# Opracowanie publikacji o kolejkach priorytetowych autorstwa Gerth St$\phi$ting Brodal

Worst-Case efficient Priority Queues[2]

Jakub Banaszewski Krzysztof Sornat

15 lutego 2011

**Streszczenie**

Opisywana praca przedstawia model kolejki priorytetowej, w której operacje *MakeQueue, FindMin, Insert, Meld* i *DecreaseKey* wykonywane są w pesymistycznym czasie stałym. Operacje Delete i DeleteMin zajmują w najgorszym wypadku $O(\log n)$ operacji. Szczegóły, którymi uzupełniamy pracę wynikają z naszej interpretacji zawartych tam rozwiązań.

## 1 Wprowadzenie

W pracy przedstawiamy strukturę, która pozwala zaimplementować metody kolejki priorytetowej w wydajny sposób patrząc na ich pesymistyczną złożoność. Operacje które będziemy rozważać to :

**MakeQueue** Tworzy i zwraca pustą kolejkę priorytetową.

**FindMin**($Q$) Zwraca element najmniejszy kolejki.

**Insert**($Q, e$) Dodaje element $e$ do kolejki $Q$.

**Meld**($Q_1$,$Q_2$) Scala dwie kolejki $Q_1$, $Q_2$ w jedną. Zwraca kolejkę wynikową.

**DecreaseKey**($Q, e, e'$) Zmniejszenie wartości elementu $e$ w kolejce $Q$ na wartość $e'$.

**DeleteMin**($Q$) Usuwa z kolejki $Q$ i zwraca element minimalny kolejki.

**Delete**($Q, e$) Usuwa z kolejki $Q$ zadany element $e$.

W kopcach Fibonacci'ego, przedstawionych w [1], koszt zamortyzowany tych operacji jest optymalny. Chcemy uzyskać ten sam koszt w przypadku pesymistycznym. Ograniczenie dolne w modelu porównań na operacje *DeleteMin* zależy od operacji *Meld*. Jeśli *Meld* jest wykonywany w $o(n)$ wtedy *DeleteMin* nie może być wykonany w czasie $o(\log n)$ w modelu porównań. Jest to opisane w [3].

# 2 Struktura

Strukturę opieramy na dwóch drzewach $T_1$ i $T_2$. Każdy wierzchołek drzewa składa się z :

- wartości
- rangi
- synów (poddrzew)

O randze można myśleć jak o logarytmie z wielkości poddrzewa ukorzenionego w danym wierzchołku.

## 2.1 Konwencja w pracy

Aby omówić dalszą część struktury potrzebne jest kilka oznaczeń :

| | |
|---|---|
| $x, y, \ldots.$ | oznaczenia wierzchołków |
| $p(x)$ | ojciec x |
| $r(x)$ | ranga x |
| $n_i(x)$ | liczba synów wierzchołka $x$ rangi $i$. |
| $t_i$ | korzeń drzewa $T_i$ gdzie $i \in \{1, 2\}$ |

## 2.2 Synowie

Synowie wierzchołka są pamiętani w dwukierunkowej liście. Są oni przechowywani w kolejności względem swoich rang. Pierwszy wierzchołek ma najwyższą rangę, ostatni najniższą. U ojca jest pamiętany wskaźnik na początek listy, gdzie są trzymane poddrzewa o najwyższej randze. Synowie trzymają wskaźnik na swoich sąsiadów (wymóg listy dwukierunkowej) oraz na swojego ojca. Dodatkowo wierzchołki $t_1$ i $t_2$ posiadają tablice dynamiczne trzymające wskaźniki na synów konkretnej rangi. W ten sposób w czasie stałym mamy do nich dostęp[1].

## 2.3 Ranga wierzchołka

Ranga wierzchołka $x$ jest nieujemną wartością całkowitą, która spełnia niezmienniki:

**S1:** Jeśli $x$ jest liściem, wtedy $r(x) = 0$,

**S2:** $r(x) < r(p(x))$,

**S3:** Jeśli $r(x) > 0$ wtedy $n_{r(x)-1} \geqslant 2$,

**S4:** $n_i(x) \in \{0, 2, 3, \ldots, 7\}$,

**S5:** $T_2 = \varnothing \vee r(t_1) \leqslant r(t_2)$

Z tych warunków wynika kilka własności drzewa. Pierwsze dwa pokazują w jaki sposób rośnie wartość rangi. Niezmiennik **S3** gwarantuje, że ranga wierzchołka jest nie większa niż logarytm z wielkości poddrzewa, którego jest korzeniem. Jako uzasadnienie zauważmy, że "najuboższym" drzewem spełniającym ten niezmiennik (oraz przy okazji niezmiennik **S4**) jest pełne drzewo binarne.

[1] Najpierw znajdujemy syna konkretnej rangi w tablicy dynamicznej, a potem na liście dwukierunkowej przeszukujemy jego sąsiadów.

Warunek **S4** ogranicza ilość wierzchołków konkretnej rangi. Dzięki temu mamy gwarancję, że liczba synów wierzchołka jest nie większa niż logarytm z wielkości poddrzewa. Dodatkowo w dalszej części pracy zauważymy, że dzięki temu warunkowi operacje na drzewie będą dużo łatwiejsze (w szczególności brak możliwości posiadania tylko 1 syna określonej rangi). Przykładowo, gdy odetniemy od $x$ wierzchołki o randze $r(x) - 1$ nie będziemy mieli problemu ze znalezieniem co najmniej dwóch synów (warunek **S3**), które wyznaczą nową rangę wierzchołka[2].

## 2.4 Zbiory wierzchołków niezachowujących porządku kopcowego

W naszym modelu dopuszczamy zakłócenia porządku kopcowego[3] w drzewach $T_1$ oraz $T_2$. Liczba wierzchołków, które mogą ten porządek zakłócać to $\theta(n)$, co jest znacznym zwiększeniem względem innych rozwiązań. Niezgodności pamiętamy w zbiorach $W$ i $V$, które są trzymane w wierzchołkach.

W $W(x)$ i $V(x)$ znajdują się tylko wierzchołki $y$ takie, że $y > x$. Więc $x$ i $y$ są w dobrej relacji kopcowej. Informacje o wszystkich niezgodnościach są rozproszone w wierzchołkach w całym drzewie, chociaż w praktyce niezgodności, z których będziemy korzystać, są przechowywane w $W(t_1)$, $V(t_1)$, $W(t_2)$, $V(t_2)$.

**Uwagi:**

- Jeśli $y \in W(x) \cup V(x)$, to $x$ i $y$ mogą należeć do różnych drzew $T_1$, $T_2$.
- Każdy wierzchołek $y$ należy do co najwyżej jednego zbioru $V$ lub $W$[4].
- Nie twierdzimy, że jeśli $y \in W(x) \cup V(x)$, to $r(y) \leqslant r(x)$.

### 2.4.1 Struktura zbiorów $W$ i $V$

Zbiory $W$ i $V$ przechowujemy jako listy dwukierunkowe. Wierzchołki dodawane do $V$ umieszczamy na początku listy. Wierzchołki dodawane do $W$ wkładamy tak, aby na liście wierzchołki o tej samej randze były sąsiadami. Jeśli takiej rangi nie ma, to wstawiamy wierzchołek na początku listy $W$.

### 2.4.2 Implementacja zbiorów $W(x)$ i $V(x)$

Wierzchołek $x$ posiada 4 dodatkowe wskaźniki:

- jeden na pierwszy wierzchołek w $W(x)$,
- jeden na pierwszy wierzchołek w $V(x)$,
- po wskaźniku na następny i na poprzedni wierzchołek na liście niezgodności, do której należy $x$ (pewien zbiór $W(y)$ lub $V(y)$).

Wskaźniki z ostatniego podpunktu są puste jeśli $x$ zachowuje porządek kopcowy. Dodatkowo za każdym razem, gdy dodajemy wierzchołek do któregoś zbioru niezgodności zawsze najpierw usuwamy go ze zbioru, do którego należał wcześniej.

[2] Jeśli ich nie znajdziemy będzie to oznaczać, że wierzchołek stał się liściem.

[3] tzn. $x \geqslant p(x)$

[4] Innymi słowy: Jeśli $z \in \bigcup_x V(x) \cup W(x)$ to istnieje dokładnie jeden wierzchołek $y$ taki, że $(z \in V(y)) \dot{\vee} (z \in W(y))$.

### 2.4.3 Dokładny opis i niezmienniki zbiorów $W$ i $V$

$W(x)$ zawiera wierzchołki o niskich rangach, a $V(x)$ zawiera wierzchołki o wysokich rangach. Dokładniej: jeśli nowy wierzchołek $x$ generujący niezgodność ma rangę taką, że $r(x) \geqslant r(t_1)$, to $x$ dodajemy do $V(t_1)$. W przeciwnym przypadku wierzchołek $x$ dodajemy do $W(t_1)$.

Aby dodawać wierzchołek do zbioru $W$ w czasie stałym musimy wiedzieć wierzchołki jakiej rangi są w tym zbiorze. Dlatego informację o tym przechowujemy w tablicy dynamicznej[5] o rozmiarze $r(t_1)$ z wskaźnikami na wierzchołki odpowiednich rang. Wskaźniki te mogą być puste.

Oznaczenie $w_i(x)$ oznacza liczbę wierzchołków w $W(x)$ o randze $i$. Struktura $W$ i $V$ jest zachowywana przez następujące niezmienniki:

**O1:** $t_1 = \min(T_1 \cup T_2)$,

**O2:** jeśli $y \in V(x) \cup W(x)$, to $y \geqslant x$,

**O3:** jeśli $y < p(y)$, to $(\exists x \neq y)(y \in V(x) \cup W(x))$,

**O4:** $w_i(x) \leqslant 6$,

**O5:** jeżeli $V(x) = (y_{|V(x)|}, \ldots, y_2, y_1)$, to $(\forall_{i=1,2,\ldots,|V(x)|})(r(y_i) \geqslant \lfloor (i-1)/\alpha \rfloor)$ dla stałej $\alpha$ opisanej dalej.

### 2.4.4 Obserwacje

- Ranga wierzchołków $t_1$ i $t_2$ może zmieniać się (w szczególności rosnąć) w trakcie operacji na kolejce. Tak samo może się zmieniać zakres działania zbiorów $W$ i $V$ dla tych wierzchołków.
- Tablica wskaźników na wierzchołki konkretnych rang w zbiorze $W$ istnieje tylko dla wierzchołka $t_1$. W momencie gdy następuje zmiana wierzchołka $t_{1_{old}}$ na $t_{1_{new}}$ następuje też zmiana tablicy wskaźników. Stara jest usuwana, generowana jest nowa. Sam zbiór $W(t_{1_{old}})$ pozostaje nienaruszony[6]. Jednak wszelkie niezgodności będą teraz dodawane do zbioru $W(t_{1_{new}})$.
- Warunek **O2** ogranicza od dołu wartość zakłócenia. Dzięki temu po operacji *DeleteMin* wiemy, że przy szukaniu nowego minimum trzeba wziąć pod uwagę zbiory $V(t_1)$ i $W(t_1)$. Pozostałe zbiory trzymające zakłócenia nie mogą posiadać wierzchołka, który byłby nowym minimum.
- Warunek **O5** jest może mało czytelny. Ale ma on szereg konsekwencji. Blokuje on możliwość dodawania wierzchołków tej samej rangi[7] w dużej ilości. Wymusza też zwiększanie się rangi wierzchołka $t_1$ w celu zrobienia miejsca na nowe wierzchołki w zbiorze $V$. Dzięki temu mamy ograniczenie na wielkość zbioru $V(x)$ do $O(\log n)$ dla każdego $x$.
- Dzięki tablicy dynamicznej trzymającej wskaźniki na wierzchołki konkretnej rangi dla zbioru $W$ dodanie lub usuwanie elementu odbywa się w czasie stałym. Odwołujemy się do wierzchołka tej samej rangi co poszukiwany wierzchołek, a potem przeszukujemy jego sąsiadów na liście dwukierunkowej (z **O4** wiemy, że nie ma ich więcej niż 6).

[5] Zakładamy, że powiększenie tej tablicy kosztuje nas czas stały w pesymistycznym wypadku.

[6] Tak się dzieje w przypadku operacji *Meld*. Jeśli wykonujemy *DeleteMin* to zbiory $W(t_{1_{old}})$ i $W(t_{1_{new}})$ są łączone, a $t_{1_{old}}$ są usuwane. Dokładny opis tych operacji będzie w dalszej części opracowania.

[7] W szczególności wierzchołków małej rangi.

### 2.5 Dodatkowe warunki dla drzew $t_1$ i $t_2$

Ze względu na aktywne działanie zbiorów $V$ i $W$ dla tych wierzchołków oraz ich rolę korzenia przyjmujemy dodatkowe niezmienniki :

**R1:** $n_i(t_j) \in \{2, 3, 4, \ldots, 7\}$ dla $i = 0, 1, 2, \ldots, t(t_j) - 1$,

**R2:** $|V(t_1)| \leqslant \alpha r(t_1)$,

**R3:** Jeśli $y \in W(t_1)$ to $r(y) < r(t_1)$.

Niezmiennik **R1** gwarantuje występowanie wierzchołków każdej rangi dla korzeni drzew $T_1$ i $T_2$. **R2** wraz z **O5** powodują, że wzrost rangi $t_1$ o jeden powoduje możliwość dodania $\alpha$ niezgodności do zbioru $V(t_1)$ bez zakłócenia niezmienników. $\alpha$ jest stałą ustalaną na początku działania programu. Im jej wartość jest większa tym wygodniejsze jest korzystanie z operacji wykonywanych w czasie stałym na kolejce. Jednak przy wywołaniu *DeleteMin* będziemy musieli "zapłacić" za tą wygodę działając na zbiorze $V(t_1)$, którego wielkość jest zależna od stałej $\alpha$.

## 3 Guide - struktura zachowująca nałożone ograniczenia

W tym rozdziale opisana jest dodatkowa struktura danych - **Guide** - potrzebna do zachowania niezmienników **R1** i **O4**.

Rozwiążemy ogólniejszy problem. Załóżmy, że mamy ciąg $k$ liczb całkowitych $x_k, x_{k-1}, \ldots, x_1$[8]. Chcemy zachować niezmiennik $x_i \leqslant T$ dla każdego $i$ i pewnej stałej $T$. Na rozpatrywanym ciągu możemy użyć procedury **Reduce**$(i)$, która zmniejsza $x_i$ co najmniej o 2 i zwiększa $x_{i+1}$ co najwyżej o 1. $x_i$ może być zwiększane lub zmniejszane pojedynczo, ale przy każdej zmianie $x_i$ możemy wykonać $O(1)$ operacji **Reduce** [9], aby zapobiec przekroczeniu $T$ przez któregoś $x_i$. Zadaniem **Guide** jest wskazywanie w pesymistycznym czasie $O(1)$, które operacje należy wykonać.

Do działania **Guide** używamy ciągu $x'_k, x'_{k-1}, \ldots, x'_1$, który jest modyfikacją $x_k, x_{k-1}, \ldots, x_1$. $x'_k, x'_{k-1}, \ldots, x'_1$ jest taki, że $x_i \leqslant x'_i \in \{T-2, T-1, T\}$. Dopóki dla każdego $i$ zachodzi $x_i \leqslant x'_i$, to nie potrzebujemy pomocy **Guide**. Bez straty ogólności zakładamy, że $T = 2$ (więc $x'_i \in \{0, 1, 2\}$) oraz **Reduce**$(i)$ zmniejsza $x'_i$ o 2 i zwiększa $x'_{i+1}$ o 1.

**Guide** dzieli ciąg $x'_k, x'_{k-1}, \ldots, x'_1$ na bloki postaci $2, 1, 1, \ldots, 1, 0$, z założeniem, że w tym ciągu może nie być jedynek. Każda dwójka musi należeć do jakiegoś bloku tego typu, natomiast 0 i 1 mogą nie należeć do żadnego bloku. Przykładowy ciąg prezentujemy poniżej. Podkreślone liczby należą do tego samego bloku.

1 <u>2 1 1 0</u> 1 1 <u>2 0</u> <u>2 0</u> 1 0 <u>2 1 0</u>

**Guide** przechowuje wartości elementów $x'_i$ w tablicy i używa drugiej tablicy do pamiętania, który element jest w którym bloku. Druga tabela zawiera wskaźniki na indeksy poszczególnych $x_i$ lub na wartość nieokreśloną $\perp$. Wszystkie elementy w tym samym bloku wskazują na tą samą komórkę z numerem indeksu. Jest nią komórka reprezentowana przez liczbę najbardziej na lewo w bloku - czyli dwójkę. Elementy nie należące do bloku wskazują na komórkę z wartością $\perp$. Kilka elementów może wskazywać na tę samą komórkę z wartością $\perp$. Struktura danych dla

[8] Dla wygody ciągi zapisujemy od prawej do lewej. Przykładowe zastosowanie: na synów $t_1$ patrzymy tak, że ten o najwyższej randze jest pierwszy z lewej strony.

[9] W praktyce wykonamy najwyżej 2.

powyższego przykładu prezentowana jest na rysunku w orginalnej pracy. Ta struktura danych ma dwie bardzo ważne właściwości:

1. Mając dany element, jesteśmy w stanie znaleźć element w jego bloku, który jest najbardziej na lewo w czasie $O(1)$.

2. W czasie $O(1)$ możemy usunąć blok poprzez wpisanie wartości $\perp$ do komórki bloku. W ten prosty sposób elementy z tego bloku wskazują na wartość $\perp$.

Gdy zwiększamy $x'_i$, wtedy **Guide** może w czasie $O(1)$ zdecydować, jakie operacje **Reduce** należy wykonać.

Używając tablicy dynamicznej możemy rozszerzyć **Guide** o nową wartość $x_{k+1}$ równą 0 lub 1 w czasie $O(1)$ [10].

# 4 Operacje

Aby zrealizować podstawowe funkcje kolejki priorytetowej jest potrzebnych kilka operacji związanych z porządkowaniem i zachowywaniem własności struktury.

## 4.1 Łączenie i Rozłączanie drzew

Wymienione tutaj operacje będziemy wykonywać w czasie stałym. Większość z nich będzie dotyczyć wierzchołków $t_1$ i $t_2$.

### 4.1.1 Łączenie

Załóżmy, że mamy trzy wierzchołki $x_1$, $x_2$ i $x_3$ o równej randze. Żaden z nich nie jest $t_i$. Przy pomocy dwóch porównań możemy znaleźć minimum spośród tych wierzchołków. Załóżmy, bez utraty ogólności , że $x_1$ jest minimum. Zwiększamy rangę $x_1$ o jeden, a $x_2$ i $x_3$ stają się nowymi synami o najwyższej randze (niezmiennik **S3**). W wyniku tych operacji nie powstają żadne zakłócenia porządku kopcowego oraz zachowane są wszelki niezmienniki.

### 4.1.2 Rozłączanie

Zanim przejdziemy do mechanizmu warto wspomnieć, że synowie wierzchołka są na dwukierunkowej liście. Rozłączanie wierzchołków polega na "uwolnieniu" wierzchołków określonej rangi, które mogą zostać użyte w innym miejscu struktury. Jeśli $x$ ma dwóch lub trzech synów o randze $r(x)-1$ są oni odcinani, a $x$ przyjmuje rangę o jeden większą niż ranga nowego największego syna [11]. Dzięki niezmiennikowi **S4** wiemy, że jeśli znajdziemy wierzchołek na liście synów, to będzie miał on co najmniej jednego brata tej samej rangi, więc nie ma problemu z niezmiennikiem **S3** Jeśli $x$ ma czterech lub więcej synów rangi $r(x)-1$ po prostu odcinamy dwóch synów, a wszystkie niezmienniki pozostaną cały czas zachowane.

W wyniku rozłączenia dla drzewa o randze $k$ otrzymujemy dwa lub trzy wierzchołki rangi $k-1$ oraz jeden wierzchołek o randze co najwyżej $k$ (jest to wierzchołek na którym wykonujemy rozłączenie).

[10] Dzieję się to przy zwiększaniu się rangi wierzchołka używającego **Guide'a**. Nowa wartość równa 2 jest nieosiągalna w takim przypadku.

[11] Będzie to następny wierzchołek po odciętych w dwukierunkowej liście, w której trzymani są synowie $x$. Jeśli lista zostanie pusta, to wierzchołek stanie się liściem

## 4.2 Dbanie o synów korzenia

Do zachowania niezmiennika **R1** trzeba nadzorować synów $t_1$ i $t_2$. W tym celu korzystamy z czterech Guide'ów. Po dwóch na każdy korzeń. Aby mieć stały dostęp do synów $t_1$ korzystamy z dynamicznej tablicy wskaźników, które wskazują na któregoś z synów danej rangi. Dzięki **R1** wiemy, że każdy wskaźnik będzie niepusty. Przez to w czasie stałym możemy przeprowadzać operacje na synu $i$-tej rangi. Żeby mieć pewność, że nie naruszymy niezmiennika **R1** korzystamy z Guide'a aby zachować warunek $n_i(t_1) \leqslant 7$ oraz z drugiego Guide'a, żeby utrzymać $n_i(t_1) \geqslant 2$ dla $i = 0, \ldots, r(t_1) - 3$.

Synowie rangi $r(t_1) - 2$ i $r(t_1) - 1$ są pilnowani osobno. Wartości, których zmiana uruchamia Guide'a dla górnego ograniczenia to $\{5, 6, 7\}$, a dla dolnego ograniczenia to $\{2, 3, 4\}$. W przypadku dodania syna rangi $i$ do $t_1$ uruchamiamy Guide'a pilnującego górnego ograniczenia. Guide podpowiada gdzie wykonać redukcje[12]. Redukcja w tym wypadku to łączenie trzech drzew rangi $i$. To zmniejsza $n_i(t_1)$ o trzy i zwiększa $n_{i+1}(t_1)$ o jeden. Jeśli zmiana dotyczy synów rangi $r(t_1) - 2$ lub $r(t_1) - 1$ łączymy ich i jeśli to potrzebne zwiększamy rangę $t_1$. Jeśli zwiększymy rangę rozszerzamy też tablice dla Guide'ów.

Odcięcie syna jest analogiczne. Tym razem operacja redukcji polega na rozłączaniu (czyli powstaje dwóch synów konkretnej rangi). Dodatkowe drzewo z tej transformacji (to o nie określonej randze) po prostu dodajemy pod $t_1$ na takiej samej zasadzie, jak przy dodawaniu nowego syna.

W czasie tych operacji w $t_1$ nie powstają żadne nowe niezgodności. Natomiast w $t_2$ mogą się one tworzyć podczas operacji rozłączania drzew. Rozłączamy synów o randze większej od rangi $t_1$. Jeśli w wyniku powstanie drzewo o randze mniejszej od $r(t_1)$, to przenosimy je do $T_1$ i żadna nowa niezgodność nie powstaje. Jeśli w wyniku powstanie drzewo o randze większej od $r(t_1)$, to zostaje ono w $T_2$ i może powstać nowa niezgodność, która w tym przypadku trafia do zbioru $V$ wierzchołka $t_1$.

## 4.3 Redukcja zakłóceń

Dzięki "leniwemu" [13] scalaniu drzew i dopuszczalnym zakłóceniom możliwe jest wykonanie wielu operacji w czasie stałym. Potrzebny jest mechanizm do zmniejszania liczby zakłóceń. Transformacja poniżej opisana redukuje ilość potencjalnych zakłóceń[14] w drzewie o co najmniej jedno. Załóżmy, że mamy dwa zakłócenia $x_1$ i $x_2$ równej rangi $k < r(t_1)$. Dodatkowo przyjmujemy, że $x_1$ i $x_2$ nie są korzeniami, ani synami korzenia $t_1$ lub $t_2$. Zauważmy, że podczas operacji na kolejce mógł się zmienić ojciec któregoś z wierzchołków. Najpierw sprawdzamy, czy $x_1$ i $x_2$ ciągle są wierzchołkami zakłócającymi porządek.

Jeśli oba wierzchołki ciągle zakłócają porządek to staramy się doprowadzić do sytuacji kiedy $x_1$ i $x_2$ są braćmi.

Z **S4** wiadomo, że $x_1$ jak i $x_2$ moją jeszcze co najmniej jednego brata. Jeśli $x_1$ i $x_2$ nie są braćmi załóżmy (bez straty ogólności), że $p(x_1) \leqslant p(x_2)$. Zamieniamy poddrzewo $x_1$ z bratem $x_2$. W tym momencie możemy jedynie zmniejszyć ilość zakłóceń[15]. Teraz $x_1$ i $x_2$ są braćmi i mają tego samego ojca $y$.

Jeśli $x_1$ ma więcej niż jednego brata odcinamy $x_1$ i dołączamy go do $t_1$. Zmniejszamy ilość zakłóceń i ciągle zachowany jest niezmiennik **S4**.

[12] Najwyżej dwie.

[13] Scalanie drzew jest przeprowadzane stopniowo podczas innych operacji na kolejce.

[14] Zakłócenia wraz z czasem mogą przestać być aktualne. Mimo to są ciągle trzymane w zbiorach. Dlatego są to potencjalne zakłócenia.

[15] Przypadek gdy brat $x_2$ też był wierzchołkiem zakłócającym.

Jeśli $x_1$ i $x_2$ są jedynymi synami $y$ o randze $k$ i $r(y) > k+1$ odcinamy obu synów i podczepiamy ich pod $t_1$. Odcinając obu synów zachowujemy niezmiennik **S4**.

Pozostał tylko przypadek, kiedy $x_1$ i $x_2$ są jedynymi synami $y$ o randze $k = r(y) - 1$. W tym momencie odcinamy $x_1$, $x_2$ i $y$. W miejsce $y$ wstawiamy wierzchołek o tej samej randze z $t_1$. Dzięki mechanizmom pilnującym liczebność synów $t_1$ możemy to bez obawy zrobić w czasie stałym. Z warunku $S_2$ mamy gwarancję, że wierzchołek zastępujący $y$ nie może być jest przodkiem.

## 4.4 Unikanie za dużej ilości zakłóceń

By zachować warunki $O4 - O5$ i $R2$ potrzebne jest pilnowanie liczebności zbiorów $V$ i $W$. Jak było wspomniane, do zbioru $V$ są dodawane zakłócenia o randzie większej niż ranga $t1$, a do zbioru $W$ pozostałe.

### 4.4.1 Zbiór $W$

Dla zachowania warunku $O4$ używamy **Guide'a** na zbiorze $W$ (górne ograniczenie na liczebność). *Redukcja* z **Guide'a** w tym przypadku uruchamia operację redukcji zakłóceń opisaną w podrozdziale wyżej. Przeprowadzamy ją tylko wtedy, gdy jest 6 zakłóceń jednej rangi i przynajmniej 2 z nich nie są synami drzewa $t_2$. Jeśli więcej niż 4 wierzchołki są synami $t_2$ odcinamy nadmiarowe zakłócenia i przenosimy je pod $t_1$. Przy tej operacji dbamy, żeby nie uruchomić **Guide'a** dla $t_2$. Zauważmy, że zmiana ilości synów z 5 na 4 nie zmienia żadnej wartości obu **Guide'ów**. Z kolei zmiana z 6 na mniejszą wartość może spowodować zniszczenie bloku, do którego należała dla ograniczenia górnego. Jednak **Guide** ogranicza wartość od góry, więc pozostawienie w pamięci **Guide'a** wartości większej nie ma negatywnych konsekwencji.

### 4.4.2 Zbiór $V$

Aby nadążyć za zakłóceniami dodawanymi do $V$ dla każdej operacji wykonywanej na kolejce zwiększamy rangę wierzchołka $t_1$ o co najmniej 1 przenosząc stałą liczbę dzieci $t_2$ do $t_1$, dopóki $T_2 \neq \phi$. Dzięki temu zwiększamy "pojemność" zbioru $V$ (warunek $R2$ i $O5$) o $\alpha$. Jednocześnie "leniwie" realizujemy operację *Meld*.

Jeśli $r(t_2) = r(t_1) + 1$ to przenosimy synów $t_2$ o randze $r(t_1)$[16] do $t_1$ zwiększając jego rangę. Następnie zmniejszone $t_2$ podpinamy pod $t_1$. $T_2$ staje się pustym drzewem, a wszelkie nowe zakłócenia będą dodawane do $W$. Z warunku **S2** i tego, że $T_2 = \phi$ wiemy, że nie istnieją wierzchołki o wyższej randze od $t_1$, a tylko takie możemy dodawać do $V$.

W przeciwnym wypadku wykonujemy operację *rozłączania* na synu $t_2$ o randze $r(t_1) + 1$. 2 uwolnione wierzchołki o randze $r(t_1)$ *łączymy* z $t_1$ zwiększając jego rangę. Jeśli wierzchołek, który "rozłączaliśmy" zmniejszył rangę odcinamy go i dodajemy do $t_1$. W przeciwnym wypadku pozostawiamy w $t_2$ bez zmian.

## 4.5 Operacje kolejki priorytetowej

W tym rozdziale opisujemy jak zaimplementować operacje kolejki priorytetowej (patrz rozdział 1) tak, aby niezmienniki z rozdziału 2 były zachowane.

**MakeQueue** Zwracamy parę pustych drzew.

**FindMin($Q$)** Zwracamy $t_1$.

[16] są to synowie $t_2$ o najwyższej randze.

**Insert**$(Q, e)$ Jest szczególnym przypadkiem operacji **Meld**, gdzie $Q_2$ jest kolejką priorytetową zawierającą tylko element $e$.

**Meld**($Q_1$,$Q_2$) Mamy co najwyżej 4 drzewa, po 2 dla każdej z kolejek. Drzewo zawierające nowy element najmniejszy staje się drzewem $T_1$. Jest to drzewo $T_1$ kolejki $Q_1$ lub kolejki $Q_2$. Jeśli nowe drzewo $T_1$ ma największą rangę, to po prostu dodajemy pozostałe drzewa pod to drzewo. W tym przypadku nie tworzone są żadne niezgodności, więc nie musimy wykonywać żadnych transformacji na wierzchołkach niezgodnych. W przeciwnym przypadku (jeśli nowe drzewo $T_1$ nie ma największej rangi) drzewo o największej randze staje się nowym drzewem $T_2$. Pozostałe drzewa dodajemy do nowego drzewa $T_2$ jak zostało to opisane w rozdziale 4.2(?), wykonując co najwyżej jedno rozłączanie jeśli nowy syn ma tę samą rangę co $t_2$. Z niezgodnościami tworzonymi w tym przypadku postępujemy tak, jak zostało to opisane w rozdziale 4.4(?). Usuwamy wszystkie Guide'y oraz tablice używane przez korzenie, które teraz zostały dołączone po korzeniem $t_2$.

**DecreaseKey**$(Q, e, e')$ Zastępujemy wartość elementu $e$ na wartość $e'$. Jeśli $e'$ nie narusza porządku kopcowego kończymy operacje. Jeśli $e'$ zakłóca porządek kopcowy sprawdzamy czy nie jest mniejszy od $t_1$. Jeśli tak to zamieniamy wartości wierzchołków $e'$ i $t_1$ i sprawdzamy, czy podmieniona wartość $t_{1_{old}}$ nie zakłóca porządku kopcowego. Jeśli występuje zakłócenie z powodu nowej wartości $e$ (niezależnie czy podmieniliśmy $t_1$ czy nie) dorzucamy go do zbioru wierzchołków zakłócających zawartych $t_1$. Mechanizmy opisane w rozdziale **4.4** pozwalają nam pilnować odpowiedniej ilości niezgodności powstałych w ten sposób.

**DeleteMin**$(Q)$ Procedura usuwania elementu minimalnego podsumowuje to opracowanie, gdyż używa praktycznie wszystkich omówionych elementów kolejki.

1. Przenosimy synów $t_2$ na listę synów $t_1$ w czasie liniowym od ich ilości (czyli $O(\log n)$). Każdego dodajemy do listy synów w czasie stałym, ponieważ mając syna rangi $i$ wystarczy, że za pomocą tablicy wskaźników na syna danej rangi (którą posiada $t_1$) dodamy tego syna jako następnego na liście. Teraz $t_2$ ma rangę 0 i również dodajemy go do listy synów $t_1$. W ten sposób "opróżniamy" drzewo $T_2$.
2. Usuwamy $t_1$, zachowując jednak informacje zawarte w wierzchołku, które będą jeszcze potrzebne. Mamy $O(\log n)$ niezależnych drzew.
3. Szukamy nowego minimum w zbiorach $V(t_{1_{old}})$, $W(t_{1_{old}})$ i wśród $O(\log n)$ powstałych niezależnych drzew. $V(t_{1_{old}})$ i $W(t_{1_{old}})$ też mają rozmiar $O(\log n)$ więc nadal mieścimy się założonym czasie.
4. Jeśli minimum nie jest korzeniem żadnego z $O(\log n)$ niezależnych drzew, to zawiera się w którymś z nich [17]. Aby "wyciągnąć" to minimum jako nowy korzeń drzewa $T_1$ trzeba "załatać" po nim dziurę. W tym celu znajdujemy wśród drzew niezależnych wierzchołek o identycznej randze i podmieniamy go z nowym minimum. Może nam powstać w wyniku tej operacji jedna nowa niezgodność[18].
5. Drzewo, którego korzeniem jest nowe minimum to nasze nowe $T_1$. Podłączamy pod nie listę niezależnych drzew. Ustawiamy jego nową rangę. $T_2$ pozostaje puste.
6. Porządkujemy synów $T_1$ tak aby warunki **S1-S5**, **R1** i **R3** były zachowane. Robimy to za pomocą operacji łączenia i rozłączania drzew. Niezależnych drzew jest najwyżej

[17] W takim wypadku należy do zbioru $V(t_{1_{old}})$ lub $W(t_{1_{old}})$.

[18] Nowe minimum było wierzchołkiem zakłócającym kolejność. Jeśli podmieniony wierzchołek też będzie zakłócał kolejność powstanie nowa niezgodność.

$O(\log n)$ [19], więc takie też jest ograniczenie czasowe na wykonanie tej operacji. Na uporządkowanie każdej rangi potrzebujemy stałą liczbę operacji. Porządkujemy je od najmniejszej do największej. Po porządkowaniu tworzymy nowe Guide'y, które pilnują dwóch ograniczeń – dolnego i górnego (warunek **R1**).

7. Łączymy zbiory $W$ i $V$ starego minimum, zbiory $W$ i $V$ nowego minimum i tak nowo powstały zbiór staje się zbiorem $W$ nowego minimum. Jeśli powstała, dodajemy jedną niezgodność wymienioną w punkcie **4.**. Nowe $W$ ma rozmiar $O(\log n)$, bo każdy ze zbiorów, który wzięliśmy do sumy miał rozmiar $O(\log n)$. Nowe $V$ jest puste, bo nie istnieje wierzchołek o randze większej niż $t_1$. Tak więc ten krok również zajmuje nam $O(\log n)$ czasu.
8. Na $W$ wykonujemy $O(\log n)$ transformacji redukujących niezgodności (rozdział 4.3) w ten sposób, aby w $W$ było co najwyżej po jednym wierzchołku danej rangi. Dzięki tym operacjom mamy zachowane warunki **O1-O5** i **R2**.
9. Tworzymy Guide dla zbioru $W(t_{1_{new}})$. Z punktu widzenia **Guide'a** będzie to tablica samych 0.

**Delete($Q, e$)** Jeśli $c$ będzie najmniejszym możliwym elementem, to tę operację zastępujemy dwoma operacjami: **DecreaseKey($Q, e, c$); DeleteMin($Q$)**.

# 5 Szczegóły implementacji

W tym rozdziale opiszemy szczegóły implementacji omawianej kolejki priorytetowej.

Każdy wierzchołek reprezentujemy jako rekord, który zawiera:

- Wartość.
- Rangę wierzchołkach.
- Wskaźnik na: lewego brata, prawego brata, ojca oraz pierwszego syna z lewej (o najwyższej randze).
- Wskaźnik na pierwszy element w swoim zbiorze $V$.
- Wskaźnik na pierwszy element w swoim zbiorze $W$.
- Wskaźnik na następny i wskaźnik na poprzedni element na liście niezgodności, do której ten wierzchołek należy. Jest to pewien zbiór $W$ lub zbiór $V$ innego wierzchołka. Jeśli ten element jest pierwszy na liście, to wskaźnik na poprzedni wskazuje na samego siebie.

Mamy dodatkowo 3 tablice dynamiczne:

- Tablica wskaźników na synów $t_1$ o randze $i$ dla $i \in \{0, 1, \ldots, r(t_1) - 1\}$.
- Tablica wskaźników na synów $t_2$ o randze $i$ dla $i \in \{0, 1, \ldots, r(t_2) - 1\}$.
- Tablica wskaźników na wierzchołki w $W(t_1)$ o randze $i$ (te wskaźniki mogą mieć wartość NULL).

Mamy 5 Guide'ów:

[19] Różnych rang jest $O(\log n)$, a ilość wierzchołków jest stała.

- Dla górnej granicy (co najwyżej 7 wierzchołków) dla $n_i(t_1)$.
- Dla dolnej granicy (co najmniej 2 wierzchołki) dla $n_i(t_1)$.
- Dla górnej granicy (co najwyżej 7 wierzchołków) dla $n_i(t_2)$.
- Dla dolnej granicy (co najmniej 2 wierzchołki) dla $n_i(t_2)$.
- Dla górnej granicy (co najwyżej 6 wierzchołków) dla $w_i(t_1)$.

# Literatura


[1] *Wprowadzenie do algorytmów*, Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Stein Clifford

[2] *Worst Case Effcient Priority Queues*, Gerth Stϕlting Brodal
BRICSy, Department of Computer Science,
University of Ny Munkegade, DK-8000 Arhus C, Denmark
gerth@brics.dk

[3] *Fast meldable priority queues*, Gerth Stϕlting Brodal.
In Proc. 4th Workshop on Algorithms and Data Structures (WADS),
volume 955 of Lecture Notes in Computer Science,
pages 282–290. Springer Verlag, Berlin, 1995.