Ćwiczenia 4

Tw. Każdy graf o n wierzchołkach i m krawędziach, ma zbiór niezależny o rozmiarze co najmniej $\frac{n^2}{2m+n}$

Dowód opiera się na oszacowaniu rozmiaru zbioru niezależnego generowanego przez następujący algorytm:

1. niech V zbiór wierzchołków, $v \in V$, t.że $deg(v)$ jest minimalny,
2. $I = I \cup \{v\}$,
3. $E = E\ -$ krawędzie sąsiadujące z v lub $adj(v)$,
4. $V = V - \{v\} - adj(v)$,
5. powtarzaj kroki 1–4 tak długo, jak $V \neq \emptyset$.

Dla $n = 1$ algorytm zwraca optymalny zbiór niezależny. Dla grafów, w których $min(deg(v)) = |V| - 1$ (graf pełny), algorytm również zwraca dobry zbiór niezależny.

Niech $IND(n, m)$ oznacza rozmiar zbioru niezależnego zwróconego przez algorytm dla grafu o n wierzchołkach i m krawędziach. Należy pokazać, że (przy założeniu, że dla $i < n$, $j \leq m$: $IND(i, j) \geq \frac{i^2}{2j+i}$),

$$IND(n, m) \geq \frac{n^2}{2m + n}.$$

Niech v oznacza wierzchołek wybrany w kroku 1, $deg(v) = d$, $d < n - 1$, a liczba usuniętych krawędzi w kroku 3 będzie równa k.

$$IND(n, m) = 1 + IND(n - 1 - d, m - k) \geq 1 + \frac{(n - 1 - d)^2}{2(m - k) + (n - 1 - d)}$$

należy pokazać, że:

$$1 + \frac{(n - 1 - d)^2}{2(m - k) + (n - 1) - d} - \frac{n^2}{2m + n} \geq 0$$

$$= 1 + \frac{d^2 + d(2 - 2n) + (n - 1)^2}{2(m - k) + (n - 1) - d} - \frac{n^2}{2m + n} =$$

$$= \frac{(d^2 + d(1 - 2n) + n(n - 1) + 2(m - k))(2m + n)}{(2(m - k) + (n - 1) - d)(2m + n)} - \frac{n^2(2(m - k) + (n - 1) - d))}{(2m + n)(2(m - k) + (n - 1) - d)} =$$

$$= \frac{d^2(2m + n) + d(2m(1 - 2n) + n(1 - n)) + 2k(n(n - 1) - 2m) + 4m^2}{(2m + n)(2(m - k) + (n - 1) - d)}$$

Mianownik ułamka ma zawsze wartość > 0 ($m \geq k$, $n - 1 > d$), stąd należy jednie sprawdzić znak licznika:

- $2k(n(n-1)-2m) \geq 0$ — ponieważ, $k \geq 0$ i $n(n-1)/2 \geq m$,

  ponieważ $k \geq (d+1)d/2$ (razem z v usuwanymi są wszyscy jego sąsiedzi, którzy również mają stopień równy co najmniej d), można skorzystać z oszacowania:

$$d^2(2m+n) + d(2m(1-2n) + n(1-n)) + 2k(n(n-1)-2m) + 4m^2 \geq$$
$$\geq d^2(2m+n) + d(2m(1-2n) + n(1-n)) + d(d+1)(n(n-1)-2m) + 4m^2 =$$
$$= d^2n^2 - 4dmn + 4m^2 = (dn-2m)^2 \geq 0$$

Równoległe obliczanie wyrażeń arytmetycznych (+,*)

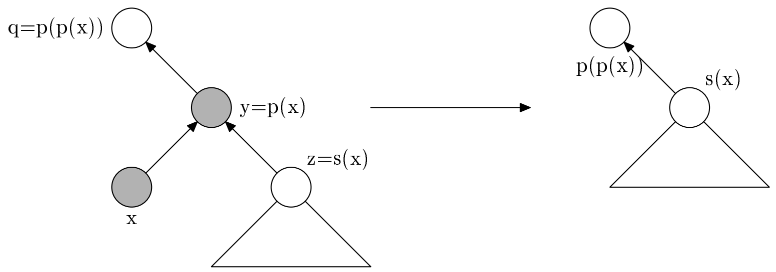
Dane: Dane poprawne drzewo wyrażeń arytmetycznych z operacjami `+` i `*`
Wynik: Pojedynczy węzeł z obliczonym wynikiem

Aby efektywnie rozwiązać ten problem, można zastosować technikę *tree contraction*:

1. inicjalizacja wartości w węzłach,
2. ponumerowanie wszystkich liści (np. za pomocą sum prefiksowych), za wyjątkiem dwóch skrajnych (tzn. pierwszego liścia od lewej i pierwszego od prawej),
3. ściskanie drzewa:
   - wykonanie procedury `rake` na wszystkich liściach o numerach nieparzystych, będących lewymi synami,
   - wykonanie procedury `rake` na wszystkich liściach o numerach nieparzystych, będących prawymi synami,
   - przenumerowanie liści, tzn. każdy liść, który nadal jest w drzewie, dzieli swój numer przez 2,
   - tak długo, jak drzewo będzie miało więcej niż 3 wierzchołki, należy powtarzać powyższą procedurę.
4. wykonanie ostatniego obliczenia i zwrócenie wyniku.

Operacja rake

Celem operacji *rake* jest wykonanie obliczeń na wskazanym fragmencie drzewa, a tym samym zmniejszeniem rozmiaru drzewa. W jej wyniku usuwane są dwa wierzchołki (wskazany liść i jego ojciec).
Bardziej dokładnie: wynikiem operacji $rake(x)$ (gdzie x jest liściem) jest usunięcie wierzchołków $\{x, parent(x)\}$, poprawienie wskaźnika *parent* dla brata x ($sibling(x)$) — jego nowym ojcem staje się $parent(parent(x))$. Należy również tak uaktualnić wyniki częściowe dla $s(x)$, by usuwanie $\{x, parent(x)\}$ nie przeszkadzało w wykonaniu zamierzonych obliczeń. Przykład działania operacji *rake* zobrazowany jest na rysunku 1.



Rys. 1. Efekt działania operacji $rake(x)$, szarym kolorem zaznaczono usuwane węzły.

Wyniki częściowe
Idea: gdyby ograniczyć zbiór dostępnych operacji jedynie do $+$, problem staje się bardzo prosty. Każdemu węzłowi jest przypisywany dodatkowy licznik s, który będzie zawierał sumę elementów z usuniętych fragmentów drzewa. Przy operacji $rake(x)$ do licznika s węzła $sibling(x)$ dodawane jest $(p(x).s + x.s + x.value)$.

Gdy dostępne są dwie operacje, sytuacja się komplikuje. W tym przypadku w charakterze "licznika" można użyć funkcji liniowej: $f(x) = ax + b$, reprezentowanej przez parę (a, b). Na początku działania algorytmu dla wszystkich węzłów drzewa ma ona wartość $f(x) = x$ — czyli (1,0).

Niech:

- $val(k)$ — oznacza wartość wyrażenia wyznaczonego przez poddrzewo o korzeniu w k, przy czym w obliczeniach uwzględniane są wyniki częściowe zapisane w k,
- $v(k)$ — wartość wyrażenia wyznaczonego przez poddrzewo o korzeniu w k, jednak bez uwzględniania wyników częściowych z k.

Powyższe wartości zdefiniowane są następująco:

- jeśli k jest liściem z zapisaną wartością y, to
$$v(k) = y,$$
- jeśli k jest węzłem wewnętrznym z przypisaną operacją $\otimes_k$, to
$$v(k) = val(k.left) \otimes_k val(k.right),$$
- jeśli węzłowi k przypisana jest funkcja $f_k(x) = a_k x + b_k$, to
$$val(k) = a_k v(k) + b_k.$$

W jaki sposób uaktualniać wyniki częściowe?
Przy tak zdefiniowanych v, val i warunkach początkowych, $val(root)$ odpowiada wartości zapisanego wyrażenia, dobrze byłoby gdyby w trakcie operacji $rake$ ta wartość nie ulegała zmianie.

W tym przypadku wystarczy zapewnić, że po operacji $rake(x)$ wartość $v(parent(parent(x))$ nie ulegnie zmianie.
Niech $q = parent(parent(x))$, $y = parent(x)$, $z = sibling(x)$, oraz $q.right = y$ i $y.right = z$ (tak jak to zostało zaznaczone na rysunku 1):

- przed operacją $rake(x)$,

$$v(q) = val(q.left) \otimes_q val(y) =$$
$$= val(q.left) \otimes_q (a_y(val(x) \otimes_y val(z)) + b_y) =$$
$$= val(q.left) \otimes_q (a_y(val(x) \otimes_y (a_z v(z) + b_z)) + b_y)$$

- po operacji $rake(x)$,

$$v'(q) = val(q.left) \otimes_q val(z) = val(q.left) \otimes_q (a'_z v(z) + b'_z),$$

  gdzie a'_z, b'_z oznaczają poprawione wartości wyników częściowych dla z.

Należy tak dobrać wartości a'_z, b'_z, by $v(q) = v'(q)$.

- Jeśli $\otimes_y = +$, to:

$$v(q) = val(q.left) \otimes_q (a_y(val(x) + (a_z v(z) + b_z)) + b_y) =$$
$$= val(q.left) \otimes_q ((a_y a_z)v(z) + (a_y val(x) + a_y b_z + b_y))$$

  ponieważ operacja $rake(x)$ nie wpływa na wartości $val(q.left)$ i $v(z)$, stąd wystarczy wybrać:

$$a'_z = a_y a_z,$$
$$b'_z = a_y val(x) + a_y b_z + b_y.$$

- analogicznie, jeśli $\otimes_y = *$, to:

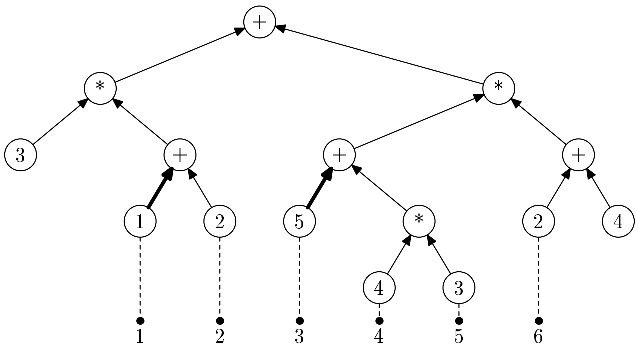
$$v(q) = val(q.left) \otimes_q (a_y(val(x) * (a_z v(z) + b_z)) + b_y) =$$
$$= val(q.left) \otimes_q ((a_y a_z val(x))v(z) + (a_y val(x) b_z + b_y))$$

  stąd:

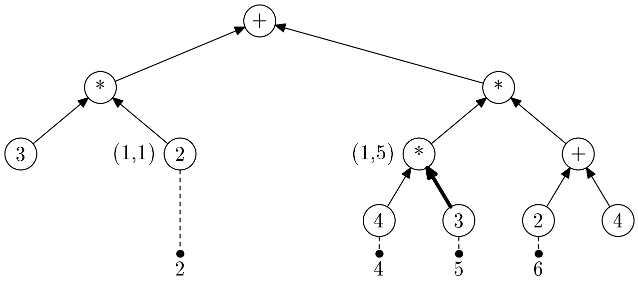
$$a'_z = a_y a_z val(x),$$
$$b'_z = a_y val(x) b_z + b_y.$$

Powyższe obliczenia dotyczą przypadku, gdy $q.right = y$ i $y.right = z$, jednak w pozostałych przypadkach otrzymane wzory są identyczne (dodawanie i mnożenie jest przemienne `;-)`)

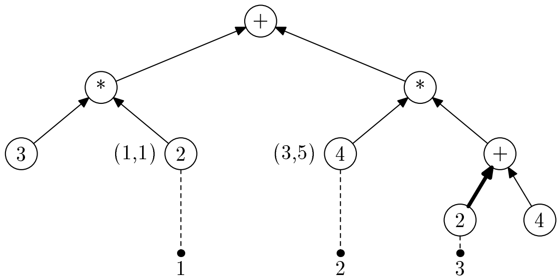
Przykład:



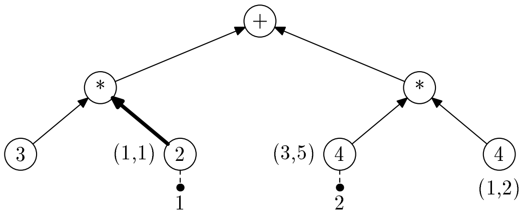
Rys. 2. Stan początkowy, kolorem pogrubionym zaznaczono liście które zostaną usunięte w następnym kroku



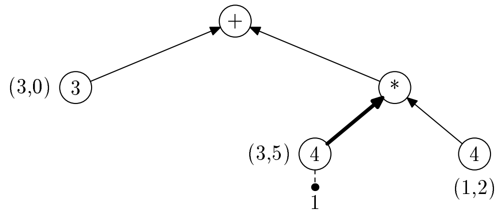
Rys. 3. Drzewo po "ściśnięciu" liści o numerach 1 i 3 (1 faza, wykonywanie *rake* dla lewych synów)



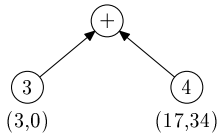
Rys. 4. Drzewo po *rake*(5) (1 faza, wykonywanie *rake* dla prawych synów), oraz przenumerowaniu liści



Rys. 5. Drzewo po *rake*(3) (2 faza, lewi synowie)



Rys. 6. Drzewo po $rake(1)$ (2 faza, prawi synowie), oraz przenumerowaniu liści



Rys. 7. Drzewo po $rake(1)$, wartość wyrażenia: $(3 \cdot 3+0)+(17 \cdot 4+34) = 918$

Równoległe obliczanie wyrażeń arytmetycznych (+,*,-,/))

Dane: Dane poprawne drzewo wyrażeń arytmetycznych z operacjami (+,-,*,/)
Wynik: Pojedynczy węzeł z obliczonym wynikiem

Problem rozwiązuje się podobnie jak poprzedni. Jednak tym razem funkcja wyników częściowych jest bardziej skomplikowana i ma postać:

$$f_i(x) = \frac{a_i x + b_i}{c_i x + d_i}.$$

Tak jak poprzednio węzły są inicjalizowane funkcją $f(x) = x$ (1,0,0,1). Niestety wzory na obliczanie nowych funkcji kosztu podczas operacji *rake* są nieco bardziej złożone, dodatkowo w przypadku operacji $-$ i $/$, występują w dwóch wariantach — gdy usuwany liść jest lewym synem oraz gdy jest prawym.

W przypadku $rake(x)$ ($y = parent(x)$, $z = sibling(x)$) wzory mają następującą postać:

- $\otimes_y = +$:

$$\begin{aligned}
a'_z &= a_y c_z val(x) + a_y a_z + b_y c_z, \\
c'_z &= c_y c_z val(x) + c_y a_z + d_y c_z, \\
b'_z &= a_y d_z val(x) + a_y b_z + b_y d_z, \\
d'_z &= c_y d_z val(x) + c_y b_z + d_y d_z,
\end{aligned}$$

- $\otimes_y = *$:

$$\begin{aligned}
a'_z &= a_y a_z val(x) + b_y c_z, \\
c'_z &= c_y a_z val(x) + d_y c_z, \\
b'_z &= a_y b_z val(x) + b_y d_z, \\
d'_z &= c_y b_z val(x) + d_y d_z,
\end{aligned}$$

- $\otimes_y = -$ (dla $y.right = z$):

$$a'_z = a_y c_z val(x) - a_y a_z + b_y c_z,$$
$$c'_z = c_y c_z val(x) - c_y a_z + d_y c_z,$$
$$b'_z = a_y d_z val(x) - a_y b_z + b_y d_z,$$
$$d'_z = c_y d_z val(x) - c_y b_z + d_y d_z,$$

- $\otimes_y = -$ (dla $y.left = z$):

$$a'_z = a_y c_z val(x) - a_y a_z - b_y c_z,$$
$$c'_z = c_y c_z val(x) - c_y a_z - d_y c_z,$$
$$b'_z = a_y d_z val(x) - a_y b_z - b_y d_z,$$
$$d'_z = c_y d_z val(x) - c_y b_z - d_y d_z,$$

- $\otimes_y = /$ (dla $y.right = z$):

$$a'_z = a_y c_z val(x) + b_y a_z,$$
$$c'_z = c_y c_z val(x) + d_y a_z,$$
$$b'_z = a_y d_z val(x) + b_y b_z,$$
$$d'_z = c_y d_z val(x) + d_y b_z,$$

- $\otimes_y = /$ (dla $y.right = x$):

$$a'_z = b_y c_z val(x) + a_y a_z,$$
$$c'_z = d_y c_z val(x) + c_y a_z,$$
$$b'_z = b_y d_z val(x) + a_y b_z,$$
$$d'_z = d_y d_z val(x) + c_y b_z,$$

Sprawdzanie poprawności wyrażeń nawiasowych, $O(n)$ procesorów EREW, czas $O(\log n)$

Dane: Wyrażenie nawiasowe, dane w postaci tablicy `A[1..n] of ['(',')']`
Wynik: Wynik zapisany w zmiennej `poprawne_wyr`

---

1. obliczenie sum prefiksowej dla tablicy B, zdefiniowanej następująco: $B[i] = 1$ wtw gdy $A[i] =$ `'('`, $B[i] = -1$ wpp.
2. niech *sum* oznacza tablicę z sumami prefiksowymi, $sum[i] = \sum_{j=1}^{i} B[j]$
3. jeśli $sum[n] \neq 0$, to wyrażenie nie jest poprawne (zawiera różną ilość nawiasów zamykających i otwierających) — koniec obliczeń,
4. `for i:=1 to n pardo if sum[i]<0 sum[i]=-1 else sum[i]=0`
5. sprawdzenie, czy tablica *sum* zawiera chociaż jeden element o wartości -1 (można to sprawdzić np. licząc sumę wszystkich elementów, jeśli jest ona mniejsza od zera, to istnieją takie elementy),
6. jeśli tablica *sum* zawiera same zera, to wyrażenie jest poprawne, wpp. wyrażenie jest błędne.

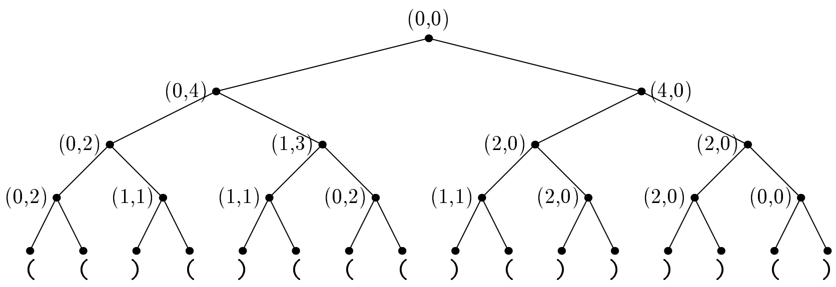
Znajdowanie odpowiadających sobie par nawiasów, $O(n)$ procesorów CREW, czas $O(\log n)$

Dane: Wyrażenie nawiasowe, dane w postaci tablicy `A[1..n] of ['(',')']`
Wynik: Tabica B, $B[i] = j$ wtw, gdy nawias otwierający i (zapisany w $A[i]$), ma odpowiadający mu nawias zamykający w $A[j]$

---

1. Zbudowanie drzewa zrównoważonego nad tablicą A. W każdym węźle drzewa przechowywana będzie informacja o niesparowanych nawiasach otwierających i zamykających z odpowiadającego mu fragmentu tablicy A, np. dla `())((()(`, będzie to `)(((`. Zawsze będzie to słowo postaci $)^i(^j$, więc można je reprezentować za pomocą pary (i, j).

   Na rysunku 8 przedstawione jest przykładowe drzewo wraz z wyliczonymi informacjami.

   Rys. 8. Drzewo z informacjami o niesparowanych nawiasach dla `(()()((()())))()`

   Informacje o niesparowanych nawiasach można wyrazić wzorem rekurencyjnym.

   Dla liści przyjmuje się odpowiednio $(1, 0)$ dla `)` i $(0, 1)$ dla `(`.

   Dla węzła wewnętrznego v, mającego synów l i r (o wartościach odpowiednio: (a_l, b_l) i (a_r, b_r)), jego wartość można wyrazić wzorem:

   $$(a_l + max(a_r - b_l, 0),\ b_r + max(b_l - a_r, 0)).$$

   (węzłowi v odpowiada słowo $)^{a_l}(^{b_l})^{a_r}(^{b_r}$, odejmowanie we wzorze oznacza "skracanie" nawiasów we fragmencie $(^{b_l})^{a_r}$)

   Utworzenie drzewa i wypełnienie wartościami jego wierzchołków można wykonać w czasie $O(\log n)$ na n procesorach CREW (właściwie to nawet EREW).

2. W drugim kroku każdy procesor, który ma przypisany nawias otwierający, szuka dla siebie odpowiadającego nawiasu zamykającego, przy czym wykorzystuje informacja zapisane w drzewie.

   Poszukiwanie dzieli się na dwie fazy:

- *up* — czyli przesuwanie się w kierunku korzenia tak długo, aż znaleziony zostanie węzeł, który zawiera w swoim poddrzewie poszukiwany nawias zamykający. Niech v oznacza aktualnie analizowany wierzchołek, zmienna x zawiera indeks (licząc od prawej) nawiasu otwierającego, od którego rozpoczęto wykonywanie procedury w słowie odpowiadającemu węzłowi x, np. dla słowa `))((((` i $x = 3$, wskazywany jest `))(`$\boxed{\texttt{(}}$`((`.

  Jeśli v jest prawym synem $parent(v)$, to należy przesunąć się do $parent(v)$, pozostawiając x bez zmian. Jeśli v jest lewym synem $parent(v)$, oraz $s = sibling(v)$, możliwe są dwa przypadki:

  - $x \leq a_s$ (gdzie (a_s, b_s) oznacza wartość zapisaną w s) — czyli odpowiedni nawias zamykający znajduje się w lewym poddrzewie, w takim wypadku cel został osiągnięty, wystarczy zwrócić parę (s, x),
  - $x > a_s$ — czyli w lewym poddrzewie jest zbyt mało niesparowanych nawiasów zamykających, a tym samym należy szukać wyżej, wcześniej modyfikując wartość x, $x := x - a_s + b_s$ (odejmuje się a_s, gdyż tyle nawiasów zamykających było w lewym poddrzewie $p(x)$ — zamykają one te, które stały przed x, dodaje b_s, gdyż z tego samego poddrzewa dochodzi tyle nowych nawiasów otwierających)

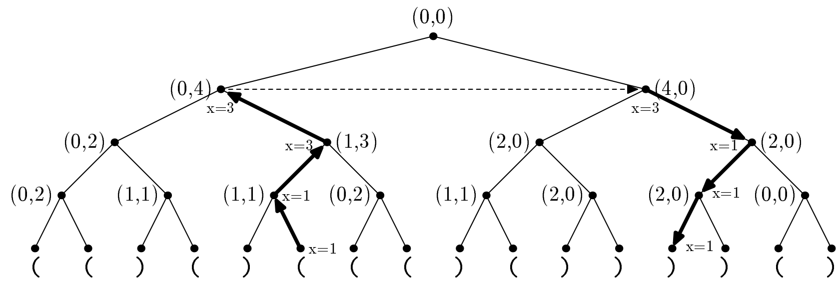
  function $up(v : vertex, x : integer)$ {
    if $p(v).right = v$
      return $up(p(v), x)$
    else
      $s = sibling(v)$
      $(a_s, b_s) = s.value$
      if $a_s \geq x$
        return (s, x)
      else
        return $up\,(p(v),\, x - a_s + b_s)$
  }

- *down* — zejście do liścia zawierającego odpowiedni nawias zamykający. Niech v oznacza aktualnie analizowany wierzchołek, poszukiwany jest (niesparowany) nawias zamykający o numerze x, możliwe są dwa przypadki:

  - $x \leq a_{v.left}$ (gdzie $(a_{v.left}, b_{v.left})$ oznacza wartość zapisaną w $v.left$) — czyli nawias znajduje się w lewym poddrzewie, w takim wypadku należy rekurencyjnie wywołać poszukiwanie dla $v.left$, pozostawiając wartość x bez zmian.
  - $x > a_{v.left}$ — czyli w lewym poddrzewie jest zbyt mało niesparowanych nawiasów zamykających, a tym samym poszukiwania należy przenieść do prawego poddrzewa, tym razem jednak należy zmodyfikować wartość x, $x := x - a_{v.left} + b_{v.left}$ (odejmuje się $a_{v.left}$, gdyż tyle nawiasów zamykających było w lewym poddrzewie, dodaje $b_{v.left}$, ponieważ tyle niesparowanych nawiasów zamykających z prawego poddrzewa jest otwieranych przez te z lewego poddrzewa)

```
function down(v : vertex, x : integer) {
  if v.leaf
    return v
  else
    (a_l, b_l) = v.left
    if a_l ≥ x
      return down(v.left, x)
    else
      return down (v.right, x − a_l + b_l)
}
```

Aby obliczyć nawias zamykający dla liścia v (zawierającego nawias otwierający), należy wykonać $down(up(v,1))$. Ponieważ czas działania procedur *up* i *down* jest ograniczony przez wysokość drzewa, stąd ten krok zajmuje $O(\log n)$ czasu.



Rys. 9. Przykład poszukiwania nawiasu zamykającego za pomocą *up* — strzałki w górę, i *down* — strzałki w dół, przy węzłach zaznaczono wartość zmiennej x na początku operacji *up*/*down*

Symulacja priority CRCW na common CRCW

Dane: Program używający p procesorów priority CRCW i m komórek pamięci
Wynik: Symulacja działania programu przy użyciu procesorów common CRCW

Symulacja 1 kroku priority CRCW na p^2 procesorach common CRCW i $m + p$ komórkach pamięci w czasie $O(1)$

Jedyną różnicą pomiędzy podanymi architekturami jest faza zapisu, stąd wystarczy wskazać sposób na symulowanie zapisów przy użyciu common CRCW. W symulacji jest wykorzystywana dodatkowa tablica $A[1..p]$. Każdemu procesorowi priority CRCW P_i przypisane jest p procesorów common CRCW $P'_{i,0..p-1}$.

Pojedyncza faza zapisu na priority CRCW:

1. jeśli procesor P_i chciałby zapisać pewną wartość do komórki M_j, to odpowiadający mu procesor $P'_{i,0}$ zapisuje $A[i] = j$, jeśli P_i w tej fazie nie wykonuje żadnych zapisów, to $P'_{i,0}$ zapisuje $A[i] = 0$,

2. dla każdego i: procesory odpowiadające za symulację P_i sprawdzają, czy w $A[1..i-1]$ nie ma odwołania do tej samej komórki pamięci, co i, jeśli tak, to anulują zapis wpisując 0 do $A[i]$,

   for $i = 1$ **to** p **pardo**
     if $A[i] > 0$ **then**
       for $j = 1$ **to** $i - 1$ **pardo**
         if $A[i] = A[j]$ **then** $A[i] = 0$

3. dla każdego i: jeśli $A[i] > 0$, to procesor $P'_{i,0}$ wykonuje odpowiedni zapis do $M_{A[i]}$.

Symulacja 1 kroku priority CRCW na $p \log p$ procesorach common CRCW i $mp + p$ komórkach pamięci w czasie $O(1)$

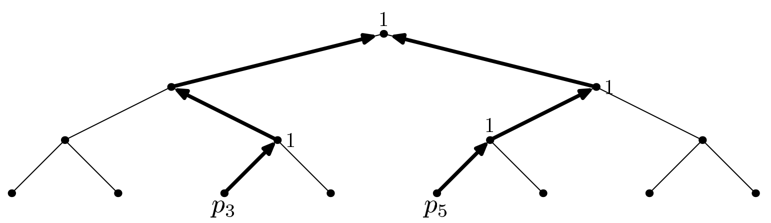
Tak jak poprzednio, wystarczy pokazać fazę zapisu. Tym razem dla każdego procesora priority CRCW P_i przypisane są $\log p$ procesorów. Dodatkowa pamięć jest wykorzystana do nadbudowania nad każdą oryginalną komórką pamięci zrównoważonego drzewa binarnego o p liściach.

Pojedyncza faza zapisu na priority CRCW:

1. jeśli procesor P_i chciałby zapisać pewną wartość do komórki M_j, to odpowiadające mu procesory $P'_{i,1..\log p}$ zaznaczają ten fakt w drzewie nad M_i.

   Zaznaczanie odbywa się w ten sposób, że dla wszystkich węzłów v na ścieżce pomiędzy korzeniem drzewa a liściem o numerze i, t. że liść i znajduje się w *lewym* poddrzewie v, zapisywana jest wartość 1.

   Ponieważ wszystkie zapisy do drzew wykonywane są równolegle, ten krok zajmuje $O(1)$ czasu.



Rys. 10. Efekt pierwszego kroku symulacji, w przypadku gdy procesory o numerach 3 i 5 chcą pisać do wspólnej komórki pamięci

2. teraz każdy z procesorów musi ustalić, czy ma uprawniania do wykonania zapisu (tzn. żaden procesor o mniejszym numerze nie chce pisać do tej samej komórki pamięci). Tu pomocna okazuje się następująca obserwacja: jeśli dla wszystkich węzłów v na ścieżce pomiędzy korzeniem drzewa a liściem o numerze i, t. że liść i znajduje się w *prawym* poddrzewie v, zapisana jest wartość 0, to znaczy, że procesor i ma najmniejszy numer spośród tych, które piszą do analizowanej komórki pamięci (jeśli byłoby inaczej, to istniałby procesor P_k ($k < i$), który spowodowałby zapis do $lca(k, i)$ (najniższy wspólny przodek k i i) wartości 1).

Dla każdej grupy procesorów $P'_{i,1..\log p}$ sprawdzenie powyższego warunku można wykonać w czasie $O(1)$ korzystając z jednej dodatkowej komórki pamięci (wystarczy wykonać operację OR na $\log p$ komórkach pamięci, z wykorzystaniem $\log p$ procesorów).

3. wszystkie procesory, które zapisywały coś do pomocniczych struktur (np. drzew), ustawiają wartości komórek, do których się odwoływały na 0.