Uogólnione drzewa Huffmana

czyli ang. lopsided trees

Juliusz Kopczewski

Seminarium Algorytmika 2009/2010

Plan prezentacji

Plan prezentacji

Kod prefiksowy

Kod

Funkcja $h : S \to W^+$ gdzie $S = \{s_1, ..., s_n\}$ jest zbiorem symboli do zakodowania, natomiast $W = \{w_1, ..., w_r\}$ to alfabet, przy pomocy którego kodujemy.

Kod prefiksowy (czy raczej bezprefiksowy)

Żaden symbol nie jest kodowany jako prefiks drugiego:
$\forall s, s' \in S, \alpha \in W^+ \quad h(s)\alpha \neq h(s')$

- Każde kodowanie prefiksowe jest jednoznaczne.
- Nie każde jednoznaczne kodowanie jest prefiksowe:
  - Np. kodowanie "bezsufiksowe":
    $S = \{a, b\}, W = \{0, 1\}, h(a) = 0, h(b) = 01$

Kod prefiksowy

Kod

Funkcja $h : S \to W^+$ gdzie $S = \{s_1, ..., s_n\}$ jest zbiorem symboli do zakodowania, natomiast $W = \{w_1, ..., w_r\}$ to alfabet, przy pomocy którego kodujemy.

Kod prefiksowy (czy raczej bezprefiksowy)

Żaden symbol nie jest kodowany jako prefiks drugiego:
$\forall s, s' \in S, \alpha \in W^+ \quad h(s)\alpha \neq h(s')$

- Każde kodowanie prefiksowe jest jednoznaczne.
- Nie każde jednoznaczne kodowanie jest prefiksowe:
  - Np. kodowanie "bezsufiksowe":
    $S = \{a, b\}, W = \{0, 1\}, h(a) = 0, h(b) = 01$

Kod prefiksowy

Kod

Funkcja $h : S \to W^+$ gdzie $S = \{s_1, ..., s_n\}$ jest zbiorem symboli do zakodowania, natomiast $W = \{w_1, ..., w_r\}$ to alfabet, przy pomocy którego kodujemy.

Kod prefiksowy (czy raczej bezprefiksowy)

Żaden symbol nie jest kodowany jako prefiks drugiego:
$\forall s, s' \in S, \alpha \in W^+ \quad h(s)\alpha \neq h(s')$

- Każde kodowanie prefiksowe jest jednoznaczne.
- Nie każde jednoznaczne kodowanie jest prefiksowe:
  - Np. kodowanie "bezsufiksowe":
    $S = \{a, b\}, W = \{0, 1\}, h(a) = 0, h(b) = 01$

Kod prefiksowy

Kod

Funkcja $h : S \to W^+$ gdzie $S = \{s_1, ..., s_n\}$ jest zbiorem symboli do zakodowania, natomiast $W = \{w_1, ..., w_r\}$ to alfabet, przy pomocy którego kodujemy.

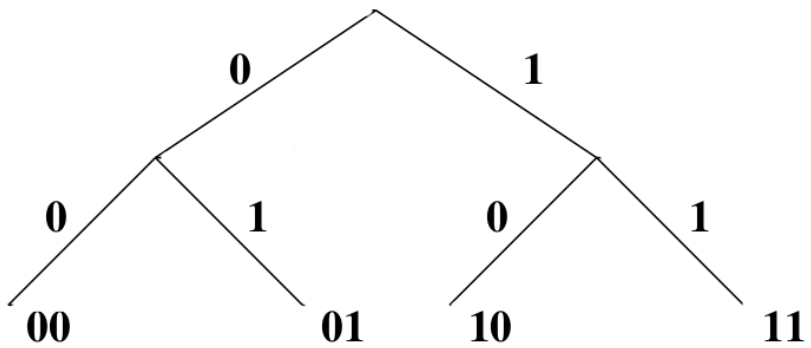
Kod prefiksowy (czy raczej bezprefiksowy)

Żaden symbol nie jest kodowany jako prefiks drugiego:
$\forall s, s' \in S, \alpha \in W^+ \quad h(s)\alpha \neq h(s')$

- Każde kodowanie prefiksowe jest jednoznaczne.
- Nie każde jednoznaczne kodowanie jest prefiksowe:
  - Np. kodowanie "bezsufiksowe":
    $S = \{a, b\}, W = \{0, 1\}, h(a) = 0, h(b) = 01$

Kod jako drzewo

- Kod prefiksowy najłatwiej przedstawić jako drzewo:
  - Krawędzie odpowiadają symbolom alfabetu.
  - Liście odpowiadają ciągom kodującym
- Nieco niedydaktyczny przykład:



Problem optymalnego kodu

Prawdopodobieństwo występowania symboli

Z każdym symbolem $s \in S$ dodatkowo wiążemy prawdopodobieństwo $p(s) \in [0,1]$ jego wystąpienia na wejściu.

Funkcja kosztu

Długość ścieżki zewnętrznej (ang. external path) drzewa kodu:
$\text{Koszt}(h) = \mathbb{E}h(s) = \Sigma_{s \in S} |h(s)| p(s)$

- Poszukujemy kodu o minimalnym koszcie.

Problem optymalnego kodu

Prawdopodobieństwo występowania symboli

Z każdym symbolem $s \in S$ dodatkowo wiążemy prawdopodobieństwo $p(s) \in [0,1]$ jego wystąpienia na wejściu.

Funkcja kosztu

Długość ścieżki zewnętrznej (ang. external path) drzewa kodu:
$\text{Koszt}(h) = \mathbb{E}h(s) = \Sigma_{s \in S} |h(s)| p(s)$

- Poszukujemy kodu o minimalnym koszcie.

Problem optymalnego kodu

- Tak sformułowany problem jest prosty: algorytm Huffmana.
- Podstawowa wersja działa dla $|W| = 2$, ale można uogólnić.
- Dlaczego to jest łatwe?
  - Mając dany zbiór ciągów kodujących wiemy jak optymalnie przypisać je do symboli.
  - Wiemy, że dwa najdłuższe ciągi kodujące mają ten sam koszt - możemy sklejać.

Problem optymalnego kodu

- Tak sformułowany problem jest prosty: algorytm Huffmana.
- Podstawowa wersja działa dla $|W| = 2$, ale można uogólnić.
- Dlaczego to jest łatwe?
  - Mając dany zbiór ciągów kodujących wiemy jak optymalnie przypisać je do symboli.
  - Wiemy, że dwa najdłuższe ciągi kodujące mają ten sam koszt - możemy sklejać.

Problem optymalnego kodu

- Tak sformułowany problem jest prosty: algorytm Huffmana.
- Podstawowa wersja działa dla $|W| = 2$, ale można uogólnić.
- Dlaczego to jest łatwe?
  - Mając dany zbiór ciągów kodujących wiemy jak optymalnie przypisać je do symboli.
  - Wiemy, że dwa najdłuższe ciągi kodujące mają ten sam koszt - możemy sklejać.

Problem optymalnego kodu

- Tak sformułowany problem jest prosty: algorytm Huffmana.
- Podstawowa wersja działa dla $|W| = 2$, ale można uogólnić.
- Dlaczego to jest łatwe?
  - Mając dany zbiór ciągów kodujących wiemy jak optymalnie przypisać je do symboli.
  - Wiemy, że dwa najdłuższe ciągi kodujące mają ten sam koszt - możemy sklejać.

Problem optymalnego kodu

- Tak sformułowany problem jest prosty: algorytm Huffmana.
- Podstawowa wersja działa dla $|W| = 2$, ale można uogólnić.
- Dlaczego to jest łatwe?
  - Mając dany zbiór ciągów kodujących wiemy jak optymalnie przypisać je do symboli.
  - Wiemy, że dwa najdłuższe ciągi kodujące mają ten sam koszt - możemy sklejać.

Plan prezentacji

Sformułowanie problemu

Nierówne koszty liter alfabetu

Koszt litery w oznaczamy jako $\text{length}(w) \in \mathbb{N}$.

Nowa funkcja kosztu

$\text{Koszt}(h) = \Sigma_{s \in S} \text{length}(h(s)) p(s)$

Sformułowanie problemu

Nierówne koszty liter alfabetu

Koszt litery w oznaczamy jako $\text{length}(w) \in \mathbb{N}$.

Nowa funkcja kosztu

$\text{Koszt}(h) = \Sigma_{s \in S} \text{length}(h(s)) p(s)$

Własności uogólnionego problemu

- Co wiadomo o problemie?
  - Niewiele: nie wiemy, czy jest NP-zupełny, nie znamy algorytmu wielomianowego.
- Rozwiązanie wielomianowe dla równych $p(s)$ (tzw. problem Varn codes).
- Rozwiązania wykładnicze dla przypadku $\text{length}(w_i) \in \mathbb{N}$ (wykładnik zależny **tylko** od kosztu liter).
- Schemat aproksymacyjny *PTAS*.

Własności uogólnionego problemu

- Co wiadomo o problemie?
  - Niewiele: nie wiemy, czy jest NP-zupełny, nie znamy algorytmu wielomianowego.
- Rozwiązanie wielomianowe dla równych $p(s)$ (tzw. problem Varn codes).
- Rozwiązania wykładnicze dla przypadku $\text{length}(w_i) \in \mathbb{N}$ (wykładnik zależny **tylko** od kosztu liter).
- Schemat aproksymacyjny *PTAS*.

Własności uogólnionego problemu

- Co wiadomo o problemie?
  - Niewiele: nie wiemy, czy jest NP-zupełny, nie znamy algorytmu wielomianowego.
- Rozwiązanie wielomianowe dla równych $p(s)$ (tzw. problem Varn codes).
- Rozwiązania wykładnicze dla przypadku $\text{length}(w_i) \in \mathbb{N}$ (wykładnik zależny **tylko** od kosztu liter).
- Schemat aproksymacyjny *PTAS*.

Własności uogólnionego problemu

- Co wiadomo o problemie?
  - Niewiele: nie wiemy, czy jest NP-zupełny, nie znamy algorytmu wielomianowego.
- Rozwiązanie wielomianowe dla równych $p(s)$ (tzw. problem Varn codes).
- Rozwiązania wykładnicze dla przypadku $\text{length}(w_i) \in \mathbb{N}$ (wykładnik zależny **tylko** od kosztu liter).
- Schemat aproksymacyjny *PTAS*.

Plan prezentacji

Algorytm naiwny

Pomysł 1

Nie działa metoda “bottom-up”, więc użyjemy metody “top-down”.

- Przeszukujemy dynamicznie przestrzeń wszystkich drzew kodów prefiksowych.
- Dokładamy kolejne krawędzie do liści drzew.
- Problem: wykładnik w złożoności będzie zależny od n oraz r.

Algorytm naiwny

Pomysł 1

Nie działa metoda “bottom-up”, więc użyjemy metody “top-down”.

- Przeszukujemy dynamicznie przestrzeń wszystkich drzew kodów prefiksowych.
- Dokładamy kolejne krawędzie do liści drzew.
- Problem: wykładnik w złożoności będzie zależny od n oraz r.

Algorytm naiwny

Pomysł 1

Nie działa metoda “bottom-up”, więc użyjemy metody “top-down”.

- Przeszukujemy dynamicznie przestrzeń wszystkich drzew kodów prefiksowych.
- Dokładamy kolejne krawędzie do liści drzew.
- Problem: wykładnik w złożoności będzie zależny od n oraz r.

Algorytm "prawie" naiwny

Pomysł 2

Wystarczy rozpatrywać drzewa z pełnymi wierzchołkami wewnętrznymi.

- Dodajemy sztuczne symbole z prawdopodobieństwem występienia $p(s) = 0$.

Algorytm "prawie" naiwny

Pomysł 2

Wystarczy rozpatrywać drzewa z pełnymi wierzchołkami wewnętrznymi.

- Dodajemy sztuczne symbole z prawdopodobieństwem występienia $p(s) = 0$.

Algorytm $O(n^{C+2})$

Pomysł 3

Wystarczy pamiętać liczbę wierzchołków na poszczególnych poziomach, zamiast struktury drzewa.

Pomysł 4

Możemy rozwijać liście w kolejności rosnącego length(s).

- Wystarczy tylko pamiętać:
  - Liczbę wierzchołków na ostatnich C poziomach
  - Liczbę pozostałych wierzchołków.

Algorytm $O(n^{C+2})$

Pomysł 3

Wystarczy pamiętać liczbę wierzchołków na poszczególnych poziomach, zamiast struktury drzewa.

Pomysł 4

Możemy rozwijać liście w kolejności rosnącego length(s).

- Wystarczy tylko pamiętać:
  - Liczbę wierzchołków na ostatnich C poziomach
  - Liczbę pozostałych wierzchołków.

Algorytm $O(n^{C+2})$

Pomysł 3

Wystarczy pamiętać liczbę wierzchołków na poszczególnych poziomach, zamiast struktury drzewa.

Pomysł 4

Możemy rozwijać liście w kolejności rosnącego length(s).

- Wystarczy tylko pamiętać:
  - Liczbę wierzchołków na ostatnich C poziomach
  - Liczbę pozostałych wierzchołków.

Algorytm $O(n^{C+2})$, c.d.

- Pozostaje mądrze zdefiniować funkcję kosztu.

Koszt drzewa

$\text{Koszt}(m; l_1, ..., l_C) = \Sigma_{i=1}^{m} \text{length}(v_i) p(s_i) + L \cdot \Sigma_{i=m+1}^{n} p(s_i)$

- Obcinamy drzewo na poziomie L.
- Liczymy sumę długości części ścieżek do liści do poziomu L włącznie.

Koszt krawędzi

Niech $T = (m; l_1, ..., l_C)$ oraz
$T' = T$ z rozwiniętymi q wierzchołkami na poziomie l_1
$\text{Koszt}(T, T') = \text{Koszt}(T) - \text{Koszt}(T') = \Sigma_{i=m+1}^{n} p(s_i)$

Algorytm $O(n^{C+2})$, c.d.

- Pozostaje mądrze zdefiniować funkcję kosztu.

Koszt drzewa

$$\text{Koszt}(m; l_1, \ldots, l_C) = \Sigma_{i=1}^{m} \text{length}(v_i) p(s_i) + L \cdot \Sigma_{i=m+1}^{n} p(s_i)$$

- Obcinamy drzewo na poziomie L.
- Liczymy sumę długości części ścieżek do liści do poziomu L włącznie.

Koszt krawędzi

Niech $T = (m; l_1, \ldots, l_C)$ oraz
$T' = T$ z rozwiniętymi q wierzchołkami na poziomie l_1
$\text{Koszt}(T, T') = \text{Koszt}(T) - \text{Koszt}(T') = \Sigma_{i=m+1}^{n} p(s_i)$

Algorytm $O(n^{C+2})$, c.d.

- Pozostaje mądrze zdefiniować funkcję kosztu.

Koszt drzewa

$$\text{Koszt}(m; l_1, \ldots, l_C) = \Sigma_{i=1}^{m} \text{length}(v_i) p(s_i) + L \cdot \Sigma_{i=m+1}^{n} p(s_i)$$

- Obcinamy drzewo na poziomie L.
- Liczymy sumę długości części ścieżek do liści do poziomu L włącznie.

Koszt krawędzi

Niech $T = (m; l_1, \ldots, l_C)$ oraz
$T' = T$ z rozwiniętymi q wierzchołkami na poziomie l_1
$\text{Koszt}(T, T') = \text{Koszt}(T) - \text{Koszt}(T') = \Sigma_{i=m+1}^{n} p(s_i)$

Algorytm $O(n^{C+2})$, c.d.

- Pozostaje mądrze zdefiniować funkcję kosztu.

Koszt drzewa

$\text{Koszt}(m; l_1, ..., l_C) = \Sigma_{i=1}^{m} \text{length}(v_i) p(s_i) + L \cdot \Sigma_{i=m+1}^{n} p(s_i)$

- Obcinamy drzewo na poziomie L.
- Liczymy sumę długości części ścieżek do liści do poziomu L włącznie.

Koszt krawędzi

Niech $T = (m; l_1, ..., l_C)$ oraz
$T' = T$ z rozwiniętymi q wierzchołkami na poziomie l_1
$\text{Koszt}(T, T') = \text{Koszt}(T) - \text{Koszt}(T') = \Sigma_{i=m+1}^{n} p(s_i)$

Analiza złożoności

- Wszystkich stanów, jest $O(n^{C+1})$.
- Przetworzenie pojedynczego stanu wymaga rozpatrzenia $n+1$ możliwych jego rozwinięć.
- Po przemnożeniu otrzymujemy $O(n^{C+2})$.

Analiza złożoności

- Wszystkich stanów, jest $O(n^{C+1})$.
- Przetworzenie pojedynczego stanu wymaga rozpatrzenia $n+1$ możliwych jego rozwinięć.
- Po przemnożeniu otrzymujemy $O(n^{C+2})$.

Analiza złożoności

- Wszystkich stanów, jest $O(n^{C+1})$.
- Przetworzenie pojedynczego stanu wymaga rozpatrzenia $n+1$ możliwych jego rozwinięć.
- Po przemnożeniu otrzymujemy $O(n^{C+2})$.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Oznaczenia: $\text{length}(w_1) = \alpha$, $\text{length}(w_2) = \beta$.

Pomysł 1

Chcemy zmniejszyć liczbę wymiarów przestrzeni, którą będziemy przeszukiwali.

Pomysł 2

Wracamy do konstrukcji "bottom-up".

Definicja

Przez ciąg charakterystyczny drzewa kodu rozumiemy:
$B(T)_i = b_i = |\{v \in V : \text{depth}(v) \geq i \wedge v \text{ jest prawym synem}\}|$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Oznaczenia: $\text{length}(w_1) = \alpha$, $\text{length}(w_2) = \beta$.

Pomysł 1

Chcemy zmniejszyć liczbę wymiarów przestrzeni, którą będziemy przeszukiwali.

Pomysł 2

Wracamy do konstrukcji "bottom-up".

Definicja

Przez ciąg charakterystyczny drzewa kodu rozumiemy:
$B(T)_i = b_i = |\{v \in V : \text{depth}(v) \geq i \wedge v \text{ jest prawym synem}\}|$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Oznaczenia: $\text{length}(w_1) = \alpha$, $\text{length}(w_2) = \beta$.

Pomysł 1

Chcemy zmniejszyć liczbę wymiarów przestrzeni, którą będziemy przeszukiwali.

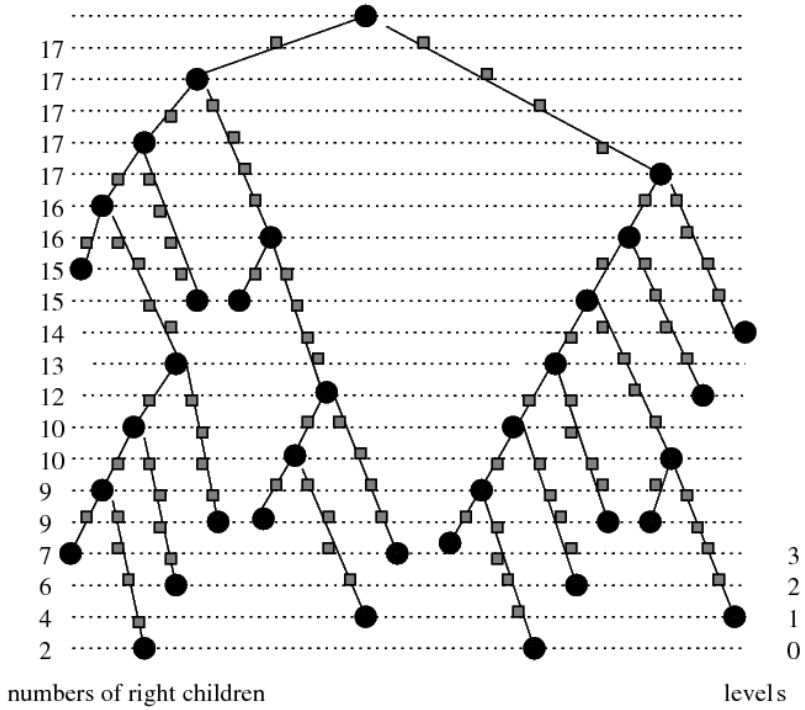
Pomysł 2

Wracamy do konstrukcji "bottom-up".

Definicja

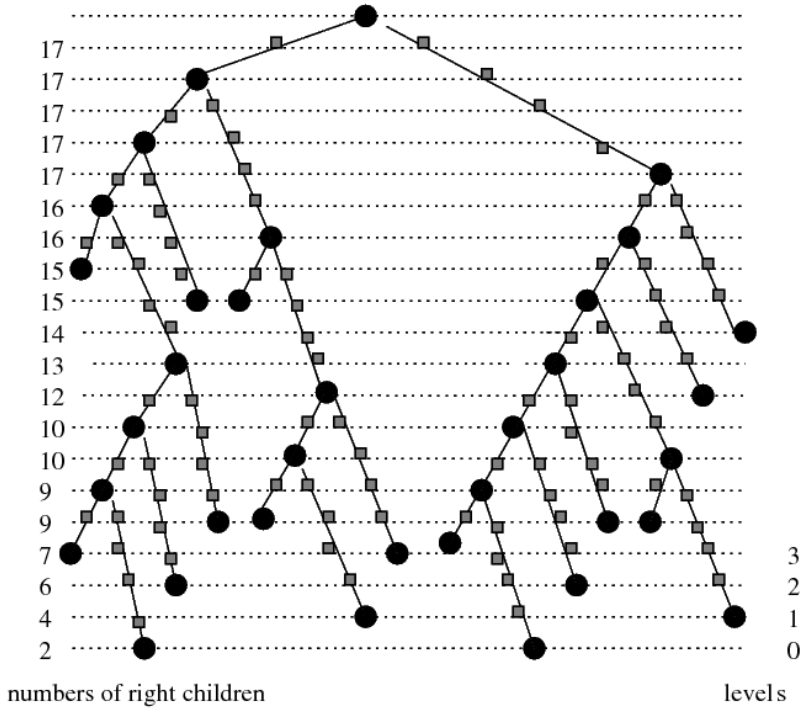
Przez ciąg charakterystyczny drzewa kodu rozumiemy:
$B(T)_i = b_i = |\{v \in V : \text{depth}(v) \geq i \wedge v \text{ jest prawym synem}\}|$

Algorytm $O(n^{C+1})$ dla $|W| = 2$



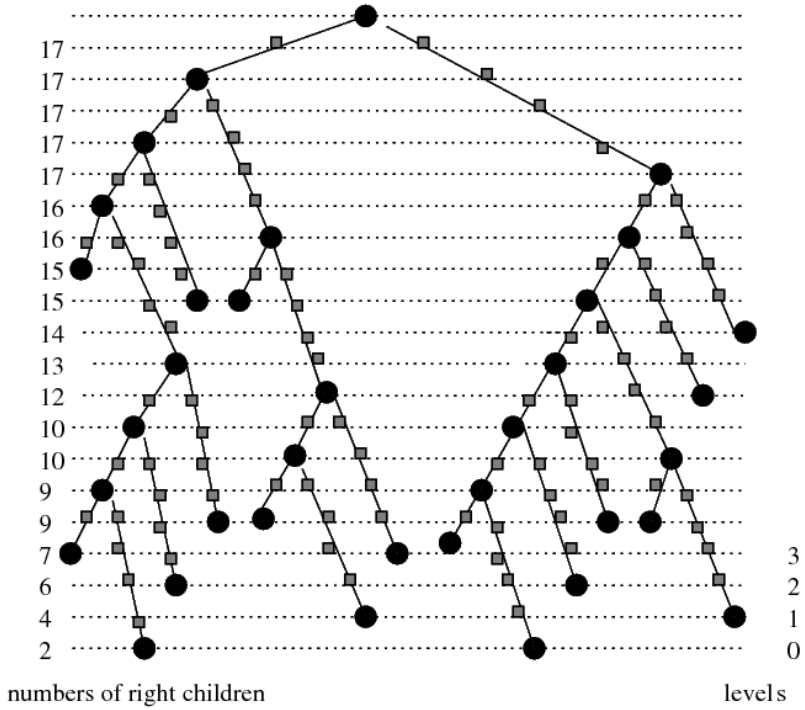
- Ile informacji da się wyciągnąć z takiego ciągu?
  - Znamy najgłębszy wierzchołek
  - Możemy obliczyć indeks jego brata w drzewie: $b_{\beta-\alpha-1}+1$

Algorytm $O(n^{C+1})$ dla $|W| = 2$



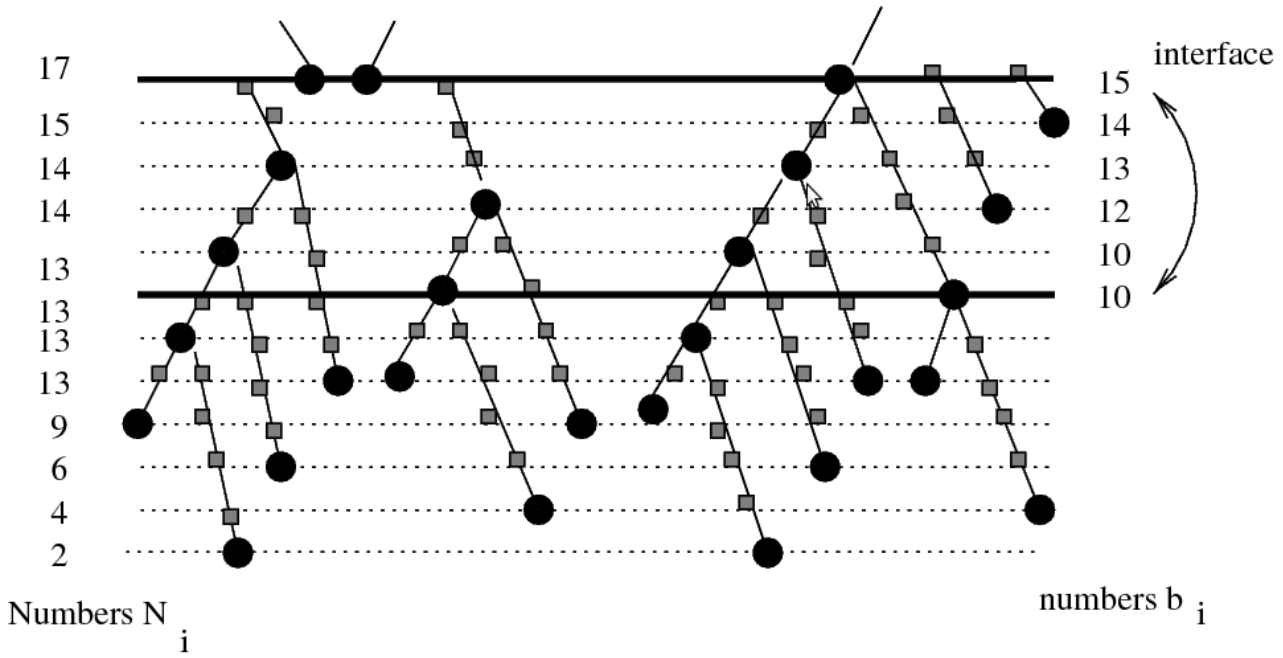
- Ile informacji da się wyciągnąć z takiego ciągu?
  - Znamy najgłębszy wierzchołek
  - Możemy obliczyć indeks jego brata w drzewie: $b_{\beta-\alpha-1}+1$

Algorytm $O(n^{C+1})$ dla $|W| = 2$



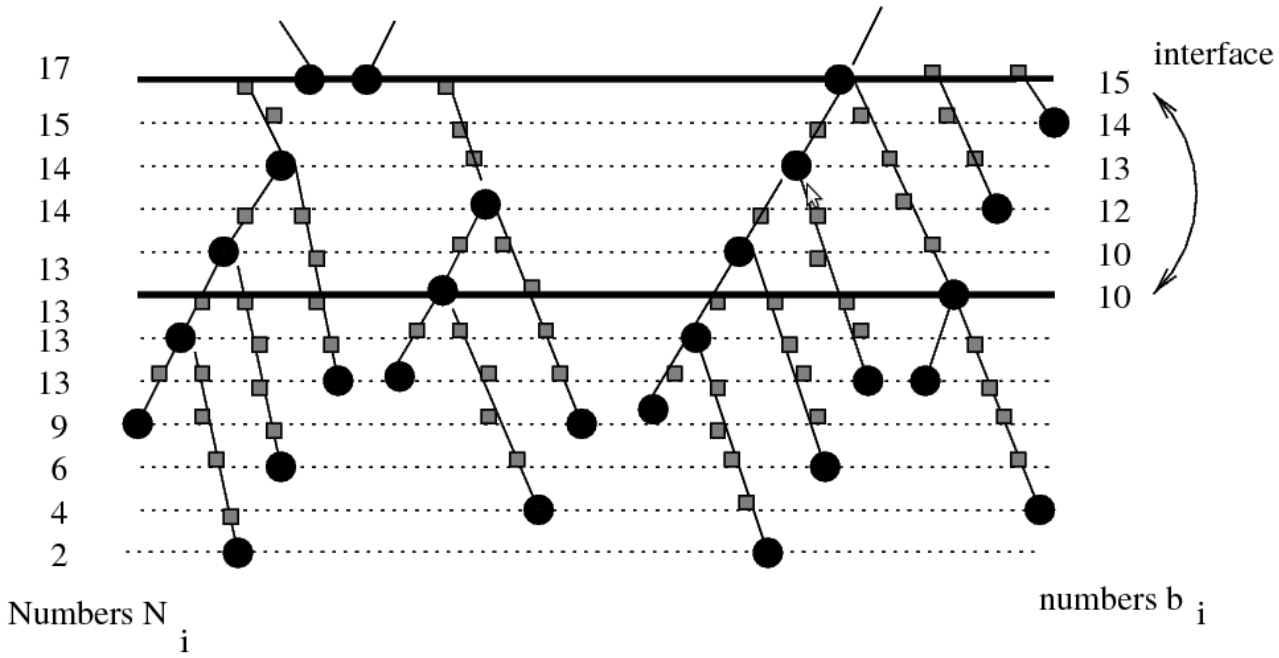
- Ile informacji da się wyciągnąć z takiego ciągu?
  - Znamy najgłębszy wierzchołek
  - Możemy obliczyć indeks jego brata w drzewie: $b_{\beta-\alpha-1}+1$

Algorytm $O(n^{C+1})$ dla $|W| = 2$



- Możemy również wyznaczyć liczbę liści poniżej pewnego poziomu:
  - $N_i = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$



- Możemy również wyznaczyć liczbę liści poniżej pewnego poziomu:
  - $N_i = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Dowód:
  - #liści = #drzew + #wierzchołków wewnętrznych
  - #wierzchołków wewnętrznych = $b_{i-\beta}$
  - #drzew = #korzeni = #korzeni będących lewymi synami + #korzeni będących prawymi synami = $(b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta})$.
  - #liści = $b_{i-\beta} + (b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta}) = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Dowód:
  - #liści = #drzew + #wierzchołków wewnętrznych
  - #wierzchołków wewnętrznych $= b_{i-\beta}$
  - #drzew = #korzeni = #korzeni będących lewymi synami + #korzeni będących prawymi synami = $(b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta})$.
  - #liści = $b_{i-\beta} + (b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta}) = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Dowód:
  - #liści = #drzew + #wierzchołków wewnętrznych
  - #wierzchołków wewnętrznych = $b_{i-\beta}$
  - #drzew = #korzeni = #korzeni będących lewymi synami + #korzeni będących prawymi synami = $(b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta})$.
  - #liści = $b_{i-\beta} + (b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta}) = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Dowód:
  - #liści = #drzew + #wierzchołków wewnętrznych
  - #wierzchołków wewnętrznych = $b_{i-\beta}$
  - #drzew = #korzeni = #korzeni będących lewymi synami + #korzeni będących prawymi synami = $(b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta})$.
  - #liści = $b_{i-\beta} + (b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta}) = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Dowód:
  - #liści = #drzew + #wierzchołków wewnętrznych
  - #wierzchołków wewnętrznych = $b_{i-\beta}$
  - #drzew = #korzeni = #korzeni będących lewymi synami + #korzeni będących prawymi synami = $(b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta})$.
  - #liści = $b_{i-\beta} + (b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta}) = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Koszt ciągu charakterystycznego

$\text{Koszt}(B) = \Sigma_{i=0}^{d-1} \Sigma_{j=0}^{N_i} p_j$

- Można pokazać, że koszt drzewa oraz koszt ciągu charakterystycznego odpowiadającego temu drzewu są sobie równe.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Koszt ciągu charakterystycznego

$\text{Koszt}(B) = \Sigma_{i=0}^{d-1} \Sigma_{j=0}^{N_i} p_j$

- Można pokazać, że koszt drzewa oraz koszt ciągu charakterystycznego odpowiadającego temu drzewu są sobie równe.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Definicja

Ciąg charakterystyczny jest **legalny**, gdy istnieje jakieś odpowiadające mu drzewo.

- Problem: nie każdy ciąg jest legalny.
- Można pokazać, że dla ciągu charakterystycznego **optymalnego** o pewnym koszcie, istnieje inny ciąg charakterystyczny o tym samym koszcie, który jest legalny.
  - Uwaga: nie oznacza to, że każdy ciąg optymalny jest legalny!
  - Dowód: indukcyjny.
  - Korzystamy ze "sklejania" pary najniższych synów.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Definicja

Ciąg charakterystyczny jest **legalny**, gdy istnieje jakieś odpowiadające mu drzewo.

- Problem: nie każdy ciąg jest legalny.
- Można pokazać, że dla ciągu charakterystycznego **optymalnego** o pewnym koszcie, istnieje inny ciąg charakterystyczny o tym samym koszcie, który jest legalny.
  - Uwaga: nie oznacza to, że każdy ciąg optymalny jest legalny!
  - Dowód: indukcyjny.
  - Korzystamy ze "sklejania" pary najniższych synów.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Definicja

Ciąg charakterystyczny jest **legalny**, gdy istnieje jakieś odpowiadające mu drzewo.

- Problem: nie każdy ciąg jest legalny.
- Można pokazać, że dla ciągu charakterystycznego **optymalnego** o pewnym koszcie, istnieje inny ciąg charakterystyczny o tym samym koszcie, który jest legalny.
  - Uwaga: nie oznacza to, że każdy ciąg optymalny jest legalny!
  - Dowód: indukcyjny.
  - Korzystamy ze "sklejania" pary najniższych synów.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Definicja

Ciąg charakterystyczny jest **legalny**, gdy istnieje jakieś odpowiadające mu drzewo.

- Problem: nie każdy ciąg jest legalny.
- Można pokazać, że dla ciągu charakterystycznego **optymalnego** o pewnym koszcie, istnieje inny ciąg charakterystyczny o tym samym koszcie, który jest legalny.
  - Uwaga: nie oznacza to, że każdy ciąg optymalny jest legalny!
  - Dowód: indukcyjny.
  - Korzystamy ze "sklejania" pary najniższych synów.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Definicja

Ciąg charakterystyczny jest **legalny**, gdy istnieje jakieś odpowiadające mu drzewo.

- Problem: nie każdy ciąg jest legalny.
- Można pokazać, że dla ciągu charakterystycznego **optymalnego** o pewnym koszcie, istnieje inny ciąg charakterystyczny o tym samym koszcie, który jest legalny.
  - Uwaga: nie oznacza to, że każdy ciąg optymalny jest legalny!
  - Dowód: indukcyjny.
  - Korzystamy ze "sklejania" pary najniższych synów.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Definicja

Ciąg charakterystyczny jest **legalny**, gdy istnieje jakieś odpowiadające mu drzewo.

- Problem: nie każdy ciąg jest legalny.
- Można pokazać, że dla ciągu charakterystycznego **optymalnego** o pewnym koszcie, istnieje inny ciąg charakterystyczny o tym samym koszcie, który jest legalny.
  - Uwaga: nie oznacza to, że każdy ciąg optymalny jest legalny!
  - Dowód: indukcyjny.
  - Korzystamy ze "sklejania" pary najniższych synów.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Przestrzeń stanów

β-krotki reprezentujące β ostatnich elementów ciągu charakterystycznego.

Krawędzie

Krawędzie prowadzimy między stanami postaci $(i_0, ..., i_{\beta-1})$ oraz $(i_1, ..., i_\beta)$.

Wagi krawędzi

$\text{Waga}(u \to v) = \text{Waga}(i_0, ..., i_\beta) = \Sigma_{k=0}^{N_\beta}$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Przestrzeń stanów

β-krotki reprezentujące β ostatnich elementów ciągu charakterystycznego.

Krawędzie

Krawędzie prowadzimy między stanami postaci $(i_0, ..., i_{\beta-1})$ oraz $(i_1, ..., i_\beta)$.

Wagi krawędzi

$$\text{Waga}(u \to v) = \text{Waga}(i_0, ..., i_\beta) = \Sigma_{k=0}^{N_\beta}$$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

Przestrzeń stanów

β-krotki reprezentujące β ostatnich elementów ciągu charakterystycznego.

Krawędzie

Krawędzie prowadzimy między stanami postaci $(i_0, ..., i_{\beta-1})$ oraz $(i_1, ..., i_\beta)$.

Wagi krawędzi

$$\text{Waga}(u \to v) = \text{Waga}(i_0, ..., i_\beta) = \Sigma_{k=0}^{N_\beta}$$

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Graf jest DAG-iem.
- Stan $(0, ..., 0)$ jest jedynym źródłem. Reprezentuje drzewa obcięte poniżej ostatniego wierzchołka.
- Stan $(n-1, ..., n-1)$ jest jedynym ujściem. Reprezentuje wszystkie drzewa z n liśćmi.
- Waga krawędzi odzwierciedla koszt przesunięcia linii obcięcia o jeden poziom w górę.
- Wniosek: najkrótsza ścieżka ze źródła do ujścia jest równa kosztowi optymalnego drzewa.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Graf jest DAG-iem.
- Stan $(0, \ldots, 0)$ jest jedynym źródłem. Reprezentuje drzewa obcięte poniżej ostatniego wierzchołka.
- Stan $(n-1, \ldots, n-1)$ jest jedynym ujściem. Reprezentuje wszystkie drzewa z n liśćmi.
- Waga krawędzi odzwierciedla koszt przesunięcia linii obcięcia o jeden poziom w górę.
- Wniosek: najkrótsza ścieżka ze źródła do ujścia jest równa kosztowi optymalnego drzewa.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Graf jest DAG-iem.
- Stan $(0, ..., 0)$ jest jedynym źródłem. Reprezentuje drzewa obcięte poniżej ostatniego wierzchołka.
- Stan $(n-1, ..., n-1)$ jest jedynym ujściem. Reprezentuje wszystkie drzewa z n liśćmi.
- Waga krawędzi odzwierciedla koszt przesunięcia linii obcięcia o jeden poziom w górę.
- Wniosek: najkrótsza ścieżka ze źródła do ujścia jest równa kosztowi optymalnego drzewa.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Graf jest DAG-iem.
- Stan $(0, ..., 0)$ jest jedynym źródłem. Reprezentuje drzewa obcięte poniżej ostatniego wierzchołka.
- Stan $(n-1, ..., n-1)$ jest jedynym ujściem. Reprezentuje wszystkie drzewa z n liśćmi.
- Waga krawędzi odzwierciedla koszt przesunięcia linii obcięcia o jeden poziom w górę.
- Wniosek: najkrótsza ścieżka ze źródła do ujścia jest równa kosztowi optymalnego drzewa.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Graf jest DAG-iem.
- Stan $(0, ..., 0)$ jest jedynym źródłem. Reprezentuje drzewa obcięte poniżej ostatniego wierzchołka.
- Stan $(n-1, ..., n-1)$ jest jedynym ujściem. Reprezentuje wszystkie drzewa z n liśćmi.
- Waga krawędzi odzwierciedla koszt przesunięcia linii obcięcia o jeden poziom w górę.
- Wniosek: najkrótsza ścieżka ze źródła do ujścia jest równa kosztowi optymalnego drzewa.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Analiza złożoności: Mamy $O(n^C)$ stanów do przeszukania.
- Przetworzenie pojedynczego stanu wymaga rozpatrzenia n możliwych krawędzi z niego wychodzacych.
- Po przemnożeniu otrzymujemy $O(n^{C+1})$.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Analiza złożoności: Mamy $O(n^C)$ stanów do przeszukania.
- Przetworzenie pojedynczego stanu wymaga rozpatrzenia n możliwych krawędzi z niego wychodzacych.
- Po przemnożeniu otrzymujemy $O(n^{C+1})$.

Algorytm $O(n^{C+1})$ dla $|W| = 2$

- Analiza złożoności: Mamy $O(n^C)$ stanów do przeszukania.
- Przetworzenie pojedynczego stanu wymaga rozpatrzenia n możliwych krawędzi z niego wychodzacych.
- Po przemnożeniu otrzymujemy $O(n^{C+1})$.

Algorytm $O(n^C)$ dla $|W| = 2$

Macierz kosztów

Dla ustalonego $\delta = (i_1, ..., i_{\beta-1})$ definiujemy macierz A_δ zadaną wzorem:

$A_\delta(i,j) = Waga(i, \delta, j) + Koszt(i, \delta)$

- j-ty wiersz macierzy zawiera wszystkie możliwe koszty dotarcia do stanu $(i_1, ..., i_\beta, j)$.

Algorytm $O(n^C)$ dla $|W| = 2$

Macierz kosztów

Dla ustalonego $\delta = (i_1, ..., i_{\beta-1})$ definiujemy macierz A_δ zadaną wzorem:
$A_\delta(i,j) = Waga(i, \delta, j) + Koszt(i, \delta)$

- j-ty wiersz macierzy zawiera wszystkie możliwe koszty dotarcia do stanu $(i_1, ..., i_\beta, j)$.

Algorytm $O(n^C)$ dla $|W| = 2$

- Można pokazać, że macierz ma tzw. własność Monge'go:
  - $A_\delta(i,j) + A_\delta(i+1,j+1) \leq A_\delta(i+1,j) + A_\delta(i,j+1)$
- Takie macierze mają inną własność, tzw. całkowitą monotoniczność:
  - $A_\delta(i,j+1) \leq A_{i+1,j+1} \implies A_\delta(i,j) \leq A_\delta(i+1,j)$.
- Dla takich macierzy działa tzw. algorytm SMAWK wyznaczający minima wszystkich wierszy w czasie $O(n)$.
  - Intuicja: pojedyncze porównanie może nam dać informację na temat liniowej liczby nierówności w macierzy.

Algorytm $O(n^C)$ dla $|W| = 2$

- Można pokazać, że macierz ma tzw. własność Monge'go:
  - $A_\delta(i,j) + A_\delta(i+1,j+1) \leq A_\delta(i+1,j) + A_\delta(i,j+1)$
- Takie macierze mają inną własność, tzw. całkowitą monotoniczność:
  - $A_\delta(i,j+1) \leq A_{i+1,j+1} \implies A_\delta(i,j) \leq A_\delta(i+1,j)$.
- Dla takich macierzy działa tzw. algorytm SMAWK wyznaczający minima wszystkich wierszy w czasie $O(n)$.
  - Intuicja: pojedyncze porównanie może nam dać informację na temat liniowej liczby nierówności w macierzy.

Algorytm $O(n^C)$ dla $|W| = 2$

- Można pokazać, że macierz ma tzw. własność Monge'go:
  - $A_\delta(i,j) + A_\delta(i+1,j+1) \leq A_\delta(i+1,j) + A_\delta(i,j+1)$
- Takie macierze mają inną własność, tzw. całkowitą monotoniczność:
  - $A_\delta(i,j+1) \leq A_{i+1,j+1} \implies A_\delta(i,j) \leq A_\delta(i+1,j)$.
- Dla takich macierzy działa tzw. algorytm SMAWK wyznaczający minima wszystkich wierszy w czasie $O(n)$.
  - Intuicja: pojedyncze porównanie może nam dać informację na temat liniowej liczby nierówności w macierzy.

Algorytm $O(n^C)$ dla $|W| = 2$

- Można pokazać, że macierz ma tzw. własność Monge'go:
  - $A_\delta(i,j) + A_\delta(i+1,j+1) \leq A_\delta(i+1,j) + A_\delta(i,j+1)$
- Takie macierze mają inną własność, tzw. całkowitą monotoniczność:
  - $A_\delta(i,j+1) \leq A_{i+1,j+1} \implies A_\delta(i,j) \leq A_\delta(i+1,j)$.
- Dla takich macierzy działa tzw. algorytm SMAWK wyznaczający minima wszystkich wierszy w czasie $O(n)$.
  - Intuicja: pojedyncze porównanie może nam dać informację na temat liniowej liczby nierówności w macierzy.

Plan prezentacji

Modyfikacje

- Ograniczenie wysokości drzewa.
- Wymóg, aby symbole były kodowane za pomocą słów z zadanego języka regularnego.

Modyfikacje

- Ograniczenie wysokości drzewa.
- Wymóg, aby symbole były kodowane za pomocą słów z zadanego języka regularnego.

Plan prezentacji

Zastosowania

- Nośniki fizyczne, w których z powodów technologicznych ograniczeń każdy bit 1 musi być poprzedzony co najmniej a zerami i co najwyżej b bitami 0.
- Alfabet Morse'a: $\text{length}(\cdot) = 1, \text{length}(-) = 2$.
- Drzewa decyzyjne, w którym podjęcie różnych decyzji wiąże się z różnym kosztem.

Zastosowania

- Nośniki fizyczne, w których z powodów technologicznych ograniczeń każdy bit 1 musi być poprzedzony co najmniej a zerami i co najwyżej b bitami 0.
- Alfabet Morse'a: $\text{length}(\cdot) = 1, \text{length}(-) = 2$.
- Drzewa decyzyjne, w którym podjęcie różnych decyzji wiąże się z różnym kosztem.

Zastosowania

- Nośniki fizyczne, w których z powodów technologicznych ograniczeń każdy bit 1 musi być poprzedzony co najmniej a zerami i co najwyżej b bitami 0.
- Alfabet Morse'a: $\text{length}(\cdot) = 1, \text{length}(-) = 2$.
- Drzewa decyzyjne, w którym podjęcie różnych decyzji wiąże się z różnym kosztem.

Źródła I

- **Mordecai J. Golin, Günter Rote,** A Dynamic Programming Algorithm for Constructing Optimal Prefix-Free Codes for Unequal Letter Costs, *IEEE Transactions on Information Theory, 1998*
- **Phil Bradford, Mordecai J. Golin, Lawrence L. Larmore, Wojciech Rytter,** Optimal Prefix-Free Codes for Unequal Letter Costs and Dynamic Programming with the Monge Property, *Journal of Algorithms, 2000*
- **Vicky Choi, Mordecai J. Golin,** Lopsided Trees: Analyses, Algorithms, and Applications, *in Automata, Languages and Programming, Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP), 2000*