Drzewa rozpinające, zbiory rozłączne, czas zamortyzowany

zajęcia 3.

Wojciech Śmietanka, Tomasz Kulczyński, Błażej Osiński

Drzewa rozpinające

Drzewa rozpinające

Mamy graf nieskierowany, ważony, wagi większe od 0.
Chcemy wybrać taki podzbiór krawędzi, żeby :

- była droga pomiędzy każdą parą wierzchołków
- suma wag wybranych krawędzi była minimalna

Drzewa rozpinające

Drzewa rozpinające

Mamy graf nieskierowany, ważony, wagi większe od 0.
Chcemy wybrać taki podzbiór krawędzi, żeby :

- była droga pomiędzy każdą parą wierzchołków
- suma wag wybranych krawędzi była minimalna

Drzewa rozpinające

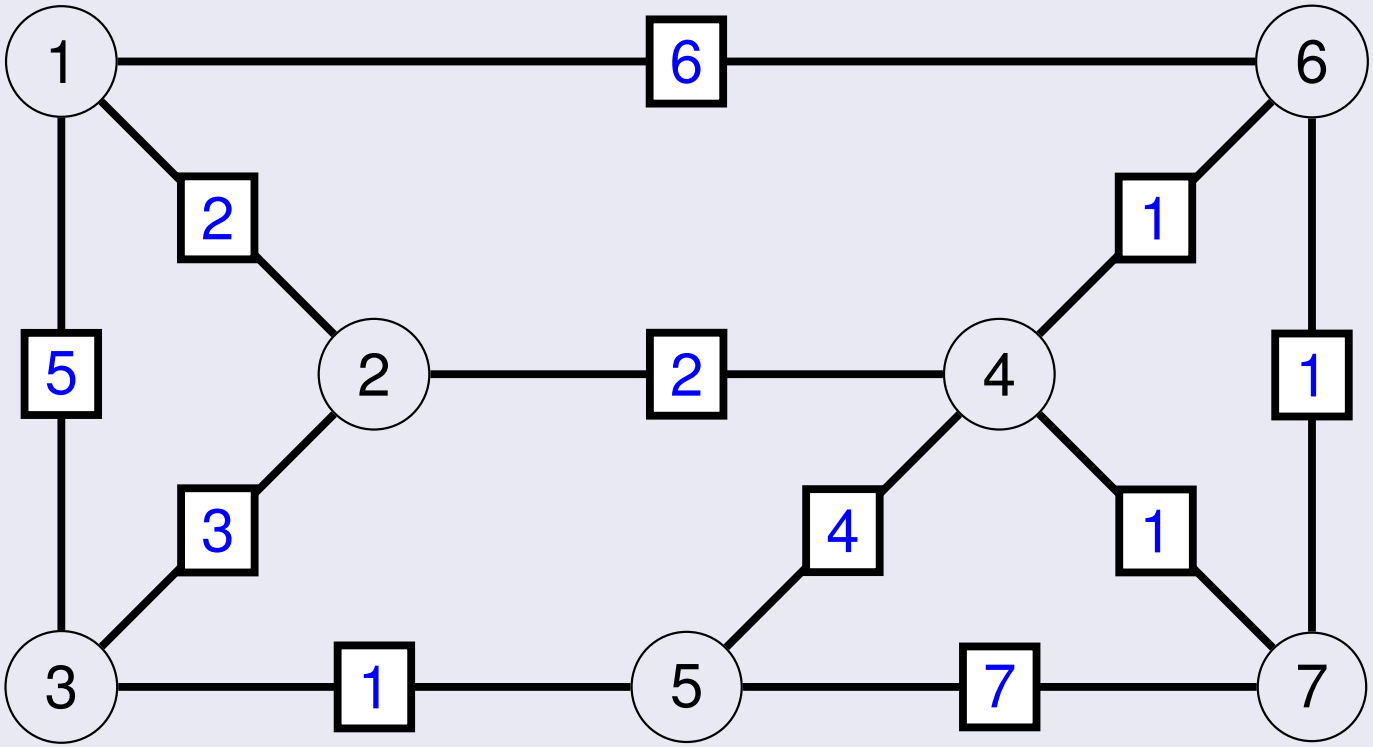
Drzewa rozpinające

Mamy graf nieskierowany, ważony, wagi większe od 0.
Chcemy wybrać taki podzbiór krawędzi, żeby :

- była droga pomiędzy każdą parą wierzchołków
- suma wag wybranych krawędzi była minimalna

Przykład

Popatrzmy na poniższą sieć połączeń i spróbujmy znaleźć drzewo rozpinające tej sieci.



Rozwiązanie

W tym celu wypiszmy sobie wszystkie krawędzie posortowane po rosnącej wadze:

koniec 1	koniec 2	waga
6	7	1
3	5	1
6	4	1
4	7	1
1	2	2
2	4	2
2	3	3
4	5	4
1	3	5
1	6	6
5	7	7

I co dalej?

Patrzymy na kolejne krawędzie z tej listy

- próbujemy dodawać je do budowanego drzewa rozpinającego
- jeśli dana krawędź łączy dwa wierzchołki, między którymi nie było drogi, to dodajemy ją do drzewa
- jeśli nie, to odrzucamy
- kontynuujemy, aż wszystkie wierzchołki będą połączone

I co dalej?

Patrzymy na kolejne krawędzie z tej listy

- próbujemy dodawać je do budowanego drzewa rozpinającego
- jeśli dana krawędź łączy dwa wierzchołki, między którymi nie było drogi, to dodajemy ją do drzewa
- jeśli nie, to odrzucamy
- kontynuujemy, aż wszystkie wierzchołki będą połączone

I co dalej?

Patrzymy na kolejne krawędzie z tej listy

- próbujemy dodawać je do budowanego drzewa rozpinającego
- jeśli dana krawędź łączy dwa wierzchołki, między którymi nie było drogi, to dodajemy ją do drzewa
- jeśli nie, to odrzucamy
- kontynuujemy, aż wszystkie wierzchołki będą połączone

I co dalej?

Patrzymy na kolejne krawędzie z tej listy

- próbujemy dodawać je do budowanego drzewa rozpinającego
- jeśli dana krawędź łączy dwa wierzchołki, między którymi nie było drogi, to dodajemy ją do drzewa
- jeśli nie, to odrzucamy
- kontynuujemy, aż wszystkie wierzchołki będą połączone

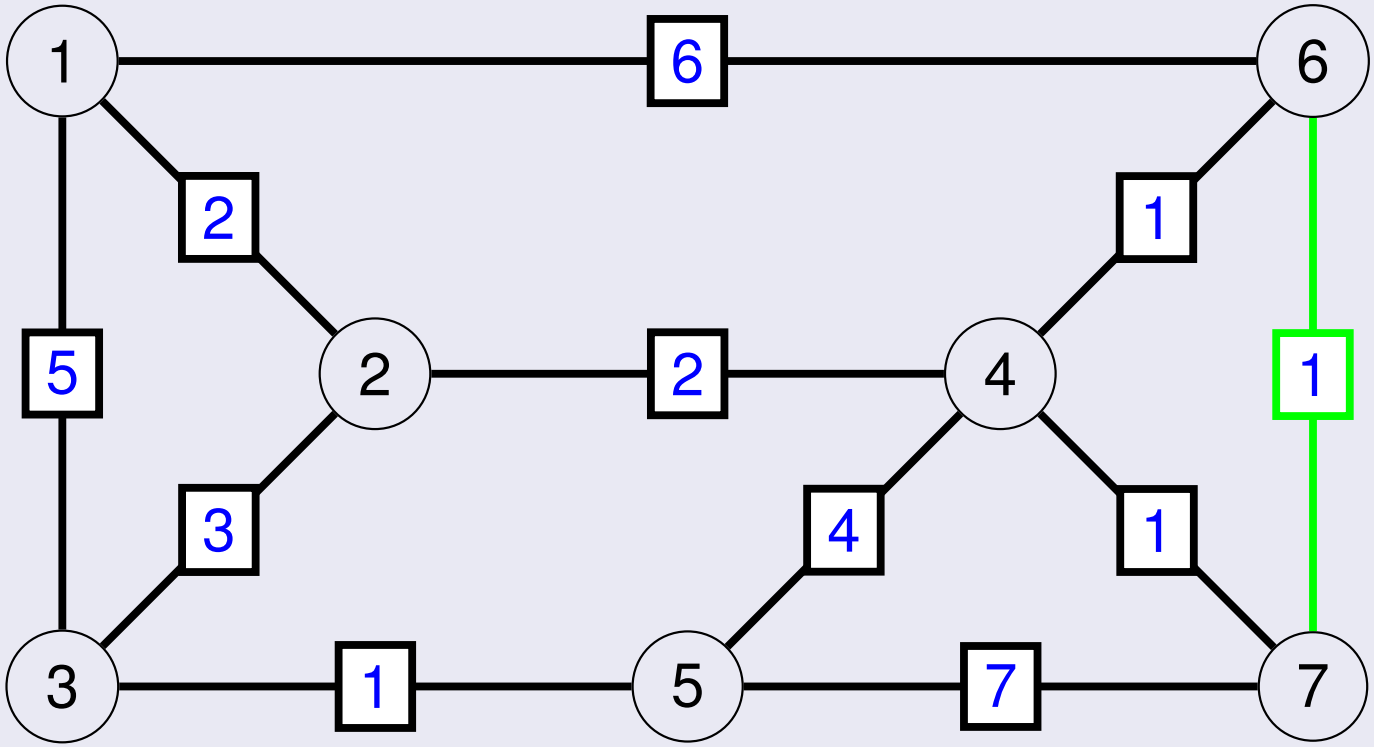
I co dalej?

Patrzymy na kolejne krawędzie z tej listy

- próbujemy dodawać je do budowanego drzewa rozpinającego
- jeśli dana krawędź łączy dwa wierzchołki, między którymi nie było drogi, to dodajemy ją do drzewa
- jeśli nie, to odrzucamy
- kontynuujemy, aż wszystkie wierzchołki będą połączone

Przykład

Sieć:

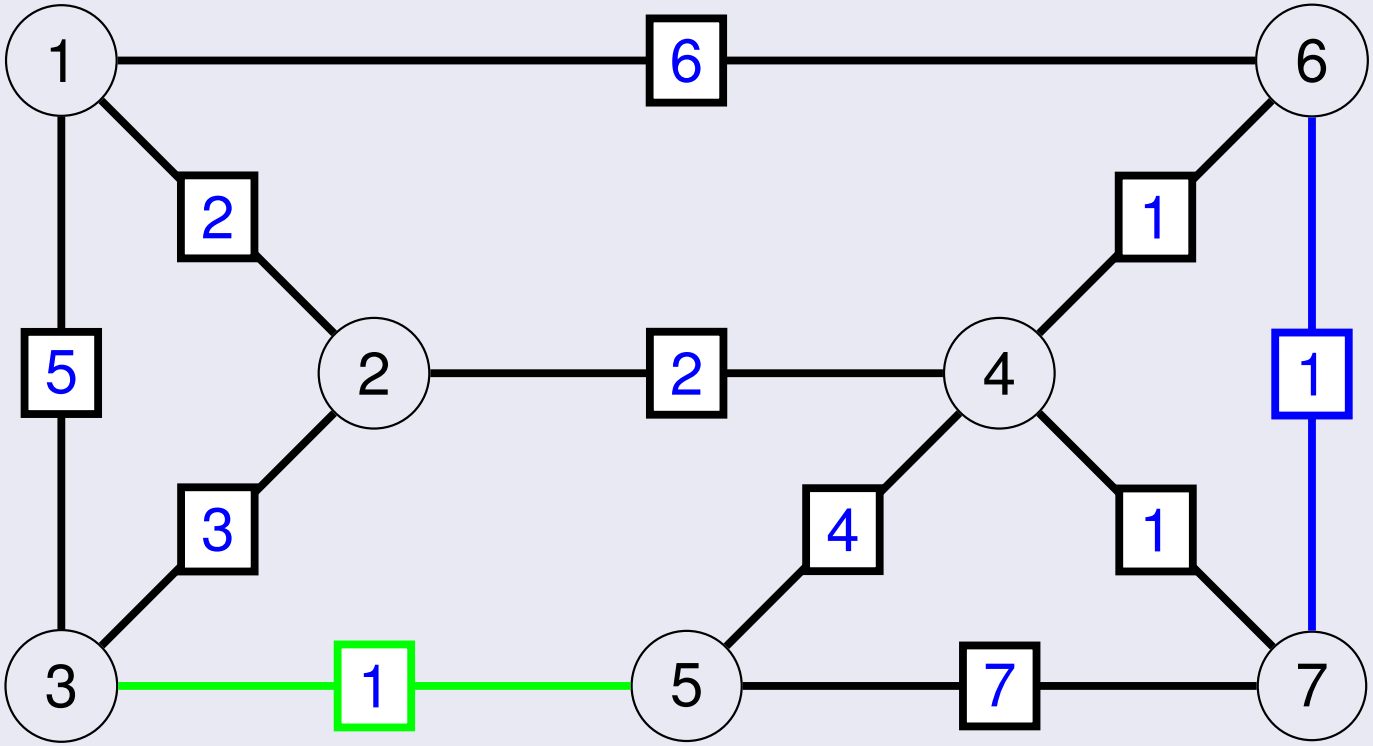


aktualna krawędź: 6 7 1

przychodzi krawędź 6 7 1, nie da się jeszcze przejechać między 6 a 7, dodajemy tę krawędź

Przykład

Sieć:

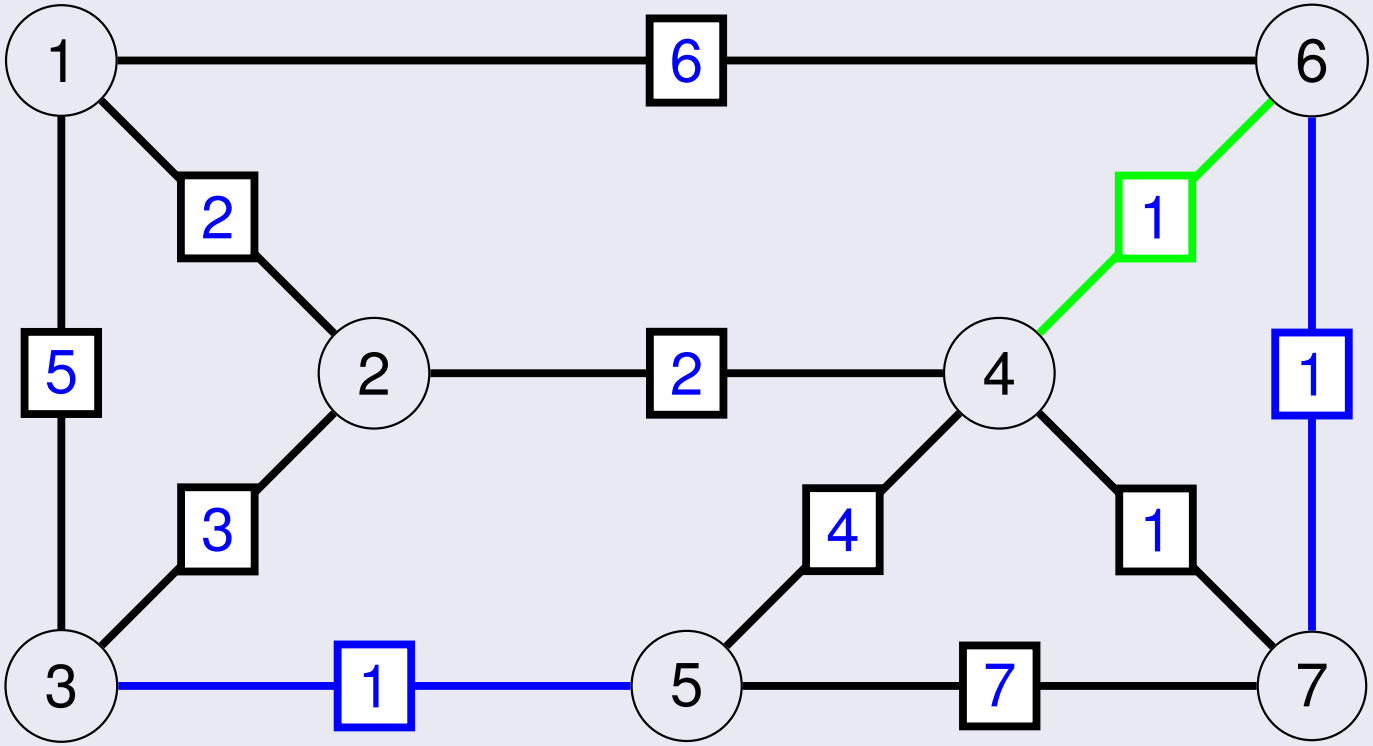


aktualna krawędź: 3 5 1

przychodzi krawędź 3 5 1, ją też dodajemy do budowanego drzewa

Przykład

Sieć:

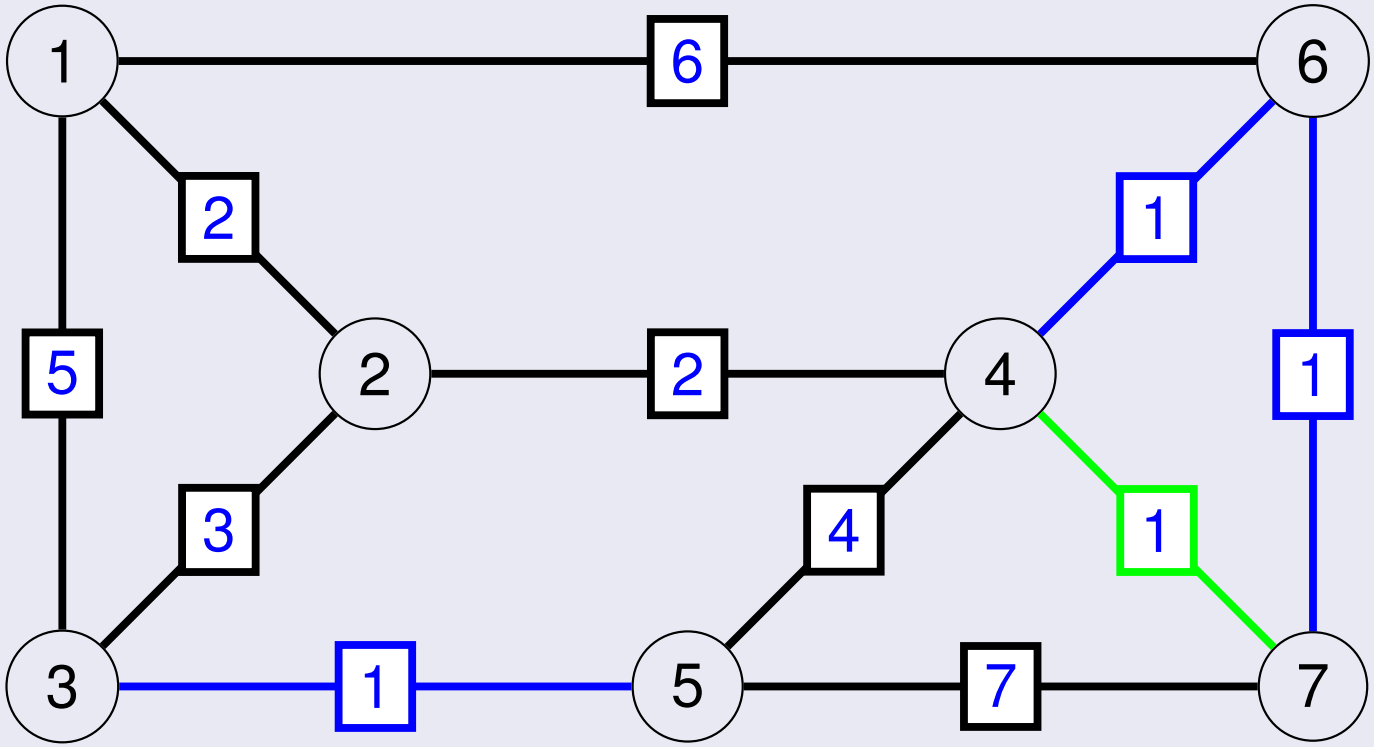


aktualna krawędź: 6 4 1

przychodzi krawędź 6 4 1, ją też dodajemy do budowanego drzewa

Przykład

Sieć:

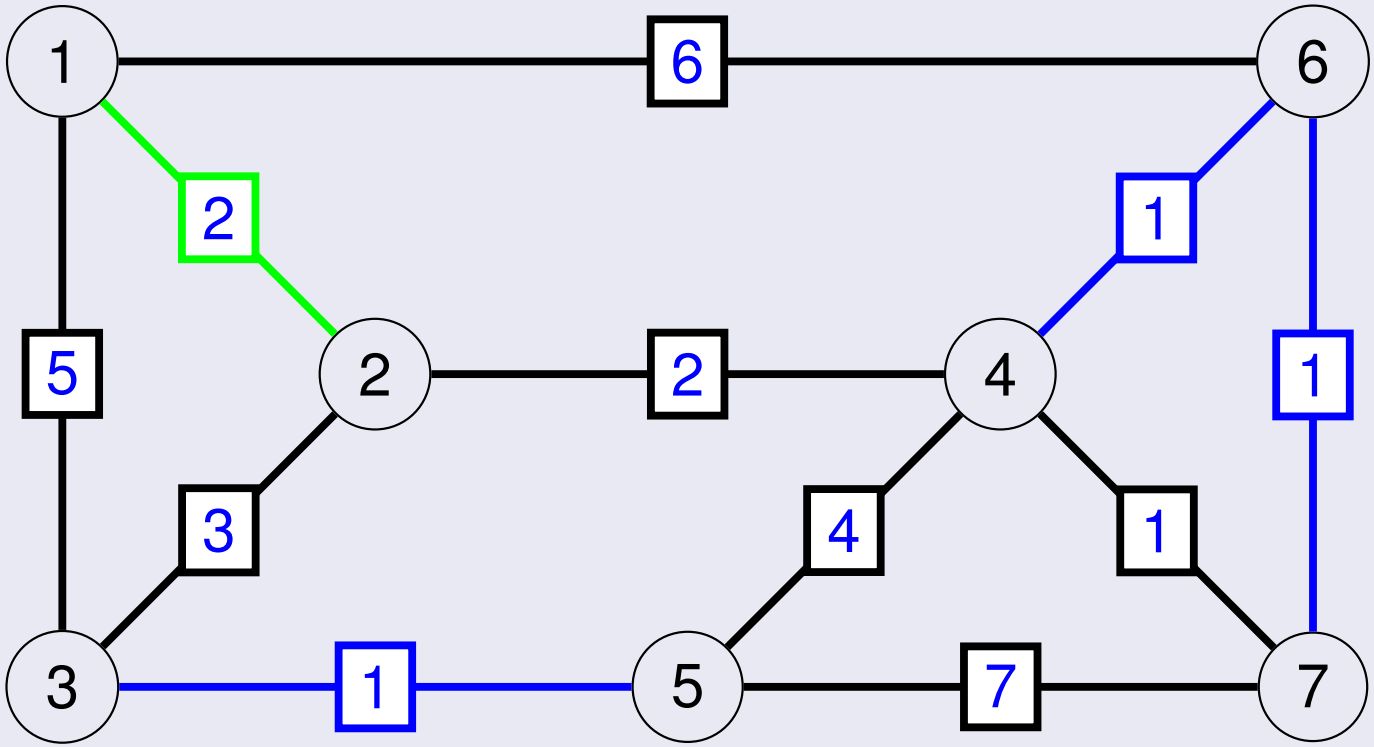


aktualna krawędź: 7 4 1

przychodzi krawędź 7 4 1, ale nie jest potrzebna, jest już droga między 7 i 4

Przykład

Sieć:

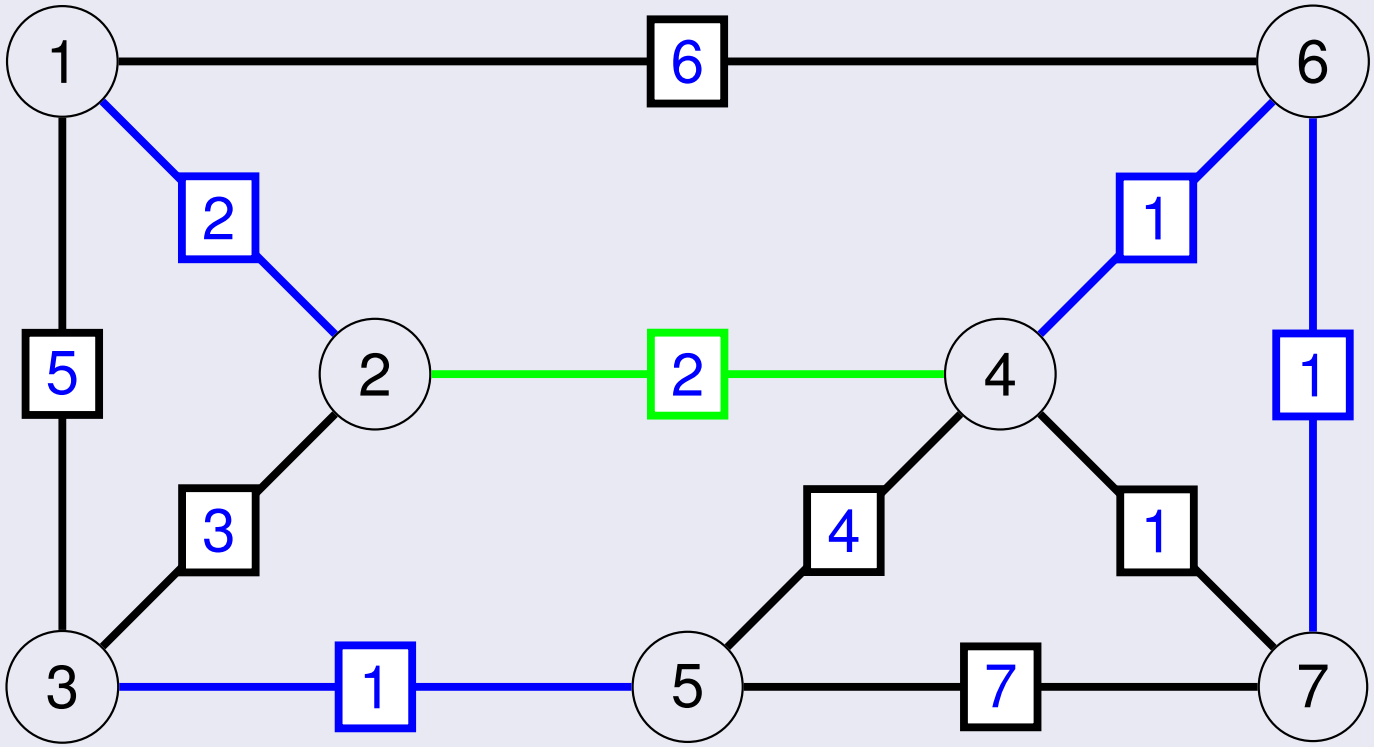


aktualna krawędź: 1 2 2

krawędź 1 2 2 łączy nie połączone jeszcze wierchołki 1, 2

Przykład

Sieć:

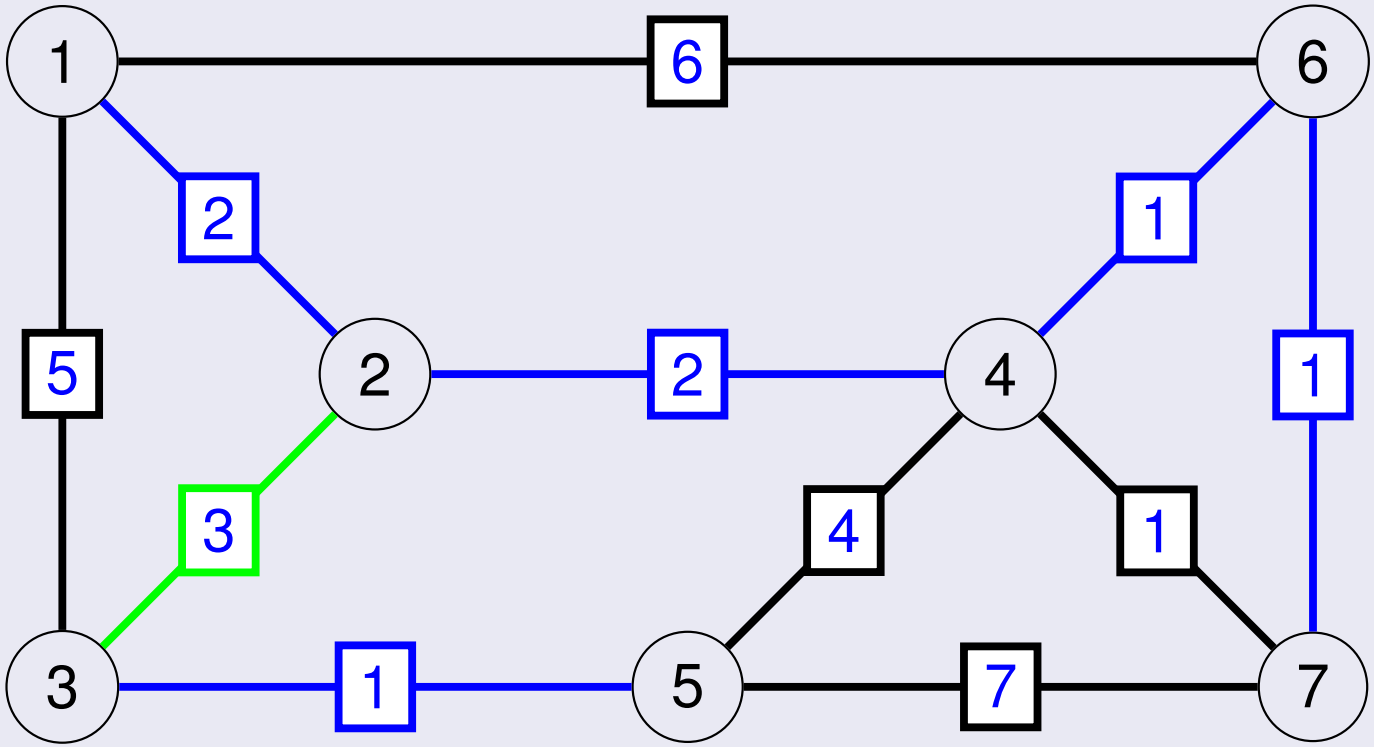


aktualna krawędź: 2 4 2

krawędź 2 4 2 łączy 2 i 4, nie były one jeszcze połączone

Przykład

Sieć:

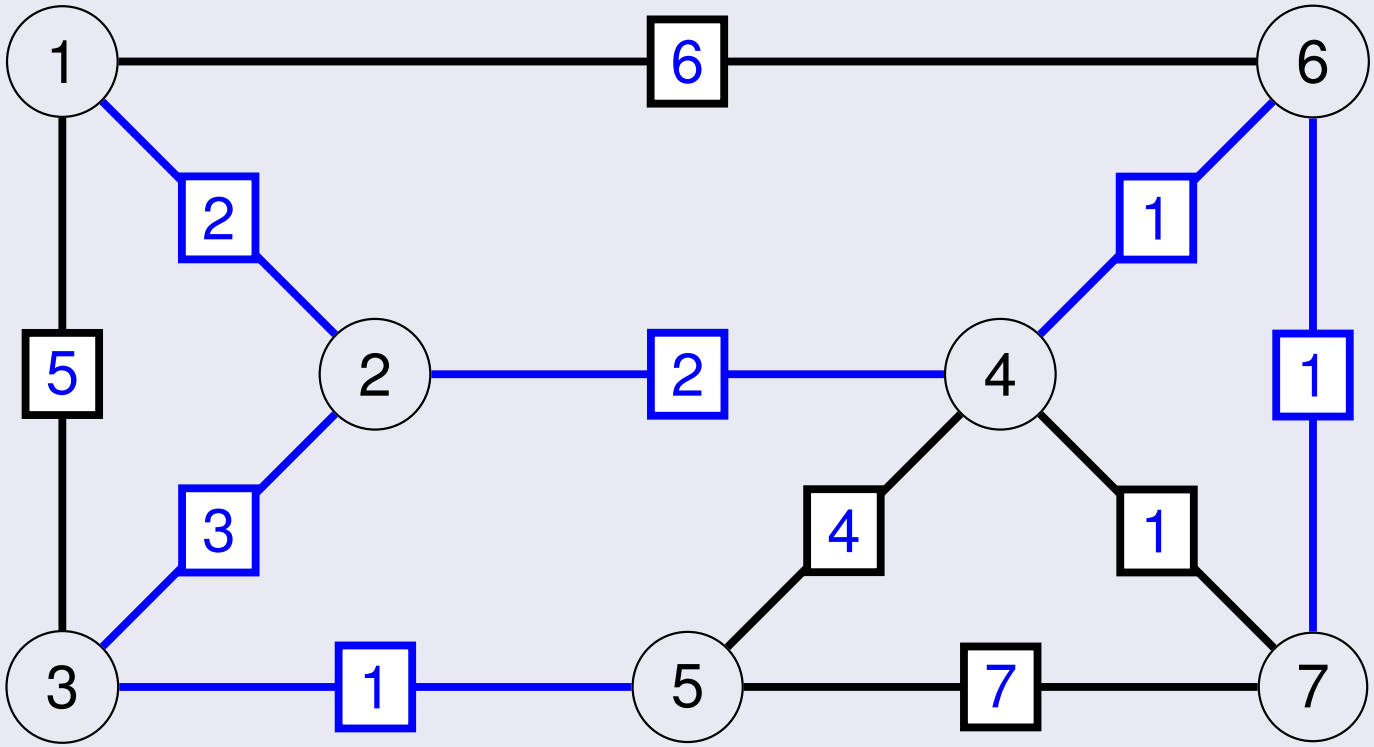


aktualna krawędź: 2 3 3

krawędź 2 3 3 łączy niepołączone dotąd 2 i 3

Przykład

Sieć:



Hurra! Cały graf jest już połączony drzewem rozpinającym!

A skąd wiadomo, że..?

A skąd wiadomo, że wierzchołki są w jednej spójnej składowej?

- do tej pory pomijaliśmy to milczeniem
- wystarczyło spojrzeć na rysunek
- niestety ciężko wytłumaczyć komputerowi, żeby spojrzał na rysunek

A skąd wiadomo, że..?

A skąd wiadomo, że wierzchołki są w jednej spójnej składowej?

- do tej pory pomijaliśmy to milczeniem
- wystarczyło spojrzeć na rysunek
- niestety ciężko wytłumaczyć komputerowi, żeby spojrzał na rysunek

A skąd wiadomo, że..?

A skąd wiadomo, że wierzchołki są w jednej spójnej składowej?

- do tej pory pomijaliśmy to milczeniem
- wystarczyło spojrzeć na rysunek
- niestety ciężko wytłumaczyć komputerowi, żeby spojrzał na rysunek

A skąd wiadomo, że..?

A skąd wiadomo, że wierzchołki są w jednej spójnej składowej?

- do tej pory pomijaliśmy to milczeniem
- wystarczyło spojrzeć na rysunek
- niestety ciężko wytłumaczyć komputerowi, żeby spojrzał na rysunek

Co potrzeba?

Co potrzeba?

Potrzebna jest struktura danych, która pozwala szybko sprawdzić, czy dane dwa wierzchołki leżą w jednej składowej oraz potrafiąca szybko łączyć dwa wierzchołki. Innymi słowy potrzebujemy reprezentantentować rodzinę rozłącznych zbiorów wierzchołków. Czasem będziemy łączyć dwa takie zbiory w jeden (dodanie krawędzi). Czasem zaś będziemy zadawać pytanie, czy dane dwa wierzchołki należą do tego samego zbioru.

Specyfikacja

Specyfikacja

Mamy początkowo n zbiorów (np. po jednym zbiorze na wierzchołek), m zapytań postaci „Czy elementy x i y należą do tego samego zbioru (zapytanie odpowiada krawędzi) oraz co najwyżej $n-1$ operacji scalenia zbiorów.
Wyróżniamy dwie operacje:

- `FIND(x)` - zwraca **reprezentantententanta** zbioru, do którego należy x, przy czym dla dowolnych różnych elementów tego samego zbioru zwraca te same elementy
- `UNION(x,y)` - jeśli x i y należą do innych zbiorów, to łączy je ze sobą

Specyfikacja

Specyfikacja

Mamy początkowo n zbiorów (np. po jednym zbiorze na wierzchołek), m zapytań postaci „Czy elementy x i y należą do tego samego zbioru (zapytanie odpowiada krawędzi) oraz co najwyżej $n - 1$ operacji scalenia zbiorów.
Wyróżniamy dwie operacje:

- `FIND(x)` - zwraca **reprezentantententanta** zbioru, do którego należy x, przy czym dla dowolnych różnych elementów tego samego zbioru zwraca te same elementy
- `UNION(x,y)` - jeśli x i y należą do innych zbiorów, to łączy je ze sobą

Specyfikacja

Specyfikacja

Mamy początkowo n zbiorów (np. po jednym zbiorze na wierzchołek), m zapytań postaci „Czy elementy x i y należą do tego samego zbioru (zapytanie odpowiada krawędzi) oraz co najwyżej $n-1$ operacji scalenia zbiorów.
Wyróżniamy dwie operacje:

- `FIND(x)` - zwraca **reprezentantententanta** zbioru, do którego należy x, przy czym dla dowolnych różnych elementów tego samego zbioru zwraca te same elementy
- `UNION(x,y)` - jeśli x i y należą do innych zbiorów, to łączy je ze sobą

Rozwiązanie 1

Rozwiązanie 1

Mamy tablicę `reprezentant[i]`, której *i*-ta pozycja zawiera informację o reprezentantentancie zbioru, do którego należy *i*. Początkowo dla każdego *i* `reprezentant[i] == i`. Jeśli robimy `UNION(i,j)`, to podmieniamy wszystkie wystąpienia wartości `reprezentant[j]` poprzez `reprezentant[i]`. Operacja `FIND(i)` zwraca `reprezentant[i]`.

Cechy rozwiązania 1

Cechy

- bardzo szybkie sprawdzanie, czy dane dwa elementy są w jednym zbiorze $O(1)$
- powolne łączenie dwóch zbiorów $O(n)$
- prosta implementacja
- strasznie duży czas całkowity $O(n^2 + m)$ (wszystkie operacje `FIND` i `UNION`)

Cechy rozwiązania 1

Cechy

- bardzo szybkie sprawdzanie, czy dane dwa elementy są w jednym zbiorze $O(1)$
- powolne łączenie dwóch zbiorów $O(n)$
- prosta implementacja
- strasznie duży czas całkowity $O(n^2 + m)$ (wszystkie operacje `FIND` i `UNION`)

Cechy rozwiązania 1

Cechy

- bardzo szybkie sprawdzanie, czy dane dwa elementy są w jednym zbiorze $O(1)$
- powolne łączenie dwóch zbiorów $O(n)$
- prosta implementacja
- strasznie duży czas całkowity $O(n^2 + m)$ (wszystkie operacje `FIND` i `UNION`)

Cechy rozwiązania 1

Cechy

- bardzo szybkie sprawdzanie, czy dane dwa elementy są w jednym zbiorze $O(1)$
- powolne łączenie dwóch zbiorów $O(n)$
- prosta implementacja
- strasznie duży czas całkowity $O(n^2 + m)$ (wszystkie operacje `FIND` i `UNION`)

Rozwiązanie 2

A jak można szybciej łączyć?

- trzymamy zbiory jako listy elementów
- dodatkowo każdy element ma wskaźnik na pierwszy element listy, czyli na reprezentantententanta
- zawsze pamiętamy rozmiar takiej listy
- łączenie takich dwóch list odbywa się na zasadzie „doklej mały do dużego”, czyli do większego zbioru doklejamy na koniec mniejszy, i w tym mniejszym zbiorze modyfikujemy wskaźniki na reprezentantententanta

Rozwiązanie 2

A jak można szybciej łączyć?

- trzymamy zbiory jako listy elementów
- dodatkowo każdy element ma wskaźnik na pierwszy element listy, czyli na reprezentantententanta
- zawsze pamiętamy rozmiar takiej listy
- łączenie takich dwóch list odbywa się na zasadzie „doklej mały do dużego”, czyli do większego zbioru doklejamy na koniec mniejszy, i w tym mniejszym zbiorze modyfikujemy wskaźniki na reprezentantententanta

Rozwiązanie 2

A jak można szybciej łączyć?

- trzymamy zbiory jako listy elementów
- dodatkowo każdy element ma wskaźnik na pierwszy element listy, czyli na reprezentantententanta
- zawsze pamiętamy rozmiar takiej listy
- łączenie takich dwóch list odbywa się na zasadzie „doklej mały do dużego”, czyli do większego zbioru doklejamy na koniec mniejszy, i w tym mniejszym zbiorze modyfikujemy wskaźniki na reprezentantententanta

Rozwiązanie 2

A jak można szybciej łączyć?

- trzymamy zbiory jako listy elementów
- dodatkowo każdy element ma wskaźnik na pierwszy element listy, czyli na reprezentantententanta
- zawsze pamiętamy rozmiar takiej listy
- łączenie takich dwóch list odbywa się na zasadzie „doklej mały do dużego”, czyli do większego zbioru doklejamy na koniec mniejszy, i w tym mniejszym zbiorze modyfikujemy wskaźniki na reprezentantententanta

Cechy rozwiązania 2

Cechy rozwiązania 2

- szybki czas sprawdzania, czy dwa elementy należą do jednego zbioru $O(1)$
- łączenie jest szybsze, bo jeśli jakiemuś elementowi podmieniamy wskaźnik na reprezentantententanta, to nagle ten element znajduje się w zbiorze co najmniej dwa razy większym
- wniosek? każdy element ma podmieniany wskaźnik co najwyżej $O(\log n)$ razy
- cały czas łączenia wynosi więc $O(n \log n)$, więc **zamortyzowany czas** przeznaczony na jedną operację `UNION` wynosi $O(\log n)$
- łączny czas działania $O(n \log n + m)$

Cechy rozwiązania 2

Cechy rozwiązania 2

- szybki czas sprawdzania, czy dwa elementy należą do jednego zbioru $O(1)$
- łączenie jest szybsze, bo jeśli jakiemuś elementowi podmieniamy wskaźnik na reprezentantententanta, to nagle ten element znajduje się w zbiorze co najmniej dwa razy większym
- wniosek? każdy element ma podmieniany wskaźnik co najwyżej $O(\log n)$ razy
- cały czas łączenia wynosi więc $O(n \log n)$, więc **zamortyzowany czas** przeznaczony na jedną operację `UNION` wynosi $O(\log n)$
- łączny czas działania $O(n \log n + m)$

Cechy rozwiązania 2

Cechy rozwiązania 2

- szybki czas sprawdzania, czy dwa elementy należą do jednego zbioru $O(1)$
- łączenie jest szybsze, bo jeśli jakiemuś elementowi podmieniamy wskaźnik na reprezentantententanta, to nagle ten element znajduje się w zbiorze co najmniej dwa razy większym
- wniosek? każdy element ma podmieniany wskaźnik co najwyżej $O(\log n)$ razy
- cały czas łączenia wynosi więc $O(n \log n)$, więc **zamortyzowany czas** przeznaczony na jedną operację `UNION` wynosi $O(\log n)$
- łączny czas działania $O(n \log n + m)$

Cechy rozwiązania 2

Cechy rozwiązania 2

- szybki czas sprawdzania, czy dwa elementy należą do jednego zbioru $O(1)$
- łączenie jest szybsze, bo jeśli jakiemuś elementowi podmieniamy wskaźnik na reprezentantententanta, to nagle ten element znajduje się w zbiorze co najmniej dwa razy większym
- wniosek? każdy element ma podmieniany wskaźnik co najwyżej $O(\log n)$ razy
- cały czas łączenia wynosi więc $O(n \log n)$, więc **zamortyzowany czas** przeznaczony na jedną operację `UNION` wynosi $O(\log n)$
- łączny czas działania $O(n \log n + m)$

Cechy rozwiązania 2

Cechy rozwiązania 2

- szybki czas sprawdzania, czy dwa elementy należą do jednego zbioru $O(1)$
- łączenie jest szybsze, bo jeśli jakiemuś elementowi podmieniamy wskaźnik na reprezentantententanta, to nagle ten element znajduje się w zbiorze co najmniej dwa razy większym
- wniosek? każdy element ma podmieniany wskaźnik co najwyżej $O(\log n)$ razy
- cały czas łączenia wynosi więc $O(n \log n)$, więc **zamortyzowany czas** przeznaczony na jedną operację `UNION` wynosi $O(\log n)$
- łączny czas działania $O(n \log n + m)$

Da się jeszcze szybciej! - Rozwiązanie 3

Trzymamy rodzinę zbiorów rozłącznych jako las drzew

- każdy element oprócz korzenia w drzewie ma wskaźnik do jakiegoś elementu wyżej
- jeśli jest korzeniem, to ma wskaźnik sam do siebie
- reprezentantententantem zbioru jest korzeń drzewa
- scalanie = podpięcie korzenia jednego drzewa do drugiego drzewa

Da się jeszcze szybciej! - Rozwiązanie 3

Trzymamy rodzinę zbiorów rozłącznych jako las drzew

- każdy element oprócz korzenia w drzewie ma wskaźnik do jakiegoś elementu wyżej
- jeśli jest korzeniem, to ma wskaźnik sam do siebie
- reprezentantentantem zbioru jest korzeń drzewa
- scalanie = podpięcie korzenia jednego drzewa do drugiego drzewa

Da się jeszcze szybciej! - Rozwiązanie 3

Trzymamy rodzinę zbiorów rozłącznych jako las drzew

- każdy element oprócz korzenia w drzewie ma wskaźnik do jakiegoś elementu wyżej
- jeśli jest korzeniem, to ma wskaźnik sam do siebie
- reprezentantentantem zbioru jest korzeń drzewa
- scalanie = podpięcie korzenia jednego drzewa do drugiego drzewa

Da się jeszcze szybciej! - Rozwiązanie 3

Trzymamy rodzinę zbiorów rozłącznych jako las drzew

- każdy element oprócz korzenia w drzewie ma wskaźnik do jakiegoś elementu wyżej
- jeśli jest korzeniem, to ma wskaźnik sam do siebie
- reprezentantententantem zbioru jest korzeń drzewa
- scalanie = podpięcie korzenia jednego drzewa do drugiego drzewa

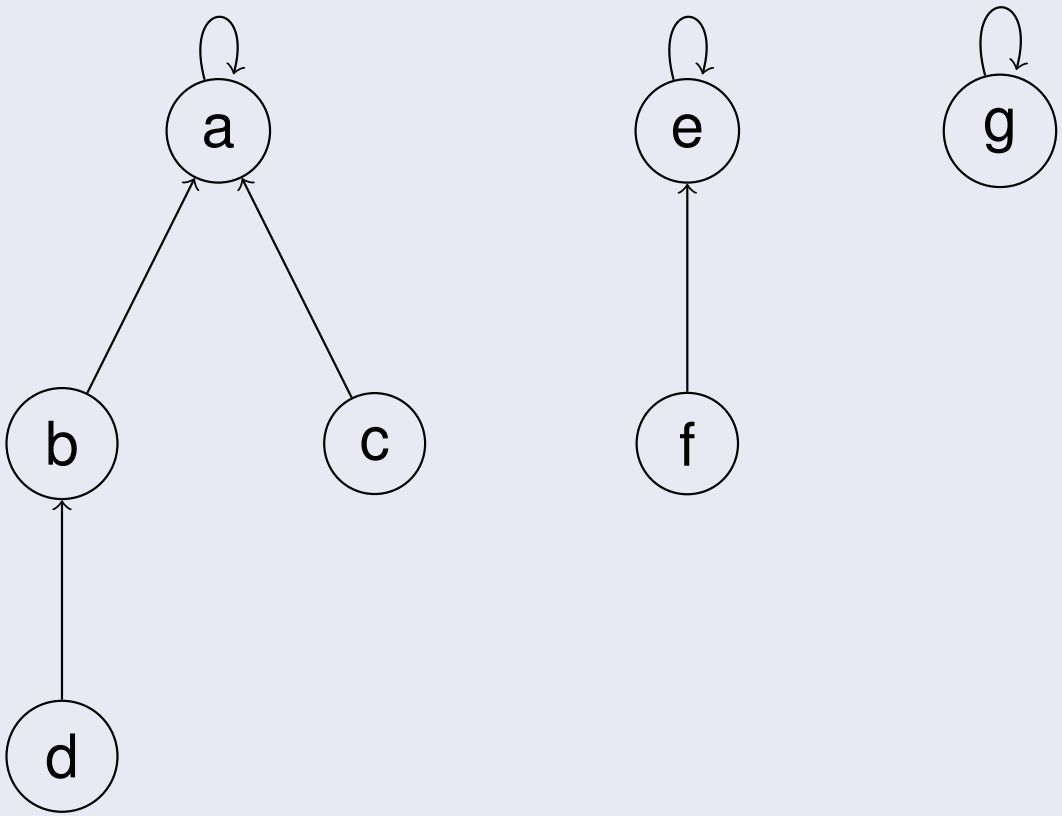
Da się jeszcze szybciej! - Rozwiązanie 3

Trzymamy rodzinę zbiorów rozłącznych jako las drzew

- każdy element oprócz korzenia w drzewie ma wskaźnik do jakiegoś elementu wyżej
- jeśli jest korzeniem, to ma wskaźnik sam do siebie
- reprezentantententantem zbioru jest korzeń drzewa
- scalanie = podpięcie korzenia jednego drzewa do drugiego drzewa

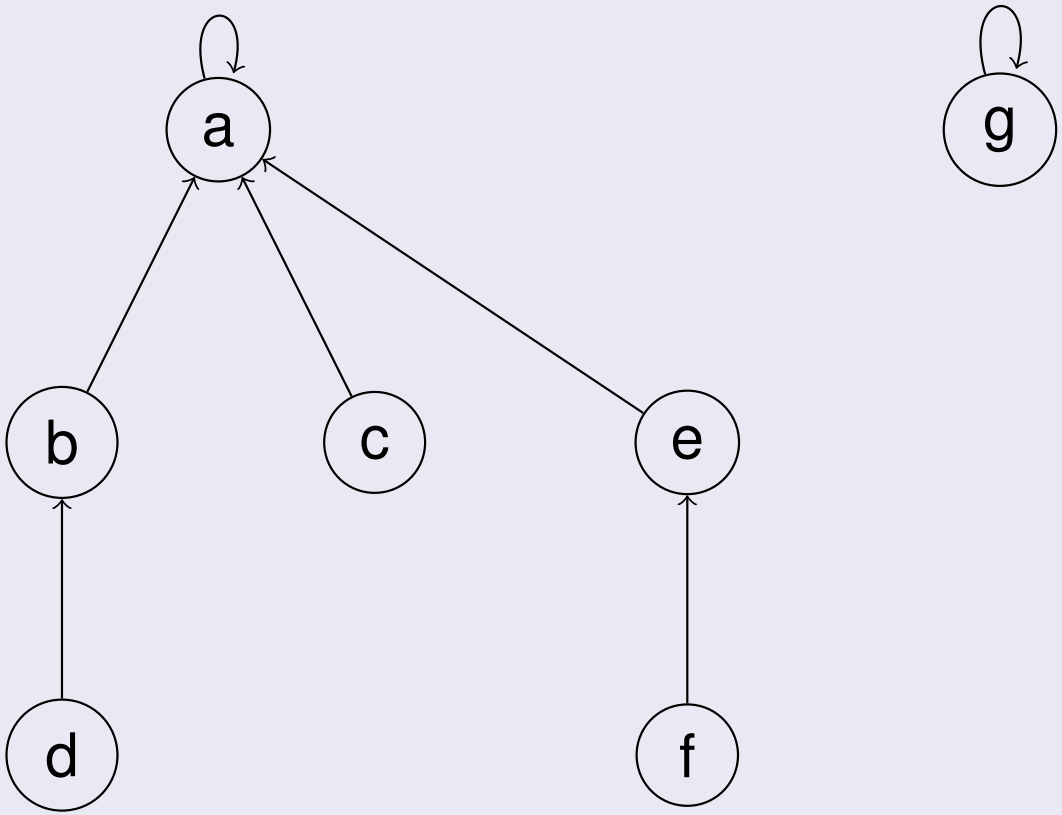
Przykład

Zasadniczo wygląda to tak:



Przykład po UNION

A jak zrobimy UNION(b,f), to wygląda tak:



Optymalizacja 1

Ranga

- dla każdego elementu z każdego zbioru trzymamy rangę, która jest wysokością drzewa, czyli odległością najdalszego elementu od korzenia
- scalanie zbiorów robimy podpinając do drzewa o większej randze drzewo o mniejszej randze
- jeśli rangi są równe, to podpinamy jakkolwiek i zwiększamy rangę w korzeniu drzewa, do którego podpięliśmy

Optymalizacja 1

Ranga

- dla każdego elementu z każdego zbioru trzymamy rangę, która jest wysokością drzewa, czyli odległością najdalszego elementu od korzenia
- scalanie zbiorów robimy podpinając do drzewa o większej randze drzewo o mniejszej randze
- jeśli rangi są równe, to podpinamy jakkolwiek i zwiększamy rangę w korzeniu drzewa, do którego podpięliśmy

Optymalizacja 1

Ranga

- dla każdego elementu z każdego zbioru trzymamy rangę, która jest wysokością drzewa, czyli odległością najdalszego elementu od korzenia
- scalanie zbiorów robimy podpinając do drzewa o większej randze drzewo o mniejszej randze
- jeśli rangi są równe, to podpinamy jakkolwiek i zwiększamy rangę w korzeniu drzewa, do którego podpięliśmy

Koszt czasowy

Koszt czasowy

- w omówionej dotąd implementacji koszt czasowy zapytania `FIND` jest logarytmiczny (warto zastanowić się dlaczego!)
- koszt operacji `UNION` to dwie operacje `FIND` plus czas podmienienia wskaźnika, czyli w sumie logarytmicznie
- całkowity koszt działania takiej struktury to $O(m \log n)$, czyli dość dużo
- na szczęście jest jeszcze ..

Koszt czasowy

Koszt czasowy

- w omówionej dotąd implementacji koszt czasowy zapytania `FIND` jest logarytmiczny (warto zastanowić się dlaczego!)
- koszt operacji `UNION` to dwie operacje `FIND` plus czas podmienienia wskaźnika, czyli w sumie logarytmicznie
- całkowity koszt działania takiej struktury to $O(m \log n)$, czyli dość dużo
- na szczęście jest jeszcze ..

Koszt czasowy

Koszt czasowy

- w omówionej dotąd implementacji koszt czasowy zapytania `FIND` jest logarytmiczny (warto zastanowić się dlaczego!)
- koszt operacji `UNION` to dwie operacje `FIND` plus czas podmienienia wskaźnika, czyli w sumie logarytmicznie
- całkowity koszt działania takiej struktury to $O(m \log n)$, czyli dość dużo
- na szczęście jest jeszcze ..

Koszt czasowy

Koszt czasowy

- w omówionej dotąd implementacji koszt czasowy zapytania `FIND` jest logarytmiczny (warto zastanowić się dlaczego!)
- koszt operacji `UNION` to dwie operacje `FIND` plus czas podmienienia wskaźnika, czyli w sumie logarytmicznie
- całkowity koszt działania takiej struktury to $O(m \log n)$, czyli dość dużo
- na szczęście jest jeszcze ..

Kompresja ścieżek

Kompresja ścieżek

- każdy element najlepiej czuje się blisko korzenia, bo wtedy nie trzeba za dużo chodzić po wskaźnikach w operacji `FIND`
- skoro już musimy tyle chodzić w operacji `FIND`, możemy poskracać ścieżki, czyli po prostu wszystkie napotkane na drodze wierzchołki podpiąć do korzenia
- całkowity czas działania struktury to $O(m \log^* n)$, gdzie $\log^* n$ oznacza wysokość stosu potęg dwójek potrzebnych do zbudowania n. Np. $\log^* 2 = 1$, $\log^* 2^2 = 2$, $\log^* 2^{2^2} = 3$,$\log^* 2^{2^{2^2}} = 4$, itd. Wartość $2^{2^{2^{2^2}}} = 2^{65536}$
- dla zainteresowanych: dowód tego faktu we „Wprowadzeniu do algorytmów“.

Kompresja ścieżek

Kompresja ścieżek

- każdy element najlepiej czuje się blisko korzenia, bo wtedy nie trzeba za dużo chodzić po wskaźnikach w operacji `FIND`
- skoro już musimy tyle chodzić w operacji `FIND`, możemy poskracać ścieżki, czyli po prostu wszystkie napotkane na drodze wierzchołki podpiąć do korzenia
- całkowity czas działania struktury to $O(m \log^* n)$, gdzie $\log^* n$ oznacza wysokość stosu potęg dwójek potrzebnych do zbudowania n. Np. $\log^* 2 = 1$, $\log^* 2^2 = 2$, $\log^* 2^{2^2} = 3$,$\log^* 2^{2^{2^2}} = 4$, itd. Wartość $2^{2^{2^{2^2}}} = 2^{65536}$
- dla zainteresowanych: dowód tego faktu we „Wprowadzeniu do algorytmów“.

Kompresja ścieżek

Kompresja ścieżek

- każdy element najlepiej czuje się blisko korzenia, bo wtedy nie trzeba za dużo chodzić po wskaźnikach w operacji `FIND`
- skoro już musimy tyle chodzić w operacji `FIND`, możemy poskracać ścieżki, czyli po prostu wszystkie napotkane na drodze wierzchołki podpiąć do korzenia
- całkowity czas działania struktury to $O(m \log^* n)$, gdzie $\log^* n$ oznacza wysokość stosu potęg dwójek potrzebnych do zbudowania n. Np. $\log^* 2 = 1$, $\log^* 2^2 = 2$, $\log^* 2^{2^2} = 3$,$\log^* 2^{2^{2^2}} = 4$, itd. Wartość $2^{2^{2^{2^2}}} = 2^{65536}$
- dla zainteresowanych: dowód tego faktu we „Wprowadzeniu do algorytmów“.

Kompresja ścieżek

Kompresja ścieżek

- każdy element najlepiej czuje się blisko korzenia, bo wtedy nie trzeba za dużo chodzić po wskaźnikach w operacji `FIND`
- skoro już musimy tyle chodzić w operacji `FIND`, możemy poskracać ścieżki, czyli po prostu wszystkie napotkane na drodze wierzchołki podpiąć do korzenia
- całkowity czas działania struktury to $O(m \log^* n)$, gdzie $\log^* n$ oznacza wysokość stosu potęg dwójek potrzebnych do zbudowania n. Np. $\log^* 2 = 1$, $\log^* 2^2 = 2$, $\log^* 2^{2^2} = 3$,$\log^* 2^{2^{2^2}} = 4$, itd. Wartość $2^{2^{2^{2^2}}} = 2^{65536}$
- dla zainteresowanych: dowód tego faktu we „Wprowadzeniu do algorytmów“.

Podsumowanie

Podsumowanie

Korzystając z przedstawionego algorytmu budowania drzewa rozpinającego, przy użyciu Rozwiązania 3, do reprezentowania zmieniających się w czasie spójnych składowych budujemy drzewo rozpinające w czasie $O(m \log m)$.