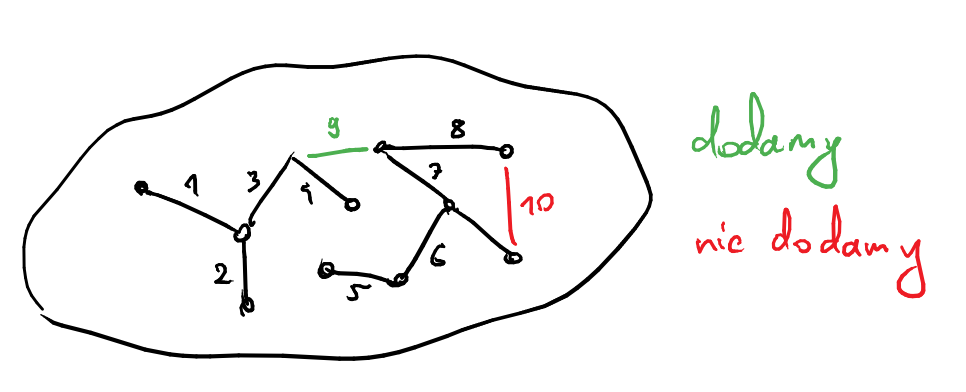
Union-Find

Algorytm Kruskala

- MST
- analizujemy wszystkie krawędzie od najmniejszej wagi (≤)
  - próbujemy je dołożyć do lasu drzew



(Union-find)

- uniwersum: {1..n}
- początek: n rozłącznych podzbiorów: {k}
- operacje:
  - union
  - find

I Rozwiązania naiwne

R[i] = i — id zbioru, w którym jest i

find(x) = R[x]

union(x,y): zastępujemy wszystkie wystąpienia x na y

- $O(n)$

Lepiej?: zbiór ~ lista elementów (ale R[i] dalej pamiętamy)

find: $O(1)$

union(x,y): $O(|y|)$ — |y| - liczebność zb. o id y

Jeszcze lepiej:

trzymać dł. list (żeby przemianowywać elementy mniej liczebnego zbioru)

find: $O(1)$

union(x,y): $T(x,y) = c \cdot \min\{|x|,|y|\} = O(n)$

! koszt zamortyzowany:

union(x,y): $O(\log n)$ [$n \log n$ na n elementów]

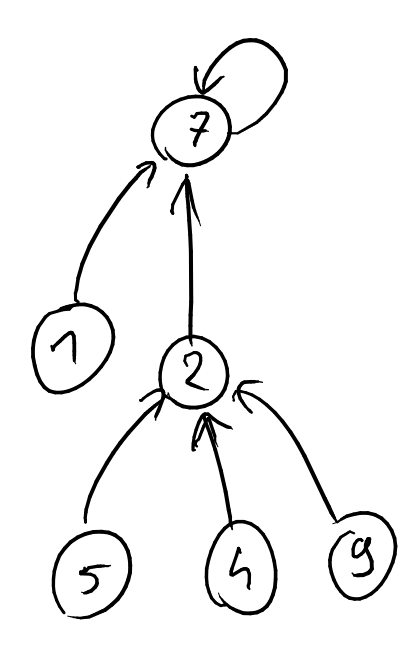
- element podłączany pod inny zbiór obarczamy kosztem → był w krótszej liście
- k razy obarczony → zbiór $\geq 2^k$

zbiór m → każdy element obarczony $\leq \log m$ razy

A gdyby rozleniwić?

II prawdziwy union-find (ale bez optymalizacji)

- struktura drzewa (będzie można zapisać w tablicy R[])
  - syn wskazuje na ojca
  - korzeń wskazuje na samego siebie
  - jedno drzewo ~ podzbiór



find(k): idziemy do korzenia
Θ(dł. ścieżki)

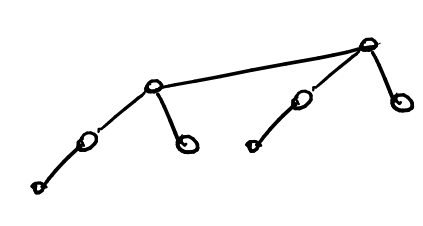
union(x,y): podepnij jedno drzewo pod drugie
$O(1)$ [jeśli dostajemy korzenie]

2 heurezy:

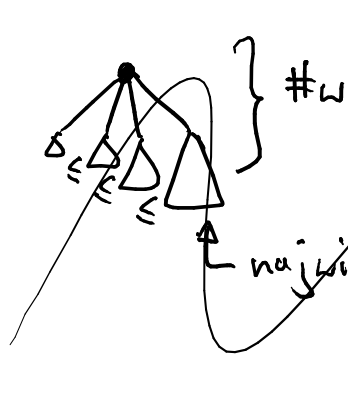
1) balansowanie union:
   podpinamy mniejsze drzewo pod większe
   - mniejsze: głębokość / #wierzchołków (wybór)

Dlaczego wystarczy patrzeć na #wierzchołków?

2) kompresja ścieżek
   wszystkie wierzchołki na znalezionej ścieżce z x do korzenia podpinamy bezpośrednio pod korzeń



T: $\#w \leq 2^k \Rightarrow$ głębokość $\leq k$



$\} \#w \leq 2^k$

największe:
1) $\leq 2^{k-1}$ → wszystkie $\leq 2^{k-1}$, gł. $\leq k-1$
2) $> 2^{k-1}$ → pozostałe $< 2^{k-1}$, gł. $\leq k-1$

T: $\forall k.\ \#w \leq 2^k \Rightarrow$ głębokość $\leq k$

Indukcja po wyprowadzeniu drzewa:

Drzewo powstało z union(t_1, t_2),

zał. b.s.o. $\#w_1 \leq \#w_2$

$\#w_2 \geq 2^{k-1}$

wtedy $\#w_1 \leq 2^{k-1}$, gł. $\leq k-1$

T: gł. $\geq k \Rightarrow \#w \geq 2^k$

Analiza zamortyzowana

Chcemy oszacować czas wykonania ciągu operacji ∈ {union, find} (#: n, #: m)

$\sigma = \{op_i\}$ $op_i \in$ {union, find}

koszt(σ) = koszt(union) + koszt(find), gdzie koszt(union) = $\Theta(n)$

Koszt find rozdzielamy pomiędzy nią i wierzchołki, które przechodzi.

Definicje:

$f(0) = 1$

$f(i+1) = 2^{f(i)}$

k	f(k)
0	1
1	2
2	4
3	16
4	2^{16}
5	$2^{\cdots}$

$\log^*(n) = \min\{k \mid f(k) \geq n\}$

k	$\log^*(k)$
0	0
1	0
2	1
3	2
4	2
5	3
6	3
⋮	
16	3
17	4
⋮	
2^{16}	4
…	5
⋮	⋮

σ' - ciąg σ bez operacji find

rząd(v) - wysokość drzewa (poddrzewa) o korzeniu v po wykonaniu σ

Fakt

wierzchołków o rzędzie = r jest nie więcej niż $\frac{n}{2^r}$.

Uzasadnienie

1) w drzewie, którego korzeń ma rząd r jest co najmniej 2^r wierzchołków
2) dwa drzewa, których korzenie mają ten sam rząd, są rozłączne

Fakt

W trakcie wykonywania σ: jeśli wierzchołek zyskuje nowego ojca, to ten ojciec ma większy rząd niż poprzedni.

Obserwacja

Rzędy rosną w górę ścieżki

Definicja

Rzędy dzielimy na grupy:
rząd r przypisujemy do grupy $\log^* r$

Przypisanie kosztów

find(x):
jeśli odwiedzamy wierzchołek v, to instrukcję find obciążamy kosztem 1 jeśli:
- v jest korzeniem
- lub synem korzenia (liczba takich na ścieżce: $O(1)$)
- lub v oraz jego ojciec mają rzędy w różnych grupach. ($O(\log^* n)$)

w p.p. kosztem obciążamy v.

Koszt wszystkich find:

Σ obciążeń find + Σ obciążeń wierzchołków

Σ obciążeń find = $O(m \cdot \log^* n)$

T: wierzchołki z jednej grupy są obciążone maksymalnie n razy (sumarycznie)

$\sum_{i=0}^{\log^* n}$ (suma obciążeń wierzch. z grupy i)

wierzchołek z grupy i