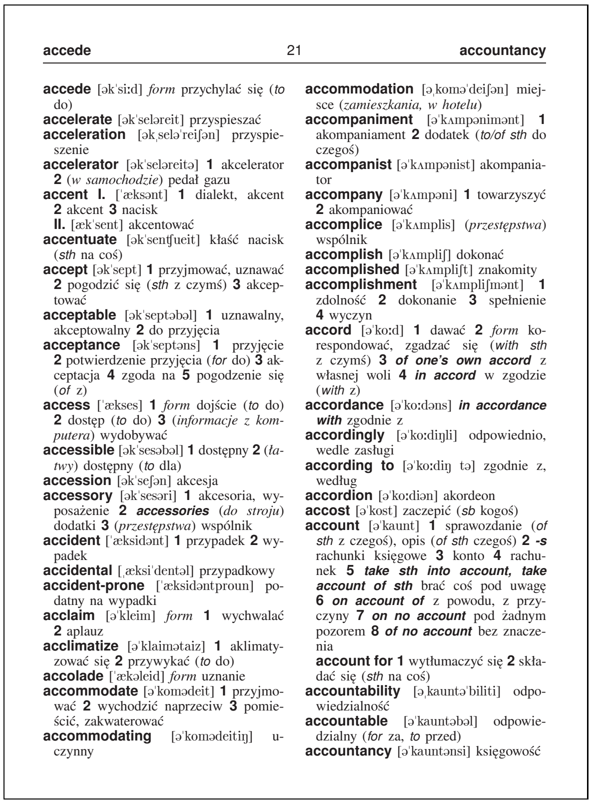
O prostym przypadku globalnej optymalizacji łamania

Marcin Woliński

W niniejszym artykule opowiem o tym, jak łamałem słownik [3]. Można powiedzieć, że dzieło to przedstawia sobą prosty przypadek trudnego łamania (zob. rys. 1). Składa się na nie mianowicie 700 stron małego formatu w układzie dwułamowym (a więc łamy są wyjątkowo wąskie), tekst jest sztywny – w ogóle nie występują rozciągliwe odstępy pionowe. Oczywiście skład ma być zgodny z regułami sztuki, a więc wszystkie kolumny muszą być tej samej wysokości i niedopuszczalne są szewce (pozostawianie u dołu kolumny pierwszego wiersza akapitu) i bękarty (umieszczanie ostatniego wiersza akapitu na początku kolumny).

Wynika stąd, że jedyną metodą wpływania na sposób łamania jest dorabianie i gubienie wierszy w wybranych akapitach poprzez manipulację odstępami międzywyrazowymi.



accede 21 **accountancy**

accede [əkˈsiːd] *form* przychylać się (*to* do)
accelerate [əkˈseləreit] przyspieszać
acceleration [əkˌseləˈreiʃən] przyspieszenie
accelerator [əkˈseləreitə] **1** akcelerator **2** (*w samochodzie*) pedał gazu
accent I. [ˈæksənt] **1** dialekt, akcent **2** akcent **3** nacisk
II. [ækˈsent] akcentować
accentuate [əkˈsentʃueit] kłaść nacisk (*sth* na coś)
accept [əkˈsept] **1** przyjmować, uznawać **2** pogodzić się (*sth* z czymś) **3** akceptować
acceptable [əkˈseptəbəl] **1** uznawalny, akceptowalny **2** do przyjęcia
acceptance [əkˈseptəns] **1** przyjęcie **2** potwierdzenie przyjęcia (*for* do) **3** akceptacja **4** zgoda na **5** pogodzenie się (*of* z)
access [ˈækses] **1** *form* dojście (*to* do) **2** dostęp (*to* do) **3** (*informacje z komputera*) wydobywać
accessible [əkˈsesəbəl] **1** dostępny **2** (*łatwy*) dostępny (*to* dla)
accession [əkˈseʃən] akcesja
accessory [əkˈsesəri] **1** akcesoria, wyposażenie **2** ***accessories*** (*do stroju*) dodatki **3** (*przestępstwa*) wspólnik
accident [ˈæksidənt] **1** przypadek **2** wypadek
accidental [ˌæksiˈdentəl] przypadkowy
accident-prone [ˈæksidəntproun] podatny na wypadki
acclaim [əˈkleim] *form* **1** wychwalać **2** aplauz
acclimatize [əˈklaimətaiz] **1** aklimatyzować się **2** przywykać (*to* do)
accolade [ˈækəleid] *form* uznanie
accommodate [əˈkomədeit] **1** przyjmować **2** wychodzić naprzeciw **3** pomieścić, zakwaterować
accommodating [əˈkomədeitiŋ] uczynny
accommodation [əˌkoməˈdeiʃən] miejsce (*zamieszkania, w hotelu*)
accompaniment [əˈkʌmpənimənt] **1** akompaniament **2** dodatek (*to/of sth* do czegoś)
accompanist [əˈkʌmpənist] akompaniator
accompany [əˈkʌmpəni] **1** towarzyszyć **2** akompaniować
accomplice [əˈkʌmplis] (*przestępstwa*) wspólnik
accomplish [əˈkʌmpliʃ] dokonać
accomplished [əˈkʌmpliʃt] znakomity
accomplishment [əˈkʌmpliʃmənt] **1** zdolność **2** dokonanie **3** spełnienie **4** wyczyn
accord [əˈkoːd] **1** dawać **2** *form* korespondować, zgadzać się (*with sth* z czymś) **3** ***of one's own accord*** z własnej woli **4** ***in accord*** w zgodzie (*with* z)
accordance [əˈkoːdəns] ***in accordance with*** zgodnie z
accordingly [əˈkoːdiŋli] odpowiednio, wedle zasługi
according to [əˈkoːdiŋ tə] zgodnie z, według
accordion [əˈkoːdiən] akordeon
accost [əˈkost] zaczepić (*sb* kogoś)
account [əˈkaunt] **1** sprawozdanie (*of sth* z czegoś), opis (*of sth* czegoś) **2** ***-s*** rachunki księgowe **3** konto **4** rachunek **5** ***take sth into account, take account of sth*** brać coś pod uwagę **6** ***on account of*** z powodu, z przyczyny **7** ***on no account*** pod żadnym pozorem **8** ***of no account*** bez znaczenia
account for **1** wytłumaczyć się **2** składać się (*sth* na coś)
accountability [əˌkauntəˈbiliti] odpowiedzialność
accountable [əˈkauntəbəl] odpowiedzialny (*for* za, *to* przed)
accountancy [əˈkauntənsi] księgowość

Rysunek 1: Typowa strona słownika [3]

Tak postawione zadanie przedstawia się dosyć beznadziejnie. Wydaje się bowiem wymagać prób dorabiania lub gubienia wierszy połączonych z wielokrotnym przeskładywaniem tekstu. Jest to zadanie, do którego TeX wyraźnie się nie nadaje.

Ponieważ jestem wyznawcą podstawowej cnoty programistów, czyli lenistwa, zadałem sobie pytanie, czy ten proces da się zautomatyzować.

Szczęśliwie rozpatrywany przypadek jest jednocześnie dość prosty: wszystkie kolumny są równej wysokości, łamy równej szerokości, a wszystkie wiersze tekstowe są równej wysokości. Oznacza to, że łamy i kolumny można rozliczać w wierszach a nie w punktach typograficznych, co biorąc pod uwagę brak kleju, pozwala w prosty sposób opisywać dostępne i zajęte miejsce na kolumnie. Najważniejsze jednak, że w tekście nie występują obiekty ruchome (*floats*), co oznacza, że cały materiał musi znaleźć się w składzie dokładnie w zadanym porządku.

TeX ma wbudowany mechanizm dorabiania i gubienia wierszy. Mechanizmem tym steruje się nadając odpowiednie wartości parametrowi pierwotnemu `\looseness`. Knuth opisuje to następująco [2, s. 103]:

> Przy ustawieniu `\looseness=1` TeX będzie się starał wydłużyć akapit o jeden wiersz w stosunku do liczby optymalnej, o ile tylko da się to osiągnąć nie przekraczając zadanej tolerancji dla brzydoty poszczególnych wierszy. Podobnie jeśli `\looseness` wynosi 2, TeX postara się uzyskać o dwa wiersze więcej. Ustawienie `\looseness=-1` spowoduje próbę skrócenia akapitu. W ogólności, TeX najpierw składa akapit jak zwykle, a potem, jeśli liczba wierszy przy wyborze optymalnych punktów złamania wynosi n, a wartość parametru `\looseness` jest l, TeX wybiera punkty podziału tak, aby ostateczna liczba wierszy była tak bliska $n + l$, jak to tylko możliwe bez przekraczania tolerancji.

Niestety nie każdy akapit podda się działaniu parametru `\looseness`. Żeby zautomatyzować proces łamania trzeba więc w jakiś sposób uzyskać informację o podatności akapitów na dorabianie/gubienie wierszy. Proponowany sposób postępowania polega na wydobyciu od TeXa potrzebnej informacji, przetworzeniu jej zewnętrznym programem i wreszcie przekazaniu wyniku z powrotem do TeXa (w drugim przebiegu).

1. Krok 1: Co można wyczytać w pliku `.log`

Celem pierwszego etapu przetwarzania jest uzyskanie od TeXa informacji, jakie są możliwości złamania każdego z akapitów. Mówiąc ściślej, chcielibyśmy dla każdego akapitu znać wszystkie liczby wierszy, na które można go podzielić.

Na ile mi wiadomo, informacji tej nie można uzyskać na poziomie makr TEXowych. Jeżeli jednak nada się niezerową wartość parametrowi pierwotnemu `\tracingparagraphs`, to w pliku `.log` pojawi się dokładna informacja o sposobie podzielenia każdego akapitu. Jeżeli dodatkowo `\looseness` będzie miało na tyle dużą wartość, że TEX nie będzie w stanie dorobić żądanej liczby wierszy (np. 1000), to w pliku `.log` znajdzie się informacja o wszystkich możliwych sposobach połamania każdego akapitu (por. rys. 2). Będzie więc w szczególności można wydobyć informację o możliwych liczbach wierszy.

```
@@38: line 4.0- t=32694410 -> @@24
goś
@\par via @@23 b=0 p=-10000 d=100
@\par via @@20 b=0 p=-10000 d=100
@\par via @@21 b=0 p=-10000 d=100
@\par via @@16 b=87 p=-10000 d=24409
@@39: line 3.2- t=446 -> @@23
@\par via @@37 b=0 p=-10000 d=5100
@\par via @@35 b=0 p=-10000 d=10100
@\par via @@17 b=87 p=-10000 d=24409
@@40: line 4.2- t=27189 -> @@37
@\par via @@38 b=0 p=-10000 d=15100
@\par via @@36 b=0 p=-10000 d=10100
@@41: line 5.2- t=32709510 -> @@38
```

Rysunek 2: Fragment pliku `.log` odpowiadający końcowi pewnego akapitu. Linie zaczynające się `@@` informują o możliwych liczbach wierszy dla tego akapitu (liczby po słowie `line`).

W wyniku zastosowania takich ustawień otrzymuje się bezużyteczny plik `.dvi`, ale w pliku `.log` jest dostępna potrzebna informacja. Przy pomocy skryptu w języku Perl wydobywam z pliku możliwe liczby wierszy i odpowiadające im oceny brzydoty (*total demerits*) dla każdego akapitu.

Dla omawianego słownika tak uzyskany plik `.log` ma objętość około 150 MB. Tabela 1 zestawia liczby akapitów podatnych na działanie poszczególnych wartości parametru `\looseness` w obu częściach słownika [3]:

Wartość	Liczba akapitów	
`\looseness`	ang.-pol.	pol.-ang.
−1	157	117
0	18939	19394
1	2117	1207
2	60	22
3	7	3
4	2	

Tabela 1

Dane te mają oczywiście charakter przykładowy. Zależą one bowiem od szerokości łamów, wartości parametru `\tolerance`, rozciągliwości odstępów międzywyrazowych (a więc użytych fontów) i wreszcie od rozkładu długości akapitów występujących w dziele. Różnice w liczbach dla części angielsko-polskiej i polsko-angielskiej wynikają głównie z tego, że w pierwszej części artykuły hasłowe są zauważalnie dłuższe niż w drugiej.

2. Krok 2: Łamanie

TEX dokonuje podziału tekstu na strony przy pomocy algorytmu nazywanego żargonowo *best-fit*. Polega on na wybieraniu dla kolejnych stron, rozpatrywanych z osobna, podziału dającego najlepszy wygląd danej strony. To znaczy, że TEX nie zauważy np. sytuacji, gdy wybranie nieco gorszego złamania danej strony pozwoli znacznie poprawić łamanie strony lub stron następnych. W tym sensie algorytm działa lokalnie.

Takie rozwiązanie zostało przyjęte, ponieważ gdy powstawał TEX, nie można było sobie pozwolić na przechowywanie w pamięci roboczej komputera całego tekstu książki. Dla Knutha oczywiste były jednak zyski z podejścia globalnego i w analogicznym problemie dzielenia akapitów na wiersze TEX stosuje algorytm wyszukujący globalnie najlepszy sposób podziału. (Akapity są bowiem zwykle na tyle krótkie, że można było budować w pamięci odpowiednie struktury danych.)

Algorytm dzielenia akapitów na wiersze wykorzystuje technikę zwaną (myląco) *programowaniem dynamicznym*. Taką technikę można zastosować, gdy rozwiązanie pewnego problemu optymalizacji daje się opisać jako kombinacja optymalnych rozwiązań pewnych jego podproblemów. Przy składaniu rozwiązań zwykle wielokrotnie pojawia się potrzeba pobrania wyniku dla tych samych podproblemów. Za efektywność programowania dynamicznego odpowiada więc stablicowanie wyników dla podproblemów, pozwalające uniknąć wielokrotnych obliczeń.

Algorytm dzielenia akapitów daje się przenieść na łamanie stron, jak pokazał M. F. Plass w swojej pracy doktorskiej [4]. Plass nie zajmuje się jednak bezpośrednio przypadkiem, z którym ja musiałem się uporać. Co prawda wspomina nawet o możliwości uwzględnienia parametru `\looseness`, ale nie wskazuje, jak można by go włączyć w proces znajdowania optymalnego łamania [4, s. 66].

Zjawisko ulepszania łamania poprzez wybór jednego z wariantów tekstu (wariantów łamania) na poziomie akapitów najbardziej przypomina zachowanie polecenia `\discretionary`. Jest to jednak przypadek ogólniejszy – pozwala on wybierać warianty tekstu nie tylko na granicy linii (por. rozważania Knutha o skrótach [1, s. 1168]).

Ponieważ w przypadku, którym się zajmowałem, można było zapisać znacznie prościej część algorytmu związaną z rozliczaniem zajętego miejsca na stronie, wolałem sformułować własną wersję algorytmu od początku.

Omówię teraz algorytm łamania. Na początek rozpatrzmy uproszczoną wersję problemu ograniczając się do składanego jednołamowo ciągu akapitów z unikaniem szewców i bękartów. Jako dane wejściowe mamy długość kolumny wyrażoną w wierszach i listę akapitów z informacją o wszystkich dopuszczalnych liczbach wierszy i związanych z nimi wycenami „brzydoty” dla każdego z nich.

Szukamy wartości `\looseness` dla każdego z akapitów dającej optymalne łamanie. A więc takie, w którym nie występują szewce i bękarty i zastosowano możliwie „ładne” podziały akapitów na wiersze.

Biorąc pod uwagę podatność akapitów na dorabianie/gubienie wierszy, w wyniku połamania ciągu akapitów zwykle możemy osiągnąć różne długości składu, czyli skład może się kończyć w różnych *miejscach* na ostatniej stronie – w naszym wypadku wyraża się to różną liczbą zajętych wierszy. Tak rozumiane pojęcie miejsca jest kluczowe dla zrozumienia algorytmu.

Wyobraźmy sobie bowiem, że znamy wszystkie miejsca osiągalne w wyniku złamania na strony pewnej liczby akapitów. Załóżmy ponadto, że dla każdego z miejsc znamy najlepszy sposób osiągnięcia tego miejsca i związany z tym sposobem koszt. Jak się za chwilę okaże, na tej podstawie można wyznaczyć analogiczne dane dla tekstu o jeden akapit dłuższego, co znaczy, że problem dał się skutecznie rozbić na dwa prostsze podproblemy.

Istotnym elementem jest tu założenie, że wystarczy pamiętać najlepszy sposób osiągnięcia danego miejsca. Jeżeli koszt łamania jest sumą pewnych składników zależnych od poszczególnych akapitów, założenie to jest spełnione.

W definicji miejsca nie musimy uwzględniać numeru strony. Zakładamy bowiem (przynajmniej w tym uproszczonym przypadku), że każda osiągnięta liczba stron jest równie dobra, a zatem jedynym, co się liczy, jest liczba wierszy zajętych na ostatniej stronie i koszt osiągnięcia takiej liczby.

Rozważmy teraz zasadniczy krok algorytmu. Przypuśćmy, że mamy dany zbiór miejsc osiągalnych po k akapitach. Niech długość kolumny wynosi g wierszy.

Dla akapitu o numerze $k+1$ rozważamy wszystkie dopuszczalne sposoby podzielenia go na wiersze. Przypuśćmy, że możliwe jest m.in. podzielenie go na l wierszy przy brzydocie b. Wówczas dla każdego miejsca $\langle t, c\rangle$ osiągalnego po k akapitach (t – liczba wierszy na ostatniej stronie, c – dotychczasowy koszt) mamy następujące dwie możliwości:

- Jeżeli $t+l$ wierszy mieści się na stronie, po $k+1$ akapitach osiągalne jest miejsce $\langle t+l, c+b\rangle$.
- W przeciwnym przypadku, jeżeli przełamanie akapitu po $g-t$ wierszach jest dopuszczalne, osiągalne jest miejsce $\langle t+l-g, c+b+\delta\rangle$, gdzie δ wyraża dodatkowe grzywny za łamanie strony.

W wyniku takiego kroku otrzymujemy listę kandydatów na nowe najlepsze miejsca. Na liście tej mogą wystąpić elementy o takiej samej liczbie wierszy na ostatniej stronie. Do dalszych etapów wystarczy spośród nich wybrać ten o najmniejszym koszcie.

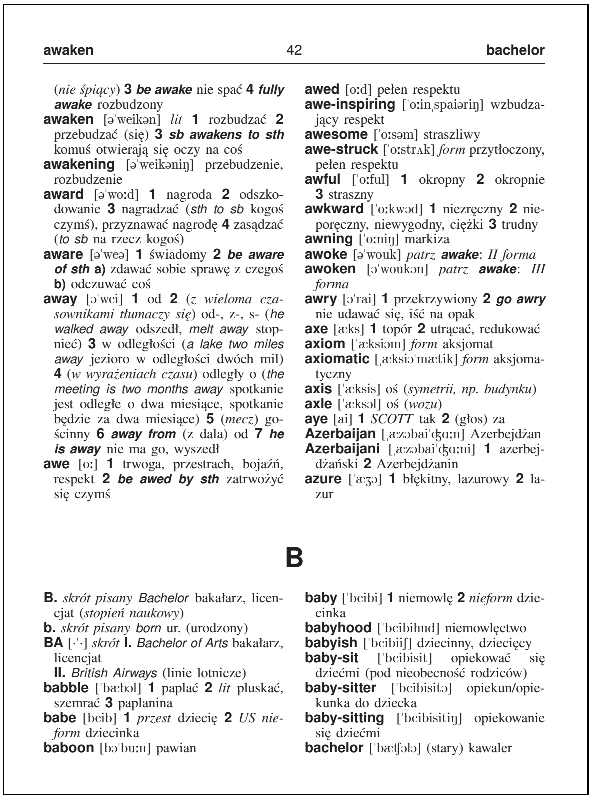
Jak widać, akapity podatne na `\looseness` mogą zwiększać liczbę dostępnych miejsc. Natomiast końce łamów mogą powodować eliminowanie miejsc przez to, że przyjęcie niektórych punktów łamania prowadziłoby do powstawania szewców lub bękartów.

Liczba miejsc rozpatrywanych dla każdego akapitu jest ograniczona z góry przez długość strony (w wierszach). To zapewnia zadowalającą efektywność działania algorytmu.

3. Krok 2: Wielołamowo

Do tej pory uporaliśmy się z uproszczonym przypadkiem składu jednołamowego. W prawdziwym świecie strona słownika może wyglądać na przykład tak, jak na rys. 3.

W przypadku większej liczby łamów na kolumnie łamy można optymalizować tak jak poprzednio.



awaken 42 **bachelor**

(*nie śpiący*) **3** ***be awake*** nie spać **4** ***fully awake*** rozbudzony
awaken [əˈweikən] *lit* **1** rozbudzać **2** przebudzać (się) **3** ***sb awakens to sth*** komuś otwierają się oczy na coś
awakening [əˈweikəniŋ] przebudzenie, rozbudzenie
award [əˈwoːd] **1** nagroda **2** odszkodowanie **3** nagradzać (*sth to sb* kogoś czymś), przyznawać nagrodę **4** zasądzać (*to sb* na rzecz kogoś)
aware [əˈweə] **1** świadomy **2** ***be aware of sth*** **a)** zdawać sobie sprawę z czegoś **b)** odczuwać coś
away [əˈwei] **1** od **2** (*z wieloma czasownikami tłumaczy się*) od-, z-, s- (*he walked away* odszedł, *melt away* stopnieć) **3** w odległości (*a lake two miles away* jezioro w odległości dwóch mil) **4** (*w wyrażeniach czasu*) odległy o (*the meeting is two months away* spotkanie jest odległe o dwa miesiące, spotkanie będzie za dwa miesiące) **5** (*mecz*) gościnny **6** ***away from*** (z dala) od **7** ***he is away*** nie ma go, wyszedł
awe [oː] **1** trwoga, przestrach, bojaźń, respekt **2** ***be awed by sth*** zatrwożyć się czymś
awed [oːd] pełen respektu
awe-inspiring [ˈoːinˌspaiəriŋ] wzbudzający respekt
awesome [ˈoːsəm] straszliwy
awe-struck [ˈoːstrʌk] *form* przytłoczony, pełen respektu
awful [ˈoːful] **1** okropny **2** okropnie **3** straszny
awkward [ˈoːkwəd] **1** niezręczny **2** nieporęczny, niewygodny, ciężki **3** trudny
awning [ˈoːniŋ] markiza
awoke [əˈwouk] *patrz* ***awake***: *II forma*
awoken [əˈwoukən] *patrz* ***awake***: *III forma*
awry [əˈrai] **1** przekrzywiony **2** ***go awry*** nie udawać się, iść na opak
axe [æks] **1** topór **2** utrącać, redukować
axiom [ˈæksiəm] *form* aksjomat
axiomatic [ˌæksiəˈmætik] *form* aksjomatyczny
axis [ˈæksis] oś (*symetrii, np. budynku*)
axle [ˈæksəl] oś (*wozu*)
aye [ai] **1** *SCOTT* tak **2** (głos) za
Azerbaijan [ˌæzəbaiˈʤɑːn] Azerbejdżan
Azerbaijani [ˌæzəbaiˈʤɑːni] **1** azerbejdżański **2** Azerbejdżanin
azure [ˈæʒə] **1** błękitny, lazurowy **2** lazur

B

B. *skrót pisany Bachelor* bakałarz, licencjat (*stopień naukowy*)
b. *skrót pisany born* ur. (urodzony)
BA [·ˈ·] *skrót* **I.** *Bachelor of Arts* bakałarz, licencjat **II.** *British Airways* (linie lotnicze)
babble [ˈbæbəl] **1** paplać **2** *lit* pluskać, szemrać **3** paplanina
babe [beib] **1** *przest* dziecię **2** *US nieform* dziecinka
baboon [bəˈbuːn] pawian
baby [ˈbeibi] **1** niemowlę **2** *nieform* dziecinka
babyhood [ˈbeibihud] niemowlęctwo
babyish [ˈbeibiiʃ] dziecinny, dziecięcy
baby-sit [ˈbeibisit] opiekować się dziećmi (pod nieobecność rodziców)
baby-sitter [ˈbeibisitə] opiekun/opiekunka do dziecka
baby-sitting [ˈbeibisitiŋ] opiekowanie się dziećmi
bachelor [ˈbætʃələ] (stary) kawaler

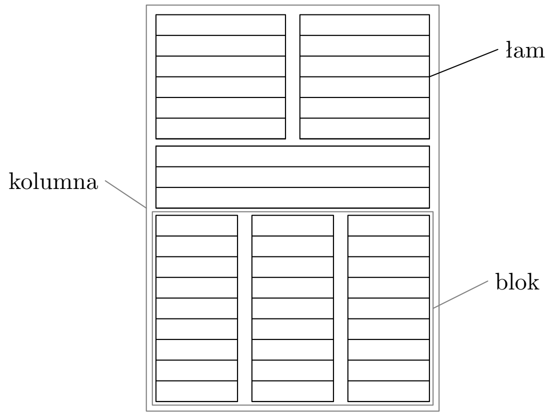
Rysunek 3: Bardziej skomplikowana strona słownika [3]

Problem jednak powodują zmiany liczby łamów wymuszające tworzenie łamów o wysokości mniejszej od wysokości kolumny. W takich bowiem sytuacjach trzeba spowodować, żeby tekst ściśle „wypełnił” wyznaczony obszar.

Ze względu na zmiany liczby łamów, w ogólnym przypadku możemy mieć do czynienia (por. rys. 4) z kolumną złożoną z bloków (których wysokości sumują się do wysokości kolumny) złożonych z potencjalnie różnych liczb łamów. Zaś problem postawiony w poprzednim akapicie polega na znalezieniu takiej wysokości bloku, którego zawartość jest ograniczona przez zmianę liczby łamów, aby uzyskać w tym bloku łamy równej wysokości. Chcemy przy tym oczywiście znaleźć najlepszy taki blok.

Żeby wyznaczyć właściwą długość bloku w momencie jego tworzenia trzeba by znać przyszłość, tj. wiedzieć, ile wierszy zajmą akapity, które zostaną w nim zawarte. Co gorsza, trzeba by zgadnąć dopuszczalne miejsca podziału tego bloku na łamy.

Można łatwo podać ograniczenia na liczbę dostępnych do wykorzystania wierszy. Otóż załóżmy, że uzupełnimy dane o akapitach o minimalną i maksymalną liczbę wierszy, które można uzyskać dla akapitów od bieżącego do najbliższego miejsca zmiany liczby łamów. (Tę informację możemy uzyskać czytając listę akapitów od tyłu i sumując odpowiednio najmniejsze i największe liczby wierszy dla poszczególnych akapitów).



Rysunek 4: Ogólny model kolumny słownika

Uzbrojeni w dodatkowe dane możemy zaproponować następującą procedurę: gdy przy przejściu do nowego bloku stwierdzimy, że jeszcze nie wykorzystanych wierszy może nie starczyć na wypełnienie strony, tworzymy alternatywne bloki, o długościach dopuszczonych przez wskazane wyżej ograniczenia. Gdy dotrzemy do końca, odrzucamy te, w których „wyszło” zbyt wiele lub zbyt mało wierszy. W ten sposób algorytm na kilkadziesiąt wierszy przed wymuszonym końcem bloku będzie miał do rozpatrzenia zauważalnie więcej przypadków. Jest to jednak cena, którą jesteśmy skłonni zapłacić. Po dotarciu bowiem do końca bloku większość hipotez zostanie odrzucona i dotychczasowe ograniczenie na liczbę rozpatrywanych miejsc znowu zacznie obowiązywać, zapewniając rozsądny czas działania algorytmu.

Ogólny schemat algorytmu pozostaje bez zmian, ale aby poradzić sobie z bardziej skomplikowaną strukturą kolumny trzeba uwzględnić nowe elementy w reprezentacji miejsc. Mianowicie miejsce to:

- liczba wierszy nad bieżącym blokiem,
- długość bieżącego bloku,
- numer bieżącego łamu,
- liczba wierszy w bieżącym łamie,
- koszt osiągnięcia danego miejsca.

Warto zauważyć, że nadal nie uwzględniamy numeru strony, nie interesuje nas, jaką strukturę (ile bloków) ma materiał ponad bieżącym blokiem, a tylko ile wierszy zostało zajęte. Zatem liczba rozpatrywanych miejsc dla każdego akapitu jest nadal ograniczona przez liczbę wierszy mieszczących się na kolumnie (we wszystkich blokach i łamach, czyli co najwyżej wysokość kolumny razy maksymalna liczba łamów).

Aby połamać słownik w zgodzie z wszelkimi regułami sztuki trzeba było uwzględnić w algorytmie jeszcze parę kwestii. Mianowicie:

- Pewne akapity muszą w całości mieścić się w obrębie łamu. Są to na przykład otwierające każdą literę słownika „akapity” z inicjałem wraz z otaczającymi go odstępami (rozliczone w wielokrotności wierszy).
- Nagłówki z inicjałem zajmują inną liczbę wierszy na górze kolumny niż wewnątrz.
- Są akapity, przed którymi lub po których nie wolno skończyć strony.
- Pierwszy i ostatni blok na kolumnie nie może być bardzo krótki (zapewne 6 wierszy jest rozsądnym ograniczeniem w tym przypadku).
- Ostatnia kolumna składu musi być rozsądnie wypełniona (tutaj ograniczenie powinno być pewnie silniejsze).

Wszystkie te szczegóły dają się prosto wyrazić jako dodatkowe ograniczenia powodujące odrzucanie przez algorytm pewnych miejsc.

4. Krok 3: TEXu, do roboty!

W dotychczasowym opisie nie wspomniałem, jak po zakończeniu algorytmu uzyskać opis optymalnego sposobu łamania. Dzieje się to tak samo, jak w TEXowym algorytmie dzielenia akapitów na wiersze.

Otóż z każdym miejscem wiążemy strukturę zawierającą wartość `\looseness` dla ostatniego akapitu przed danym miejscem i dowiązanie do takiej struktury opisującej poprzednie miejsce. Efektywnie każde miejsce zawiera listę wartości `\looseness` dla sposobu „rozciągnięcia” akapitów pozwalającego osiągnąć to miejsce. Części tych list dla różnych miejsc są jednak wspólne, liczba utworzonych struktur jest ograniczona przez liczbę wszystkich możliwych sposobów złamania wszystkich akapitów.

W istocie wystarczy, żeby listy te zawierały jedynie akapity, dla których trzeba było zastosować niezerową wartość parametru `\looseness`. Pozwala to wydatnie ograniczyć zapotrzebowanie algorytmu na pamięć operacyjną.

Po ustaleniu optymalnego rozwiązania odczytuje się odpowiadającą mu listę wartości `\looseness`. Wartości te są następnie czytane przez TEXa w drugim przebiegu, w którym następuje ostateczne łamanie. Ten etap jest stosunkowo prosty, jeśli uwzględni się pułapki opisane w artykule [5].

5. Ocena użyteczności i efektywności

Poniższa tabela prezentuje czas potrzebny na wykonanie opisanych powyżej kroków przetwarzania. Pomiaru dokonano na komputerze z procesorem Athlon 1,4 GHz pod systemem operacyjnym Linux. W fazie 1. i 3. wystąpił TEX w implementacji teTEX, a w drugiej interpreter języka Perl w wersji 5.6.

	ang.-pol.	pol.-ang.
I przebieg TEXa	37s	
łamanie	2m 52s	1m 54s
II przebieg TEXa	18s	

Drugi przebieg TEXa daje punkt odniesienia: tyle czasu zajmuje normalne przetworzenie książki. Pierwszy przebieg TEXa trwa około dwukrotnie dłużej – użycie niezerowego `\looseness` we wszystkich akapitach powoduje, że TEX ma do wykonania zauważalnie więcej pracy. Ponadto niezerowy parametr `\tracingparagraphs` spowodował konieczność wygenerowania gigantycznego pliku `.log` (czas zapisania tego pliku na dysku zapewne nie jest uwzględniony w danych, komputer miał bowiem wystarczająco dużo pamięci operacyjnej, aby wszystkie dane mieściły się jednocześnie w pamięci).

Mimo że liczba akapitów w obu częściach słownika jest podobna, łamanie pierwszej części trwało zauważalnie dłużej, niż drugiej. Wynika to z większej liczby akapitów podatnych na dorabianie/gubienie wierszy w tej części i tym samym większej liczby możliwości do rozpatrzenia.

Algorytm łamania został zapisany w języku Perl bez wielkiej dbałości o efektywność. Zapewne można go znacznie przyspieszyć wprowadzając odpowiednio przemyślaną strukturę danych do przechowywania listy rozpatrywanych miejsc (obecnie lista ta jest w każdym kroku sortowana dla usunięcia duplikatów). Takie usprawnienie może mieć sens, gdyż na przykład na komputerze z procesorem Pentium 200 MMX faza łamania trwa kilkadziesiąt minut.

Następna tabela zestawia rozwiązania znalezione przez algorytm:

	ang.-pol.	pol.-ang.
liczba rozwiązań	3	16
liczba wierszy na ost. stronie	8,10,13	6–41
liczba interwencji	508	276

Część angielsko-polska słownika zawiera niewiele haseł na litery X, Y i Z. To powoduje, że przed samym końcem łamania rozwiązania ulegają znacznemu przetrzebieniu i w końcowym rachunku pozostają tylko trzy możliwości. W części polsko-angielskiej otrzymuje się 16 rozwiązań dających od 6 do 41 wierszy na ostatniej kolumnie.

Liczba interwencji, czyli akapitów w których algorytm zaordynował zastosowanie niezerowej wartości `\looseness`, jest zbliżona do liczby interwencji, których dokonali ręcznie łamacze w poprzednim wydaniu. Należy jednak zwrócić uwagę, że interwencje ręczne były miejscami dokonywane przy pomocy bardziej brutalnych środków – wkładania tekstu w za ciasne pudełka, dokładania poleceń `\kern` itp.

W sumie dzięki kilku minutom pracy komputera uzyskano wynik wymagający dotychczas wielu godzin ręcznej pracy.

6. Rozszerzenia

Przedstawiony algorytm można na wiele sposobów rozszerzać. Po pierwsze można zliczać numery stron, na których wypadają dane miejsca. Dzięki temu można by na przykład wybierać rozwiązanie dające liczbę stron podzielną przez 8 lub 16. W tym celu należy uwzględnić numer bieżącej strony w strukturze reprezentującej miejsca i odpowiednio uaktualniać go w trakcie pracy algorytmu.

Ten prosty zabieg powoduje jednak, że liczba miejsc rozpatrywanych w każdym kroku przestaje być ograniczona przez liczbę wierszy na kolumnie. To sprawia, że czas łamania istotnie rośnie:

	ang.-pol.	pol.-ang.
I przebieg TEXa	37s	
łamanie	12m 31s	3m 36s
II przebieg TEXa	18s	

W tej wersji algorytmu dla części angielsko-polskiej słownika uzyskano takie same możliwe liczby wierszy na ostatniej kolumnie ale skład mógł liczyć od 412 do 424 kolumn. Daje to 13 możliwych długości składu, czyli na pewno da się dobrać liczbę stron podzielną przez 8.

Inne rozszerzenia mogły by obejmować wprowadzenie skokowo rozciąganego kleju pozwalającego na przykład na rozliczenie inicjału rozpoczynającego hasła na daną literę w pięciu lub sześciu wierszach. Zapewne należałoby zadbać, aby rozciągnięcie tego kleju było takie samo w obrębie dwóch padających na siebie stron.

Można by też wprowadzić możliwość skracania obu stron na rozwarciu o jeden wiersz. Poważniejszym rozszerzeniem byłoby wprowadzenie obiektów ruchomych i rozciągliwego kleju, czyli w zasadzie powrót do algorytmu Plassa rozszerzonego o wykorzystanie `\looseness`.

7. Podsumowanie

Przy pomocy technik programowania dynamicznego udało się całkowicie zautomatyzować najbardziej uciążliwą fazę pracy nad przygotowaniem słownika do druku. Tym samym TEX okazał się najlepszym narzędziem dla kolejnej (niewielkiej) klasy problemów. Wypada po raz kolejny wyrazić uznanie dla Autora systemu, że daje się on wykorzystać do wciąż nowych zadań. Nie wydaje mi się, żeby z któregokolwiek z konkurencyjnych programów dało się wydobyć informacje potrzebne jako dane dla opisanego tu algorytmu.

Pracy nad algorytmem łamania nie uważam za zakończoną. Przy składzie omawianego słownika byłem na etapie euforii wynikającej z faktu, że w ten sposób daje się znaleźć jakiekolwiek rozwiązania problemu. W dalszej pracy warto by rozważyć dokładniej kwestię, jaki sposób łamania należy uznać z optymalny. Na razie przyjąłem bowiem dość uproszczony model udziału brzydoty poszczególnych wariantów łamania akapitu w brzydocie całego składu (opiera się on jedynie na różnicy wartości *total demerits* względem najlepszego sposobu złamania danego akapitu). Warto przeprowadzić dokładniejsze studia tego zagadnienia. W szczególności opracować heurystyki, jakie warianty łamania akapitów można uznać za akceptowalne (znaleźć sposób ustalania sensownej wartości dla odpowiednika parametru `\tolerance`).

Literatura

[1] Donald E. Knuth, Michael F. Plass, *Breaking Paragraphs into Lines*, Software—Practice and Experience, 11, s. 1119–1184, 1981.

[2] Donald E. Knuth, *The TEXbook*, Addison-Wesley, 1986.

[3] Tadeusz Piotrowski, Zygmunt Saloni, *Nowy słownik angielsko-polski i polsko-angielski*, wydanie czwarte, Wilga, Warszawa 2002.

[4] Michael F. Plass, *Optimal Pagination Techniques for Automatic Typesetting Systems*, praca doktorska na Uniwersytecie Stanforda, 1981.

[5] Marcin Woliński, *Trochę za luźne \looseness*, Biuletyn GUST 17, s. 13–16.

◇ Marcin Woliński
wolinski@gust.org.pl