0.31 | 0.21 | 0.32 | 0.50 | 1.43 | 2.40 | 3.18 |
| H1 | 0.02 | 0.03 | 0.05 | 0.23 | 0.21 | 0.36 | 0.56 | 1.23 | 1.91 | 2.77 |

Tabela 4.4: Czasy wykonania dla grafów z grupy Rzad2

## 4.2. Wnioski

Z testów wynika, że najwięcej faz algorytm DMM wykonuje przy grafach rzadkich (dla $m/n$ rzędu $1-5$). Jednakże dzięki temu, że DMM nie kończy fazy po znalezieniu zbioru rozłącznych wierzchołkowo, najkrótszych ścieżek powiększających, liczba faz może się zmniejszyć nawet

| m | DMM | DMM$^-$ | PRE | H0 | H1 |
|---|---|---|---|---|---|
| 7000 | 0.16 | 0.20 | 0.11 | 0.06 | 0.10 |
| 14000 | 0.26 | 0.33 | 0.21 | 0.11 | 0.13 |
| 21000 | 0.31 | 0.35 | 0.22 | 0.15 | 0.17 |
| 35000 | 0.42 | 0.45 | 0.28 | 0.24 | 0.23 |
| 49000 | 0.52 | 0.55 | 0.34 | 0.26 | 0.33 |
| 70000 | 0.74 | 0.79 | 0.44 | 0.41 | 0.39 |
| 105000 | 1.04 | 1.15 | 0.65 | 0.70 | 0.65 |
| 140000 | 1.32 | 1.38 | 0.80 | 0.85 | 0.91 |
| 210000 | 1.97 | 2.21 | 1.19 | 1.37 | 1.19 |
| 350000 | 3.28 | 3.30 | 1.89 | 1.91 | 1.95 |
| 840000 | 8.08 | 8.17 | 4.85 | 4.62 | 4.82 |
| 1610000 | 15.69 | 15.78 | 8.88 | 9.73 | 9.82 |
| 3500000 | 33.25 | 33.24 | 19.70 | 20.79 | 21.66 |
| 7000000 | 71.00 | 70.74 | 44.07 | 41.66 | 44.20 |

Tabela 4.5: Czasy wykonania dla grafów z grupy Edg

| m | DMM | DMM$^-$ | PRE | H0 | H1 |
|---|---|---|---|---|---|
| 7000 | 5 | 10 | 4 | 1 | 1 |
| 14000 | 4 | 15 | 5 | 2 | 2 |
| 21000 | 4 | 10 | 3 | 2 | 2 |
| 35000 | 3 | 6 | 3 | 2 | 2 |
| 49000 | 3 | 5 | 2 | 2 | 2 |
| 70000 | 3 | 5 | 2 | 2 | 1 |
| 105000 | 3 | 4 | 2 | 2 | 1 |
| 140000 | 3 | 4 | 2 | 2 | 2 |
| 210000 | 3 | 4 | 2 | 2 | 1 |
| 350000 | 3 | 3 | 2 | 2 | 1 |
| 840000 | 3 | 3 | 2 | 1 | 1 |
| 1610000 | 3 | 3 | 2 | 2 | 1 |
| 3500000 | 3 | 3 | 2 | 1 | 1 |
| 7000000 | 3 | 3 | 2 | 1 | 1 |

Tabela 4.6: Liczba faz dla grafów z grupy Edg

| n | 1000 | 2000 | 3000 | 4000 | 5000 | 5500 |
|---|---|---|---|---|---|---|
| DMM | 0.02 | 0.05 | 0.14 | 0.19 | 0.23 | 0.29 |
| DMM$^-$ | 0.03 | 0.07 | 0.18 | 0.25 | 0.32 | 0.36 |
| PRE | 0.01 | 0.05 | 0.12 | 0.16 | 0.19 | 0.20 |
| H0 | 0.02 | 0.06 | 0.11 | 0.12 | 0.18 | 0.21 |
| H1 | 0.02 | 0.04 | 0.09 | 0.14 | 0.16 | 0.19 |

Tabela 4.7: Czasy wykonania dla grafów z grupy W

trzykrotnie (Tab. 4.6), tyle tylko, że taki efekt występuje dla grafów rzadkich, dla których i tak algorytm działa bardzo szybko. Natomiast dla grafów z dużą liczbą krawędzi algorytm

| n | 1681 | 2601 | 5041 | 10201 | 40401 |
|---|---|---|---|---|---|
| m | 4880 | 7600 | 14840 | 30200 | 120400 |
| DMM | 0.05 | 0.08 | 0.17 | 0.33 | 1.53 |
| DMM$^-$ | 0.06 | 0.06 | 0.13 | 0.28 | 1.14 |
| PRE | 0.05 | 0.06 | 0.13 | 0.33 | 1.25 |
| H0 | 0.04 | 0.08 | 0.13 | 0.27 | 1.26 |
| H1 | 0.03 | 0.05 | 0.09 | 0.25 | 1.46 |

Tabela 4.8: Czasy wykonania dla grafów z grupy S

nieco zwalnia (dla około 7 mln krawędzi). Nie można tego wytłumaczyć zwiększoną liczbą faz, bo tych dla grafów gęstych jest mało, przeważnie 3. Okazuje się, że problem leży w alokowaniu dużej ilości małych kawałków pamięci. Autor zauważył, że dla największych testów program zajmował kilkaset megabajtów pamięci do obsługi kolejki łączników.

Heurystyki przetwarzania wstępnego okazują się niezwykle pomocne nawet dla tak szybkiego algorytmu jakim jest DMM. Po pierwsze zmniejszają liczbę faz algorytmu DMM. Po drugie znajdują maksymalny zbiór rozłącznych ścieżek powiększających o długości 1 o wiele szybciej niż DMM. H0 potrzebuje na to czasu $m+n$, PRE $n$ , a sam DMM około $6m$. Dlatego też PRE dla grafów gęstych mimo, że nie jest tak efektywny jak H0 czy H1, okazuje się minimalnie szybszy. Siłę wszystkich heurystyk można zauważyć na wszystkich testach. Średnio zmniejszają one czas wykonania o połowę.

Algorytm DMM połączony z heurystyką, ma nie tylko dobre asymptotyczne oszacowanie czasu, ale także staje się szybkim sposobem na szukanie najliczniejszego skojarzenia w praktyce.

## 4.3. Podsumowanie

Algorytm DMM jest poważną alternatywą dla popularnego algorytmu podanego przez Micaliego i Vaziraniego. Oba podejścia są do siebie podobne, bo wykorzystują te same idee: zwijanie kielichów, $O(\sqrt{n})$ faz. Jednakże rozróżnianie przez DMM ścieżek aktualnych i alternatywnych sprawia, że część Connect jest prostsza od odpowiadającej jej części MAX z algorytmu Micali-Vazirani. Dodatkowo, wydaje się, że dowód poprawności DMM jest mniej skomplikowany. Sam algorytm jest dopracowany w najdrobniejszych szczegółach, a implementacja DMM zajmuje niewiele, bo tylko 400 wierszy kodu. Dalsze badania nad DMM powinny się koncentrować na optymalizacji zużycia pamięci i efektywnych metodach przetwarzania wstępnego.

# Dodatek A

# Zawartość płyty CD

Do pracy załączono płytę CD na której znajdują się:

- program implementujący algorytm DMM wraz z heurystykami z rozdziału 3.
- testy; dane wejściowe, wyniki
- wyniki przeprowadzonych testów
- skrypty porównujące wyniki i sprawdzające wydajność programu
- kopia pracy magisterskiej

# Spis literatury


[1] C. Berge, *Two theorems in graph theory*, Proc. Natl, Acad. Sci 43 (1957), 842-844.

[2] N. Blum, *Jack Edmonds orginal algorithm needs only $O(n^3)$ time*, Research Report No. 8568-CS, University of Bonn (1991).

[3] N. Blum, *A simplified realization of Hopcroft-Karp approach to maximum matching in nonbipartite graphs*, Research Report, University of Bonn (1999).

[4] N. Blum, *Maximum matching in nonbipartite graphs without explicit consideration of blossoms*, Research Report, University of Bonn (1999).

[5] J. Edmonds, *Paths, trees, and flowers*, Canad. J. Math, 17 (1965), 449-467.

[6] S. Even, O. Kariv, *An $O(n^{2.5})$ algorithm for maximum matching in general graphs*, Proc. 16th Annual IEEE Symposium on Foundations of Computer Science (1975), 100-112.

[7] Zvi Galil, *Efficient Algorithms for Finding Maximum Mathing in Graphs*, Computing Surveys, Vol. 18, No. 1, March 1986.

[8] H. Gabow, *An efficient implementation of Edmonds' algorithm for maximum matching on graphs*, J. ACM 23 (1976), 221-231.

[9] H. N. Gabow, R. E. Tarjan, *A linear-time algorithm for special case of disjoint set union*, J. Comput. Syst. Sci., bf 30 (1985), 209-221.

[10] H. N. Gabow, R. E. Tarjan *Faster scaling algorithm for general graph-matching problems*, J. ACM 38 (1991) 815-853.

[11] J. Hopcroft, R. M. Karp, *A $O(n^{5/2})$ algorithm for maximum matching in bipartite graphs*, SIAM J. Comput. 2(1973).

[12] T. Kameda, I. Munro, *A $O(|V||E|)$ algorithm for maximum matching of graphs*, Computing 12 (1974), 91-98.

[13] D. E. Knuth, *The Stanford GraphBase: a platform for combinatorial computing, ACM Press*, New York(1994).

[14] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Hold, Rinehart and Winston, New York 1976.

[15] S. Micali, V. V. Vazarini, *An $O(\sqrt{|V|}|E|)$ algorithm for finding maximal matching in general graphs* , Proceedings of the 21st Annual IEEE Symposium on Fundations of Computer Science. IEEE, New York, pp 17-27.

[16] Marcin Mucha, Piotr Sankowski, *Maximum matchings via gaussian elimination*, preprint.

[17] Vijay V. Vazirani, *A theory of alternating paths and bloossoms for proving correctness of the $O(\sqrt{|V|}|E|)$ general graph maximum matching algorithm*, Combinatorica 14(1) (1994) 71-109.

[18] Maciej M. Sysło, Narsingh Deo, Janusz S. Kowalik, *Algorytmy optymalizacji dyskretnej*, Wydawnictwo Naukowe PWN Warszawa 1995.

[19] C. Witzgall, C. T. Jr. Zahn, *Modification of Edmonds maximum matching algorithm*, J. Res. Nat. Bur. standards, 69B (1965), 91-98.